CALM
Common Assembly Language for Microprocessors

Manuel d'utilisation

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

Table des matières

Fichiers CALM

Vous devez trouver les fichiers suivants sur votre système d'exploitation (nous supposons qu'il s'agit d'un assembleur CALM pour le Z80):

ASCALM.*	assembleur CALM
MUFBIN.*	convertit le format binaire généré (MUFOM)
Z80.PRO	module pour le microprocesseur Z80
TZ80.ASM	fichier de test avec des instructions Z80
ASCALMER.TXT	liste d'erreurs pour l'assembleur

Configuration minimale:

Atari ST:	680x0, 256 Koctets libre, une disquette
PC/MS-DOS >= 2.x:	proc. comp. avec iAPX86, 256 Koctets, une disque.
PC/MS-DOS:	mettez dans CONFIG.SYS au minimum FILES=16.
DOS, TOS:	ASCALMER.TXT, *.PRO et *.REF: le PATH act. est util.; on peut changer celui-ci par SET CALM=répertoire(s).

Ligne de commande

L'assembleur est démarré par: ASCALM <fichier_source> [/option]

Les options sont facultatives. Les options à disposition sont:

/Apath1;path2;	ASCALM définit des "paths" add. pour ASCALM, c.à.d. pour ASCALMER.TXT, les *.PRO et *.REF. Exemple: DOS, TOS: PATH C:\ASCALM;C:\BIN; ligne de commande: /AA:\PRODEF; instruction: .PROC Z80 essaye d'ouvrir: 1) Z80.PRO 2) A:\PRODEF\Z80.PRO 3) C:\ASCALM\Z80.PRO 4) C:\BIN\Z80.PRO De même pour ASCALMER.TXT et les *.REF. Long. Limite pour tous les "paths" est de 80 caractères. Pour PROC A:Z80 , ASCALM ne cherche que A:Z80.PRO.
fichier/B .MUF	génère un objet (format MUFOM)
fichier/C .ASC	génère fichier.ASC: toutes les macros et les .IF/ .ELSE/.ENDIF sont remplacées. Pas possible si /E.
/Dnom=valeur	définit un symbole avec nom et valeur (+ ou -, déc.)
fichier/E .ERR	copie les messages d'erreurs dans ce fichier
/F	Fix n'enlève pas les symb. inutilisés chargés par .REF.
/Ipath1;path2;	Insère définit des "paths" add. pour .INS. Exemple: ligne de commande:/IA:\SOURCE;B:\DEF\;C:\PROJET; instruction: .INS IO_PART essaye d'ouvrir: 1) A:\SOURCE\IO_PART.ASM 2) B:\DEF\IO_PART.ASM 3) C:\PROJET\IO_PART.ASM 4) IO_PART.ASM Longueur maximale pour tous les "paths": 80 caract. Pour .INS B:IO_PART , ASCALM cherche B:IO_PART.ASM.
fichier/L .LST	génère un listage
/R	Read les fichiers sources ne sont pas modifiés
fichier/S .REF	contient tous les symb. util. sous forme d'assign.
fichier/SA .REF	comme /S sauf que adr. avec des ":"; +TRUE etc.
fichier/Si .REF	ne pas écrire quelques symboles dans .REF: sans valeurs (i=0), adresses (1), == (2) ou .SYSCALL (3).
/V	Verify ne génère pas d'objet (annule /B)
/W	Wait attend à chaque ligne avec une erreur
fichier/X .XRF	génère des ind. de réf. croisées (ajoutées au list.)

Si aucune option n'est spécifiée, l'assembleur génère uniquement le programme objet. L'option /B est nécessaire si l'on veut donner au fichier objet .MUF un autre nom ou le placer sur un autre

disque. Le fichier source peut être créé par PFED ou tout autre éditeur similaire. La longueur d'une ligne ne doit pas dépasser 128 caractères et doit finir par <CR><LF>.

A titre d'exemple, nous allons tester l'assembleur et son module: Nous allons assembler le fichier de test correspondant et générer un objet et un listage:

```
ASCALM TZ80/L
```

L'assembleur cherche le fichier TZ80.ASM et génère l'objet TZ80.MUF et le listage TZ80.LST. Si l'on veut assembler un fichier dans le disque B: et mettre l'objet et le listage dans le disque A:, il faut introduire:

```
ASCALM B:TZ80 A:TZ80/L/B      ou
ASCALM B:TZ80 A:TZ80/L A:TZ80/B
```

Fonctionnement

L'assembleur est démarré par la commande: ASCALM <source>/L

On demande aussi un listage. L'assembleur ouvre le fichier source et génère les fichiers d'objet (.MUF) et de listage (.LST). Le fichier source est lu deux fois. La génération de l'objet se fait dans la deuxième passe. Si des erreurs sont détectées dans le programme, la ligne d'erreur avec l'indication d'erreur est affichée dans le listage, dans le fichier d'erreur (si /E), à l'écran et dans le source

(s'il n'y a pas /R):

```
        MOVE    B,ALPHA         ; ligne dans un programme Z80
                  ^ 31
```

Pour cela, l'assembleur cherche au début de l'assemblage le fichier ASCALMER.TXT. S'il n'existe pas sur le disque courant, toutes les erreurs sont affichées par des nombres. Sinon, l'exemple précédent apparaît comme suit:

```
        MOVE    B,ALPHA         ; ligne dans un programme Z80
                  ^ symbol value undefined
```

Les touches suivantes sont interprétées lors d'un message d'erreur affiché:

"D" ne plus attendre après une erreur

"S" stopper le programme (cette commande est toujours possible)

"W" attendre après une erreur

Lors d'une erreur fatale (fichier n'existe pas, disque plein, etc.), l'assembleur s'arrête et affiche à l'écran les informations associées (ASCALM retourne 4 au système, sinon 0). L'utilisateur doit

corriger l'erreur et peut redémarrer l'assembleur.

Les fichiers d'E/S comme CON:, AUX:, etc. sont supportés.

La ligne de commande suivante dévie le listage à l'écran et ne génère pas d'objet: ASCALM TZ80 CON:/L/V

Important: L'assembleur fait une copie du fichier source afin d'insérer les messages d'erreurs. Prevoyez donc assez de place sur disque! Cette copie (.AST) avec les messages d'erreurs remplacera ensuite le fichier original. Avec l'option /R, l'assembleur ne modifiera pas le(s) fichier(s) source(s). Cette copie est aussi faite pour les fichiers insérés (.INS).

Caractères dans les noms des fichiers: "0".."9","A".."Z","a".."z", "_","?","-",":","\",".".

Liste des références croisées

Afin d'obtenir la liste des références croisées d'un programme, il faut indiquer l'option /X: ASCALM TZ80 B:TZ80/X

L'assembleur génère le fichier TZ80.XRF dans le disque B:. Plus il y a de symboles plus ce fichier sera volumineux. La liste des références croisées est écrite (ajoutée si /L) dans TZ80.LST. Le ficher TZ80.XRF sera détruit.

Format d'objet

L'assembleur CALM génère des fichiers objets avec l'extension .MUF. Ce format est un format ASCII. Les formats absolus (non translatables) sont uniquement générés. Pour convertir le format MUFOM en, par exemple, le format .COM, il faut utiliser MUFBIN:

MUFBIN <objet.MUF>/options

MUFBIN limite actuellement la longueur du fichier de sortie à 64 Koctets (exception: pas de limitations pour /B/N, /H/N, /I/U, /M/U (/N: si données consécutives).

Les options suivantes déterminent le format de sortie (rien n'est par défaut):

/B	.BIN, binaire, il faut aussi indiquer /N ou /Y
i/E	.EXE, pour PC/MS-DOS, i (1..3) définit la répart. de mém.
/H	.BIN, hexadécimal, ASCII, il faut aussi indiquer /N ou /Y
/I	.HEX, format hex d'Intel
/M ou i/M	.FRS, format S de Motorola (par défaut: S0, S3 et S7; 1/M: S0, S1 et S9; 2/M: S0, S2 et S8; 3/M: S0, S3 et S7)
/T	.TOS, pour Atari ST

Pour les formats de sortie /B et /H on peut insérer une entête devant le fichier original. Ainsi, ces fichiers sont composés d'une entête (d'une longueur de 256 octets) et de données binaires. L'entête contient les informations suivantes:

dépl.	contenu [déterminé par]
0	adresse de chargement [.LOC le plus bas avec du code gén.]
2	ongueur [code]
4	adresse de début [.START début]

L'ordre des octets des trois valeurs est: LSB-MSB.

Si vous ne voulez pas ajouter une entête, introduisez /N. Le fichier généré est aussi compatible avec un fichier .COM si l'adresse de début et de chargement sont 16'100. Et avec /Y vous pouvez ajouter une entête. D'autres options sont:

/A	déplace l'objet par une val. quelconque (nombre_hexadécimal/A).
/D	taille des données (utilisé avec /E; nombre_hexadécimal/D).
/F	effectue une opération ET entre la valeur de filtrage indiquée et les adresses (nombre_hexadécimal/F).
/J hh/J:	valeur pour les octets non-définis, par défaut: 00/J. Utilisez FF/J pour des EPROMs.
/L	fixe le nombre d'octets de données dans /H, /I et /M (gamme: 1 à 250; valeurs par défaut: 39, 32, 32; valeur/L). La commande /H 0/L ne généra pas de <CR><LF>.

/O	change le nom du fichier de sortie (nom_fichier/O).
/S	taille de la pile (utilisé avec /E; nombre_hexadécimal/S).
/U	ne remplit pas les zones non définies (utilisé avec /I ou /M).
/V	affiche toutes les inform. (sauf données) d'un fichier MUFOM.
/W	échange LSB et MSB dans un mot de 16 bits (word swap).

Exemples:

- changer le nom du fichier de sortie (p.ex. il faut générer le format d'Intel): MUFBIN objet_entrée/I objet_sortie/O. Les fichiers d'E/S (CON: ou AUX:) pour objet_sortie sont possibles.

- il faut placer les deux zones d'adresses 16'0 à 16'FFF et 16'F000 à 16'FFFF dans une EPROM de 8 Koctets. Par la commande: MUFBIN objet_entrée/B/N 1FFF/F un fichier binaire d'une longueur de 8 Koctets est généré qu'on peut directement utiliser pour la programmation de l'EPROM. Sinon le fichier aurait eu une longueur de 64 Koctets.

- dépl. un objet par une val. quelcon.: MUFBIN objet_entrée/I 200/A Si, par exemple, le fichier a été ass. avec .LOC 0, MUFBIN gén. un fichier en format .HEX d'Intel qui commence en 16'200. Le code objet n'a pas été modifié. Si /F a été indiqué en même temps, l'opération de filtrage est effectuée en premier.

Pseudo-instructions

L'assembleur CALM supporte uniquement les pseudo-instructions suivantes: .ALIGN, .APC, .ASCII, .ASCIZ, .ASCIZE, .BASE, .BLK.n, .CHAP, .DATA.n, .ELSE, .END, .ENDIF, .ENDLIST, .ENDMACRO, .ERROR, .EVEN, .EXITMACRO, .FILL.n, .IF, .INS, .LAYOUT, .LAYOUTMACRO, .LIST, .LISTIF, .LOC, .LOCALMACRO, .MACRO, .MESSAGE, .ODD, .PAGE, .PROC, .PROCSET, .PROCVAL, .RANGE, .REF, .START, .STRING, .SYSCALL, .TITLE, .8, .16 und .32 .

Remarques concernant quelques pseudo-instructions (voir aussi UPDATESF.*):

.ASCIZE
- correspond à .ASCIZ suivi de .EVEN (génère 0).

.IF/.ELSE/.ENDIF
- IF <expression> est vrai si <expression> <> zéro.
- IF..ENDIF peut être imbriqué jusqu'à 32 fois.
- IF et l'ENDIF corres. doivent se trouver dans le même fichier.
- IF..ELSE..ELSE..ENDIF est possible.

.INS

- Avec .INS fichier,READONLY le fichier inséré est seulement lu. Les messages d'erreurs ne seront pas insérés dans ce fichier. Ils sont ajoutés dans le fichier principal (si permis) ou le fichier d'erreur (si /E).
- extension par défaut: .ASM.
- on ne peut imbriquer .INS qu'une fois.

.LAYOUT

- les paramètres suivants sont possibles avec .LAYOUT:
HEX (adr. et données en représentation hexadécimale)
LENGTH n (n lignes par page de listage, n=0: infini)
Exemple: .LAYOUT HEX, LENGTH 60 ; valeurs de l'assembleur
- les paramètres suivants ne sont pas modifiables:
HEX (OCT n'est pas possible)
WIDTH 127 (longueur d'une ligne)
TAB 8 (un tabulateur correspond à 8 espaces)

.LIST/.ENDLIST
- LIST <expression> est vrai si <expression> est <> zéro.
- LIST..ENDLIST peut être imbriqué jusqu'à 255 fois.
- LIST/ENDLIST corres. doivent se trouver dans le même fichier.

.LISTIF <expression>
- montre toutes les pseudo-instr. .IF/.ELSE/.ENDIF dans le listage.
- .LISTIF est actif si <expr.> est <> zéro ou si <expr.> manque.

.REF fichier

- fichier.REF est un fichier de texte qui contient des assignat., des .SYSCALL et des commentaires. Le fichier n'est lu qu'une fois et n'est jamais modifié.
- le PATH actuel est respecté.

.SYSCALL.n nom (n = 8, 16 ou 32)
- définit une macro spéciale: .MACRO nom; .n nom%1; .ENDMACRO. Les SYSCALL sont admis dans les fichiers .REF. Exemple: INTDOS = 16'CD21; .SYSCALL.16 INT; appel: INT DOS; génère: .16 INTDOS.

Les pseudo-instructions suivantes ne sont pas supportées: .ENDTEXT, .EXPORT, .IMPORT, .TEXT

Différences assembleur CALM - standard CALM

L'assembleur CALM ne supporte pas le standard CALM complet. Les différences:

symbole:
- nom: 32 (étiquettes locales: 29) caractères signif.; caractères: "A".."Z", "a".."z", "_", "?" et "0".."9" (<> 1re position). Les accents sont convertis en maj.
- valeur: 32 bits avec signe.

expression:
- longueur de mot: 32 bits avec signe.
- le nombre maximal d'opérations ouvertes est de 15.
- ampl. de décalage (.SR., .SL. et .ASR.): l'ampl. De décalage est limitée à 8 bits (-256..+255). Une amplitude négative inverse la direction du décalage.

général:
- la longueur maximale d'une ligne de source est de 127.
- l'APC a une taille de 32 bits.
- quelques pseudo-instructions ne sont pas traitées.
- les commandes "\" ne sont pas supportées.
- plusieurs .PROC dans le même source ne sont pas poss.

Messages d'erreur

Voir ASCALMER.TXT. Les messages d'erreur en français se trouve en ASCALMEF.TXT. Ce fichier peut être copié dans ASCALMER.TXT.

Quelques remarques concernant les erreurs fatales:

101erreur dans .PROC
(Il y a une erreur dans la description du proc. On peut trouver l'instruction qui provoque cette erreur par l'option /D).

102 .PROC trop long
(pas assez de mémoire)

103 fichier n'existe pas
(Le fichier indiqué après la pseudo-instruction .PROC ou .INS n'existe pas.)

104 fichier d'entrée n'existe pas
(L'assembleur ne trouve pas le fichier source.)

105 fichier n'est pas créable.
(Le fichier objet ou le fichier listage ne peut pas être créé.)

106 pas de .PROC
(Il faut placer .PROC processeur au début du fichier.)

107 rebobinage du source ne va pas
(Le fichier de source est remis zéro au début de la seconde passe. Vérifiez le système.)

108 mauvaise version du .PROC
(L'assembleur et la description du proc. ne sont pas compat.)

109 ligne de commande vide
(Il faut indiquer le fichier source et les options sur la ligne de commande.)

110 nouveau symbole dans la deuxième passe
(Réassemblez si vous travailler sur un réseau. Essayez de trouver la ligne fautive avec l'option /D.)

111 erreur de pile dans .PRO
(Erreur fatale pendant l'interprétation de la description du processeur.)

112 table des symboles trop grande

113 fin du fichier: .ENDMACRO manque

114 place pour macro trop petite

115 trop de .INS imbriqués

116 stoppé
(L'assemblage a été arrêté par la touche "S".)

117 .PROC/.REF: doit précéder la génération de code
(.PROC et .REF doivent se trouver au début du fichier; après .TITLE.)

Inline avec l'assembleur CALM

TurboPascal et Pascal/MT+ permettent d'insérer des codes machines directement dans le source d'un programme Pascal grâce à l'instruction INLINE. Pour plus de détails concernant la syntaxe et les limites d'INLINE, consultez vos manuels d'utilisation de ces compilateurs Pascal.

L'assembleur CALM peut être utilisé pour générer ces codes machines. L'exemple suivant vous montre les différentes étappes nécessaires pour générer des instructions INLINE pour les processeurs iAPX86 (PC/MS-DOS) et Z80 (CP/M-80).

La fonction HEXNIBBLE teste et convertit le caractère d'entrée en un nombre si le caractère est un nombre hexadécimal ('0'..'9','A'..'F','a'..'f').

```
FUNCTION HEXNIBBLE(VAR H:INTEGER):BOOLEAN;
{in: ActCh, out: H (value), HEXNIBBLE (true or false)}
VAR C:CHAR;
  BEGIN
    C:=UpCase(ActCh); HEXNIBBLE:=TRUE;
    IF (C >= '0') AND (C <= '9')
    THEN BEGIN
      H:=ORD(C)-ORD('0');
    END
    ELSE
      IF (C >= 'A') AND (C <= 'F')
      THEN BEGIN
        H:=ORD(C)-ORD('A')+10;
      END
      ELSE BEGIN
        H:=0; HEXNIBBLE:=FALSE;
      END;
  END;
```

Maintenant il faut traduire cette procédure de Pascal en assembleur. Les deux pages suivantes vous montrent la version iAPX86 et Z80 de la fonction HEXNIBBLE. Pour cela, les instructions en assembleur sont écrites normalement dans un fichier source et assemblées (avec listage). Ensuite on détruit les lignes de Pascal dans la fonction HEXNIBBLE entre BEGIN et END et on insère le fichier listage après BEGIN. Vous effacez toutes les lignes d'assembleur superflues et les adresses (4 caractères au début de chaque ligne). Il aut placer ensuite les octets de listage générés sous la forme INLINE: commencer par INLINE(, placer des $ et / entre les octets,

etc. Mettez les instructions d'assembleur en commentaire (avec (* et *)). Finalement, il faut remplacer toutes les variables par leurs noms (ici: RESULT, HEX).

Il est d'ailleurs important que vous connaissez bien la représentation interne des différents types de données. Vous pouvez ainsi considérablement améliorer la version en assembleur (vitesse d'exécution, longueur du code). Par exemple, la valeur des variables booléennes est ici 1 pour TRUE et 0 pour FALSE. Il faut aussi savoir comment on peut accéder aux différentes variables. Le manuel d'utilisation de votre compilateur Pascal vous donne plus de détails.

Pour mesurer la vitesse de la version Pascal et la version assembleur, on a utilisé le programme suivant:

```
PROGRAM THEX;
VAR {teste la version en assembleur et en Pascal de HEXNIBBLE}
  ACTCH: CHAR;
  RESULT: BOOLEAN;
  I, VALUE: INTEGER;
{$I P_HEX} { P_HEX: Pascal, A_HEX: assembleur }
BEGIN
  WRITELN('START');
  FOR I:=1 TO 1000 DO  {1000 x}
  BEGIN
    FOR ACTCH:=' ' TO '~' DO  {95 caractères}
     RESULT:=HEXNIBBLE(VALUE);
    END;
  WRITELN('END');
END.
```

Les résultats suivants ont été obtenus:

	longueur du code	secondes (pour THEX)
version Pascal	192 octets	26,3
version assembleur	80 octets	20,6
différence	-58 %	-22 %

(compatible XT, horloge 4.77 MHz, TurboPascal 3.0 pour PC-DOS)

Ces nombres peuvent vous donner un ordre de grandeur. La version assembleur est souvent nettement plus rapide (2..4) que la version Pascal. Et la longueur du code de la version assembleur est presque toujours nettement plus courte.

La version iAPX86 (PC/MS-DOS), Listage:

```
0000                            .TITLE  HEXNIBBLE
0000                            .PROC   IAPX86
2710     00002710               .LOC    10000
2710                    RESULT:                 ; address > 8 bit:
2710                    HEX:                ; assembler gets 16 bit
0000     00000000               .LOC    0
0000                    HEXNIBBLE:
0000   8A861027                MOVE.8  [SS]+{BP}+RESULT,AL
0004   31C9                    XOR.16  CX,CX   ; CL = FALSE (=0),
0006   2C30                    SUB.8   #"0",AL ; CH = VALUE
0008   7211                    JUMP,LO END$
000A   3C09                    COMP.8  #9,AL
000C   760A                    JUMP,LS OK$
000E   2C07                    SUB.8   #"A"-"0"-10,AL
0010   3C0A                    COMP.8  #10,AL
0012   7207                    JUMP,LO END$
0014   3C0F                    COMP.8  #15,AL
0016   7703                    JUMP,HI END$
0018   88C5            OK$:    MOVE.8  AL,CH   ; nibble
001A   41                      INC.16  CX      ; CL = TRUE (=1)
001B   88AE1027        END$:   MOVE.8  CH,[SS]+{BP}+RESULT
001F   888E1027                MOVE.8  CL,[SS]+{BP}+HEX
```

Procédure:

```
FUNCTION HEXNIBBLE(VAR H:INTEGER):BOOLEAN;
{in: ActCh, out: H (value), HEXNIBBLE (true or false)}
VAR RESULT:INTEGER; HEX:BOOLEAN;
BEGIN
  RESULT:=ORD(UpCase(ActCh));
  INLINE(
  $8A/$86/RESULT/ (*        MOVE.8  [SS]+{BP}+RESULT,AL          *)
  $31/$C9/        (*        XOR.16  CX,CX;CL=FALSE (=0), CH=VALUE*)
  $2C/$30/        (*        SUB.8   #"0",AL                      *)
  $72/$11/        (*        JUMP,LO END$                         *)
  $3C/$09/        (*        COMP.8  #9,AL                        *)
  $76/$0A/        (*        JUMP,LS OK$                          *)
  $2C/$07/        (*        SUB.8   #"A"-"0"-10,AL               *)
  $3C/$0A/        (*        COMP.8  #10,AL                       *)
  $72/$07/        (*        JUMP,LO END$                         *)
  $3C/$0F/        (*        COMP.8  #15,AL                       *)
  $77/$03/        (*        JUMP,HI END$                         *)
  $88/$C5/        (*OK$:    MOVE.8  AL,CH   ; nibble             *)
  $41/            (*        INC.16  CX      ; CL = TRUE (=1)     *)
  $88/$AE/RESULT/ (*END$:   MOVE.8  CH,[SS]+{BP}+RESULT          *)
  $88/$8E/HEX);   (*        MOVE.8  CL,[SS]+{BP}+HEX             *)
  HEXNIBBLE:=HEX; H:=RESULT;
END;
```

La version Z80 (CP/M-80), Listage:

```
0000                            .TITLE  HEXNIBBLE
0000                            .PROC   Z80
2710     00002710               .LOC    10000
2710                    RESULT:
2710                    HEX:
0000     00000000               .LOC    0
0000                    HEXNIBBLE:
0000   3A1027                   MOVE    RESULT,A
0003   010000                   MOVE    #0,BC   ; C = FALSE (=0),
0006   D630                     SUB     #"0",A  ; B = VALUE
0008   3810                     JUMP,LO R8^END$
000A   FE0A                     COMP    #9+1,A
000C   380A                     JUMP,LO R8^OK$
000E   D607                     SUB     #"A"-"0"-10,A
0010   FE0A                     COMP    #10,A
0012   3806                     JUMP,LO R8^END$
0014   FE10                     COMP    #15+1,A
0016   3002                     JUMP,HS R8^END$
0018   47               OK$:    MOVE    A,B     ; nibble value
0019   0C                       INC     C       ; TRUE (=1)
001A   78               END$:   MOVE    B,A
001B   321027                   MOVE    A,RESULT
001E   79                       MOVE    C,A
001F   321027                   MOVE    A,HEX
```

Procédure:

```
FUNCTION HEXNIBBLE(VAR H:INTEGER):BOOLEAN;
{in: ActCh, out: H (value), HEXNIBBLE (true or false)}
VAR RESULT:INTEGER; HEX:BOOLEAN;
BEGIN
  RESULT:=ORD(UpCase(ActCh));
  INLINE(
  $3A/RESULT/     (*        MOVE    RESULT,A                      *)
  $01/$00/$00/    (*        MOVE    #0,BC  ;C=FALSE (=0), B=VALUE*)
  $D6/$30/        (*        SUB     #"0",A                        *)
  $38/$10/        (*        JUMP,LO R8^END$                       *)
  $FE/$0A/        (*        COMP    #9+1,A                        *)
  $38/$0A/        (*        JUMP,LO R8^OK$                        *)
  $D6/$07/        (*        SUB     #"A"-"0"-10,A                 *)
  $FE/$0A/        (*        COMP    #10,A                         *)
  $38/$06/        (*        JUMP,LO R8^END$                       *)
  $FE/$10/        (*        COMP    #15+1,A                       *)
  $30/$02/        (*        JUMP,HS R8^END$                       *)
  $47/            (*OK$:    MOVE    A,B     ; nibble value       *)
  $0C/            (*        INC     C       ; TRUE (=1)          *)
  $78/            (*END$:   MOVE    B,A                           *)
  $32/RESULT/     (*        MOVE    A,RESULT                      *)
  $79/            (*        MOVE    C,A                           *)
  $32/HEX);       (*        MOVE    A,HEX                         *)
  HEXNIBBLE:=HEX; H:=RESULT;
END;
```

Génération de programmes .EXE avec l'assembleur CALM

Le système d'exploitation PC/MS-DOS connaît deux types de programme: .COM et .EXE. Les programmes .COM sont des résidus du CP/M-80. Les segments de programme, de données et de pile sont dans une zone de mémoire 64 K. L'exécution du programme commence en 16'100. Et les quatre registres de segment du iAPX86 (CS, DS, ES et SS) ont tous la même valeur et pointent au début d'un segment de 64 K. Mais lors de l'exécution d'un programme .COM, toute la mémoire libre est occupée et pas seulement les 64 K.

Les programmes .EXE sont plus compliqués. Ils ont une entête qui contient des informations sur la longueur du code, les valeurs initiales du compteur d'adresse (CS:IP) et du pointeur de pile (SS:SP), etc.. Un programme .EXE peut avoir des segments séparés de programme, de données et de pile. La limitation à 64 K est ainsi supprimée. Et un programme .EXE occupe uniquement la place mémoire nécessaire.

Il est possible de générer des programmes .EXE avec l'assembleur CALM. Pour cela, le programmeur doit savoir où se trouvent les segments (programme, données, pile), comment ils ont été initialisés et comment il faut les accéder. La longueur du segment de programme est par contre limitée à 64 K (sans astuces). Et les segments de pile et de données peuvent aussi avoir individuellement 64 K (et à peu de frais une longueur illimitée).

La programmation de programmes .EXE avec l'assembleur CALM nécessite un certain soin. Par exemple, il n'est pas possible de charger un registre de segment (DS ou ES) avec une constante (p.ex. par l'intermédiaire d'AX avec une étiquette), car le programme est chargé à une autre adresse (inconnue). La translation (relogement) est assurée par le programmeur. Il initialise correctement les registres de segment et les utilise comme registre de base. Notez que ce style

de programmation est possible avec tous les assembleurs. Beaucoup de programmes .EXE ne nécessitent pas de translation avant l'exécution (la table de translation dans l'entête de .EXE est vide).

Pour générer des programmes .EXE, vous avez besoin de l'assembleur

CALM (ASCALM et MUFBIN), et le module de processeur iAPX86 (iAPX186, iAPX286).

L'assembleur CALM est aussi utilisé pour générer les programmes .COM. Dans les programmes .COM il ne faut jamais modifier les registres de segment et le programme commence par .LOC 16'100. Dans les programmes .EXE il faut au moins initialiser le segment de données. Avant l'exécution, le système initialise les segments CS (programme) et SS (pile). Les segments DS et ES (données) pointent sur le PSP (segment préfixe de programme). Le programme .EXE commence dans le fichier source toujours par .LOC 0. En plus, il faut spécifier une adresse de départ avec .START. Vous pouvez finalement assembler le fichier source et convertir l'objet à l'aide de MUFBIN fichier i/E en un fichier binaire .EXE spécifique. Par contre, les tailles de la pile et des données ne sont pas connues et il faut les spécifier.

MUFBIN supporte actuellement trois répartitions de mémoire. Ces trois possibilités dépendent de votre programmation. Vous trouvez trois exemples correspondants (TESTEXE1/2/3.ASM) avec le module iAPX86. Ces exemples vous donnent aussi plus de détails.

La ligne de commande pour MUFBIN a l'allure suivante:

MUFBIN fichier_d'entrée{.MUF} i/E {taille_de_la_pile/S} {taille_des_données/D}

Les indications entre accolades sont facultatives. i choisit une des trois répartitions de mémoire. /S et /D initialisent les tailles de la pile et des données (si nécessaire). Exemple (deuxième cas):

MUFBIN TESTEXE2 2/E 80/S 104/D

Utilisation d'étiquettes locales

Les étiquettes locales (p.ex. LOOP$) ne sont pas réellement différentes des étiquettes globales (p.ex. START). Mais les étiquettes locales n'apparaissent pas dans la liste des références croisées puisque leur signification est seulement locale. De plus, les étiquettes locales sont uniquement valides entre deux étiquettes globales. Pour ces raisons, les étiquettes locales sont utiliées de préférence dans des sous-programmes où ils fixent les points pour les boucles, les conditions et la sortie. Exemple:

```
TEXTHL:
PRINTC$ =       2               ; assignation locale
            PUSH    HL              ; in HL ^.ASCIZ
LOOP$:
            MOVE    {HL},A
            OR      A,A
            JUMP,EQ R8^END$         ; <NULL> trouvé ?
            PUSH    HL
            MOVE    A,E
            MOVE    #PRINTC$,C      ; montre le caractère à l'écran
            PUSH    HL
            CALL    BDOS
            POP     HL
            INC     HL
            JUMP    LOOP$
END$:
            POP     HL
            RET
```

LOOP$ et END$ sont aussi utilisables dans d'autres sous-programmes. Pour cette raison, on ne doit pas toujours inventer des nouveaux noms comme LOOP1, LOOP2, etc. De plus, seulement TEXTHL (le nom et le point d'entrée du sous-programme) apparaît dans la liste des références croisées.

Instructions de saut optimisées automatiquement

Il y a souvent deux possibilités d'adressage pour les instructions de saut dans plusieurs microprocesseurs 8 bit:

1) JUMP R8^étiquette (adr. relatif, APC-128..APC+127, 2 octets)
2) JUMP 16^étiquette (adr. absolu, 0..16'FFFF, 3 octets)

Si les indicateurs d'adresses R8^ et 16^ ne sont pas présents, l'assembleur CALM choisit automatiquement soit le cas 1) ou 2). Mais quand est-ce que le cas 1) est choisi?
Le cas 1) a deux avantages par rapport au cas 2): adressage relatif (indépendant de l'adresse) et un code machine plus court (2 au lieu 3 octets).

Considérons la situation suivante:

```
                ...
AVANT:
                ...
                JUMP    AVANT       ; (a)
                ...
                JUMP    APRES       ; (b)
                ...
APRES:
```

Nous remarquons que JUMP AVANT (a) saute à une étiquette qui est définie avant l'instruction de saut, et que JUMP APRES (b) saute à une étiquette qui est définie après l'instruction de saut.

L'assembleur CALM générera en (a) peut-être l'adressage relatif. Mais il faut que la distance de l'APC de l'instruction JUMP AVANT à l'étiquette AVANT soit inférieure à 129 octets. Si la distance est plus grande, l'assembleur CALM choisira l'adressage absolu.

L'assembleur CALM choisit en (b) toujours l'adressage absolu, même si APRES est moins loin que 127 octets de (b). Ceci est dû à la raison suivante: L'assembleur CALM lit le source deux fois (deux passes) pour générer le code machine. Dans la première passe, l'assembleur CALM rencontre en (b) l'étiquette inconnue APRES. L'assembleur CALM, ne connaissant pas la valeur, considère le cas pire (c.à.d. L'étiquette est très loin) et génère un saut absolu (3 octets). La première passe sert uniquement à calculer les adresses correctes. Dans la deuxième passe, l'assembleur CALM rencontre en (b) une deuxième fois l'étiquette APRES, cette fois-ci connue. Et, dans certains cas, la distance est réellement inférieure à 128, c.à.d. l'adressage relatif serait possible. Mais, c'est maintenant impossible: Si l'assembleur CALM choisirait ici une instruction de saut relatif (2 octets), alors toutes les adresses suivantes seraient changées d'un octet! Ainsi, l'assembleur CALM doit choisir en (b) la même instruction comme dans la première passe, c.à.d. une instruction de saut absolue. Si l'on veut forcer le cas 1) ou 2), il faut utiliser les indicateurs d'adresses R8^ et 16^. La même chose est valide pour les autres instructions optimisées automatiquement.

Extensions de l'assembleur CALM

L'assembleur CALM est un macroassembleur, c.à.d. qu'il traite aussi les macros.

Les définitions multiples dans les assignations et les étiquettes sont permises si les symboles ont le même nom et la même valeur (CR = 13; CR = 16'D).

Les assignations locales (A$ = 10) sont admises et ont les mêmes

possibilités que les étiquettes locales (A$:).

Les assignations multiples (A == 10) sont possibles et la valeur peut être modifiée (A == 20). Correspond à SET sur d'autres assembleurs. On ne peut pas mélanger les assignations normales (A = 10) avec les assignations multiples (A == 20).

La base des nombres peut être indiquée par des lettres:

```
16'nnnn  H'nnnn  X'nnnn  H'AF
10'nnnn  D'nnnn          D'100
 8'nnnn  O'nnnn  Q'nnnn  Q'377
 2'nnnn  B'nnnn          B'110
```

Fin du document.