1 Introduction

Les microprocesseurs existent déjà depuis plus de dix ans. Mais chaque microprocesseur possède son propre langage d'assemblage qui se différencie souvent considérablement des autres. Ce fait est curieux car tous les microprocesseurs se ressemblent - au moins du point de vue fonctionel - très fortement.

Mais il y a quelques années encore, on ne pouvait pas définir un langage d'assemblage uniforme, puisque chaque nouveau microprocesseur offrait des possibilités entièrement nouvelles (opérations, modes d'adressages). Cette évolution se ralentissait lors de l'ère des processeurs 16/32 bits car il était de plus en plus clair, ce qu'on attendait d'un microprocesseur moderne comme opérations et modes d'adressage. Pour cette raison, les micro-processeurs 16/32 bits ne se distinguent que de très peu. On constate une symétrie croissante des modes d'adressage, des longueurs et des types de données à l'intérieur d'une instruction.

Nous allons présenter ici un des langages d'assemblage universels. Son nom est CALM (Common Assembly Language for Microprocessors, en français: voir titre).

1.1 Qu'est-ce qu'est CALM ?

CALM n'est pas un nouveau langage de programmation mais une autre notation des instructions. Une longue expérience a montré que les instructions et leurs notations définies par CALM sont suffisantes pour décrire 95% des instructions d'un microprocesseur.

Le problème est en fait assez simple: Il faut trouver (inventer) une notation qui, d'une part, exprime d'une manière précise ce que fait le matériel, et, d'autre part, ne devient pas trop compliquée. CALM satisfait ces exigences dans presque tous les cas. Les exceptions sont dues à des opérations/modes d'adressage particuliers qui ne justifient pas une propre notation et sont notés d'une façon simplifiée. Mais on peut expliquer ces cas par une suite d'instructions en notation CALM. Il ne faut donc pas un méta-langage supplémentaire.

1.2 Avantages d'un langage d'assemblage uniforme

Actuellement, la programmation en assembleur est relativement embarassante et exige beaucoup de temps. Sans compter les exigences plus élevées de la programmation en assembleur en soi, il y a des difficultés artificielles: Chaque fabricant utilise d'autres termes et définit une notation spécifique des instructions. Cette différente terminologie trouble l'utilisateur, ne permet pas des comparaisons de processeurs et rend inutilement difficile les changements de processeur.

CALM définit une syntaxe uniforme pour les instructions et les pseudo-instructions. Ainsi, CALM crée une base commune pour la formation et la communication. Par exemple, un programme écrit en assembleur devient aussi pour ceux compréhensible qui ne connaissent pas le processeur pour lequel les instructions ont été écrit.

De plus, une notation uniforme permet une comparaison objective de micro-processeurs. Des tests de comparaison sont établis d'une manière plus efficace et plus simple puisque il ne faut pas apprendre un langage d'assemblage spécifique au processeur. Ceci est surtout intéressant pour les micro-processeurs avec compatibilité ascendante.

Finalement, une syntaxe uniforme simplifie l'assembleur. On peut construire un assembleur qui est constitué d'une partie principale indépendante du processeur et d'une partie spécifique au processeur. Dans la partie principale, les fonctions suivantes sont réalisées: traitement des symboles et des pseudo-instructions, évaluation des expressions, gestion des fichiers et des fonctions générales pour l'analyse de textes. Dans la partie spécifique au processeur, la syntaxe et la sémantique d'une instruction, ainsi que les règles pour générer le code objet sont définies.

1.3 Intérêt de la programmation en assembleur

On peut naturellement se demander si la programmation en assembleur est aujourd'hui encore utile. Matériel et logiciel de plus en plus pérformants répriment continuellement la programmation en assembleur. Mais la programmation en assembleur reste dans certains cas irremplaçable. Il faut citer les interfaces de matériel, les optimisations de tout genre, micro-ordinateur (sur une puce), unités microprogrammées, etc. Cependant, des spécialistes s'occupent de plus en plus de ces travaux et mettent à disposition un appel de sous-programme au niveau de programmation supérieur. Mais ne serait-il pas dans ces cas un avantage si l'on pouvait écrire ces quelques programmes en assembleur dans un langage d'assemblage uniforme?

1.4 Développement historique de CALM

L'existence de CALM est due indirectement au 8080. Quand Intel sortait ce microprocesseur en 1974, le langage d'assemblage associé semblait à beaucoup de gens trop primitif. De nombreuses propositions poussaient comme des champignons.

CALM a été défini à l'école polytechnique fédérale de Lausanne. Il prouvait sa capacité, entre autres, par sa consistence et son utilité fonctionelle à l'emploi pratique. Notamment, on pouvait exprimer aussi les autres processeurs 8 bits d'une façon satisfaisante.

Avec l'ère des processeurs 16/32 bits, CALM a été étendu et amélioré. CALM définissait avant tout une notation explicite pour les modes d'adressage et les indicateurs de données. En 1983, la version présente a été rédigé en collaboration avec DIN [1].

Parmi les efforts mondiaux de définir un langage d'assemblage commun, il faut mentionner le groupe de travail IEEE P694. Ce groupe commençait son travail en 1977 et publiait en 1984 la dix-huitième version. Pendant un certain temps, il y a eu un échange d'expérience intense avec CALM. Mais P694 restait trop fixé sur les architectures des microprocesseurs 4 et 8 bits et a surtout méconnu les problèmes d'une notation explicite des modes d'adressage. De plus, on est toujours d'avis qu'un langage d'assemblage doit être composé de noms les plus courts possibles et qu'avec chaque instruction il ne faut indiquer que le strict minimum.

2 Instructions

Une instruction exécute une action révolue, définie par le fabricant du microprocesseur. Le texte suivant illustre, comment une instruction est décomposée et comment ces parties séparées sont nommées et quelle notation est utilisée.

2.1 Composition d'une instruction

Une instruction est constituée d'un code opératoire et d'opérandes. A cela il faut ajouter selon la complexité des codes conditions, des indicateurs d'adresses et des indicateurs de données.

Cette composition modulaire d'une instruction et l'indépendance de la signification des parties séparées permettent des extensions quelconques. Mais ce concept impose que toutes les indications doivent être présentes même pour les instructions avec possibilités limitées.

CALM définit 49 instructions de base (fig. 1). Avec celles, on peut exprimer environ 95% des instructions d'un microprocesseur. Le reste est dû à des instructions spéciales où on réutilise le code opératoire du fabricant, mais on utilise la notation CALM pour les opérandes, le code condition, et les indicateurs d'adresses et de données.

L'instruction MOVE détermine l'ordre source -> destination (de gauche à droite). Il faut aussi garder cet ordre pour les autres instructions, par exemple pour l'instruction SUB.

Instructions de transfert

```
MOVE source,destination transfère de la source vers la destination.
PUSH source             empiler (équivalent à  MOVE source,{-SP}).
POP  destination        dépiler (équivalent à  MOVE {SP+},dest.).
CONV source,destination effectue une conversion de type de données.
CLR  destination        met à zéro l'op. de dest.(= MOVE #0,dest.).
SET  destination        init. l'op. de dest.:1s (= MOVE #-1,dest.).
EX   opérande1,opérande2  echange les deux opérandes.
SWAP sourcedestination  echange les deux moitiés de l'opérande.
```

Instructions arithmétiques

```
ADD  source,sourcedestination  additionne deux opérandes.
     source1,source2,destination
ADDC voir ADD           additionne deux opérandes et l'indicateur C
SUB  voir ADD           soustrait le premier opérande du second op.
SUBC voir ADD           2nd op. - 1er op. - l'indicateur C.
ACOC voir ADD           add. le compl. à un du 1er op., 1 et l'ind.
                        de report C au 2nd op. L'opération eff. est
                        éq. à SUBC mais les ind. sont modif. autre.
NEG  sourcedestination  calcule le complément vrai (compl. à 2 ou
     source,destination  soustr. de 0) de l'opérande de source.
NEGC voir NEG           calc. le compl. vrai avec l'ind. de report
                        C, c.à.d. soustrait l'op. et C de zéro.
MUL  source,sourcedestination  multiplie les deux op. de source.
     source1,source2,destination Si l'opérande de dest. n'est pas
                        explicitement précisé par sa taille de
                        données il n'est pas clair si la taille de
                        l'op. de destination est simple ou double.
DIV  diviseur,dividendequotient divise le 2nd op. par le premier.
     diviseur,dividende,quotient
     diviseur,dividende,quotient,reste
INC  sourcedestination  incrémente l'opérande de un.
DEC  sourcedestination  décrémente l'opérande de un.
COMP source1,source2    compare le second opérande au premier.
                        C'est la même opération que SUB, mais
                        il n'y a pas de résultat généré (uniquement
                        les indicateurs sont mis à jour).
CHECK limite_inf,limite_sup,source contrôle une val. contre 2 lim.
```

Instructions logiques

```
AND  source,sourcedestination effectue un ET logique bit à bit des
     source1,source2,destination  deux opérandes de source.
OR   voir AND           effectue un OU inclusif bit à bit des 2 op.
XOR  voir AND           effectue un OU exlusif bit à bit des 2 op.
NOT  sourcedestination  inverse chaque bit de l'opérande de source
     source,destination   (complément à un).
```

Instructions de décalage

```
SR   sourcedestination  eff. un déc. de l'op. de source à droite.
     amplitude,sourcedestination  Le MSB est remplacé par un 0 et
     amplitude,source,destination  l'ind. de report C par le LSB.
ASR  voir SR            effectue un déc. de l'op. de source avec
                        bit de signe à droite. Le MSB est conservé
                        et l'ind. de report C est rempl. par LSB.
SL   voir SR            eff. un déc. de l'op. de source à gauche.
                        Le LSB est rempl. par 0 et C par le MSB.
ASL  voir SR            effectue un décalage de l'op. de source
                        à gauche. Id. à SL mais V mod. autrement.
RR   voir SR            effectue une rot. de l'op. de source à
                        droite. MSB et C sont rempl. par le LSB.
RRC  voir SR            effectue une rot. de l'op. de source
                        avec C à droite. Le MSB est rempl. par C
                        et celui-ci par le LSB.
RL   voir SR            même effet que RR mais à gauche.
RLC  voir SR            même effet que RRC mais à gauche.
```

Fig. 1. Instructions CALM (1ère partie)

```
Instructions de test
TEST source              teste le signe et la valeur (si zéro) de
     source:adresse_de_bit  l'op. de source. Lors d'un adr. de bit,
                          uniquement l'ind. de nullité Z est modifié.
TCLR sourcedestination   teste et met à zéro le(s) bit(s) d'op.
     sourcedestination:adresse_de_bit
TSET voir TCLR           teste et met à un le(s) bit(s) d'opérande.
TNOT voir TCLR           teste et invertit le(s) bit(s) d'opérande.
Instructions de commande du flux d'instructions
JUMP adresse_de_saut     saute à l'adresse indiquée.
JUMP,cadresse_de_saut    saute à l'adr. si la cond. est satisfaite.
DJ,c sourcedestination,adresse_de_saut  déc. le 1er op. de 1 et
                          saute à l'adr. donnée par le 2nd op. si c.
SKIP,c                   saute l'instr. suiv. si la cond. est vraie.
CALL adresse_de_saut     appel de sous-prog. (sauve l'adr. de retour
CALL,cadresse_de_saut    sur la pile et saute à l'adresse indiquée).
RET                      retour de sous-programme (saute à l'adresse
RET,c                    récupérée de la pile).
TRAP expression          appel de sous-programme spécial.
TRAP,c
WAIT                     attend sur une interruption.
HALT                     arrête le processeur jusqu'à ce qu'il soit
                         réinitialisé par un signal externe.
RESET                    réinitialise les périphériques.
NOP                      instruction vide.
ION                      permet des interruptions.
IOFF                     interdit des interruptions.
Instructions spéciales
Toutes les instructions spéciales (p.ex.: DAA, LINK, FFS).
```

Fig. 1. Instructions CALM (2ème partie)

2.2 Code opératoire

Le code opératoire exprime l'opération effectuée. Le code opératoire est un nom dérivé d'un verbe anglais. Dans certains cas, on utilise aussi les abréviations et leurs combinaisons.

En général, on retrouve les codes opératoires définis par CALM aussi dans les langages d'assemblage des microprocesseurs individuels. Les seules différences concernent les codes opératoires dans lesquels le fabricant anticipe un mode d'adressage (p.ex. BRANCH, LEA, XLAT, ADDI). Par contre, on rencontre très souvent des différences orthographiques mineures: COMP au lieu CMP, MOVE au lieu MOV, etc.

2.3 Opérandes

Un opérande indique qui (registre, location mémoire, etc.) participe à une opération et comment on accède à l'information. L'opérande adresse cette information par l'intermédiaire d'une notation définie. Cette notation doit suffir à exprimer tous les modes d'adressage possibles (et utiles). Voir le chapitre 3.

Cette notation indépendante du processeur est jusqu'à présent unique. Il suffit pour s'en convaincre de comparer les notations du fabricant des modes d'adressage de quelques microprocesseurs. Un microprocesseur a généralement un nombre limité de modes d'adressage. Ceci conduit à une courte notation qui simplifie le travail de l'assembleur, mais un concept d'adressage n'est pas présenté.

Une notation pécise est aussi nécessaire parce qu'on ne peut plus intégrer le mode d'adressage dans le code opératoire. En plus, l'accumulateur n'est plus le centre unique d'un microprocesseur comme autrefois. Et des modes d'adressage de plus en plus pérformants ont besoin d'une indication précise.

2.4 Code condition

Un code condition exprime l'état d'un indicateur ou la combinaison d'états d'indicateurs. Les indicateurs sont modifiés par des

instructions. Si les instructions elles-mêmes dépendent des indicateurs, on ajoute un code condition séparé par une virgule au code opératoire. Les codes conditions sont des noms composés usuellement de 2 lettres (fig. 2).

```
   général:
     EQ  égal               NE non égal
     BS  bit à un           BC bit à zéro
     CS  report à un        CC report à zéro
     VS  dépassement à un   VC dépassement à zéro
     MI  moins, négatif     PL plus, positif
   après comparaison non signée:
     LO  inférieur          LS inférieur ou égal
     HI  supérieur          HS supérieur ou égal
   après comparaison arithmétique:
     LT  plus petit que     LE plus petit que ou égal
     GT  plus grand que     GE plus grand que ou égal
   divers:
     PE  parité paire       PO parité impaire
     NV  jamais             AL toujours          NMO non égal à -1
   Quelques codes conditions équivalents: EQ=BC, NE=BS, CS=LO, CC=HS.
```

Fig. 2. Codes conditions

Les indicateurs sont usuellement mémorisés dans le registre d'indicateurs F (fig. 3). D'autres indicateurs (I, S, T) peuvent exister et sont souvent mémorisés dans un registre d'état spécial S.

lettre/fonction	utilisation
C report	additionneur binaire et décalage
H report auxiliaire	report 4 bits (unique. pour les proc. 8 bits)
L lien	unique.si jamais utilisé comme ind. de report
N signe	à un: MSB à un (résultat négatif)
V dépassement	à un: dép. de capacité pour des nombr. arith.
Z zéro	à un: résultat nul
I interruption	à un: interruption active (permise)
S superviseur	à un: mode superviseur actif
T traçage	à un: mode de traçage actif

Fig. 3. Indicateurs

Les noms des codes conditions sont souvent identiques avec ceux des fabricants. Seulement la manière avec laquelle ils sont ajoutés au code opératoire est différente. Chez les fabricants, le code condition est directement concaténé avec le nom du code opératoire (une lettre). La notation de CALM est plus favorable, car un code opératoire quelconque peut être combiné avec un code condition quelconque et on peut distinguer les deux informations clairement (fig. 4).

```
           JUMP,NE adresse_de_saut
           RET,LO
           CALL,CS adresse_de_saut
           DJ.16,NE CX,adresse_de_saut
   SKIP,LO DJ.16,NMO D0,adresse_de_saut
```

Fig. 4. Exemples avec des codes conditions

2.5 Indicateurs de données

Les indicateurs de données ne sont devenus nécessaire qu'avec les microprocesseurs 16/32 bits. Il faut indiquer la taille et le type des données transférées avec ces microprocesseurs (fig. 5).

U ou rien	non signé ou pas spécifié
A	arithmétique (représentation complément à deux)
D	décimal (non signé, 2 chiffres par octet, BCD)
F	point flottant (format IEEE 754)
O	déplacement (2n-1 biais)
S	avec signe (signe et valeur absolue)

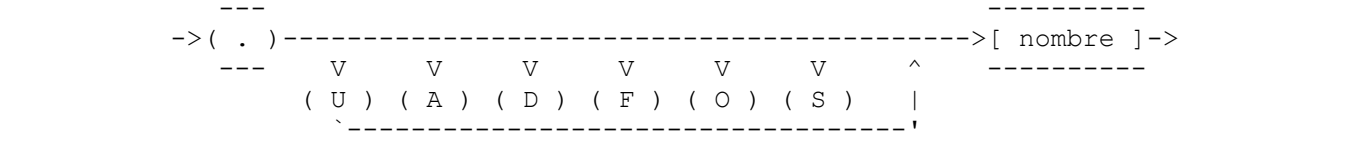

Fig. 5. Indicateurs de données

La taille de données indique le nombre de bits participant à une opération. L'indicateur est spécifié soit avec le code opératoire (taille de données valable pour tous les opérandes suivants) soit avec chaque opérande individuellement. La taille de données est directement exprimée en bits. Un point est utilisé comme séparateur.

Le type de données donne des informations sur les données utilisées. Cet indicateur est une lettre placée entre le point et la taille de données. Si le type de données n'est pas indiqué explicitement, on suppose qu'il s'agit de données non spécialement traitables. Sinon, un indicateur de type de données est nécessaire pour désigner une unité spécifique (p.ex. additionneur BCD ou FPU):

```
MOVE.32 R6,R1
ADD.D32 R5,R0
CONV    R1.A16,R2.F64
```

Fig. 6. Exemples avec des indicateurs de données

Les indicateurs de données sont aussi connus dans la notation de fabricant des instructions. Mais seulement la taille de données est notée séparément.

La taille de données est indiquée par une lettre ajoutée directement ou séparée par un point au code opératoire. Mais l'utilisation de lettres reste limitée: d'une part, il n'y a pas que les tailles de données de 8, 16 et 32 bits, et, d'autre part, tout le monde ne sera jamais d'accord sur leur assignation.

Aux types de données correspondent chez les fabricants différents codes opératoires, par exemple IMUL pour MUL.A16, ABCD pour ADDX.D8, NEGL pour NEG.F64.

2.6 Indicateurs d'adresses

Les indicateurs d'adresses sont devenus de plus en plus indispensables avec les possibilités d'adressage croissantes. Une adresse est composée plus souvent de sous-adresses qui n'ont pas la taille d'adresse entière. Il faut donc indiquer la taille de ces sous-adresses et préciser comment il faut les étendre à la taille d'adresse entière.

Tous ces informations sont fournies par les indicateurs d'adresses. Le programmeur peut bien comprendre sur papier ce qui se passe dans le microprocesseur, c.à.d. comment une adresse est construite. Il peut ainsi indiquer à l'assembleur le mode d'adressage à choisir.

U ou rien — non signé, extension de zéro
A — arithmétique, extension de signe
R — rel., valeur codée ajoutée avec ext. de signe au PC

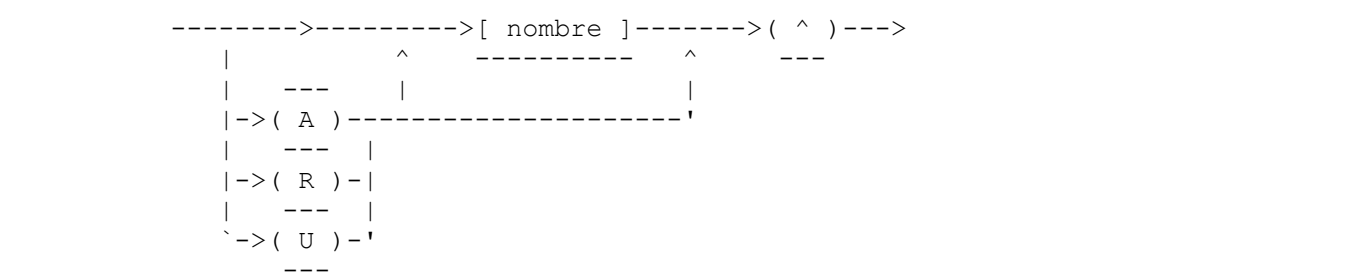

Fig. 7. Indicateurs d'adresses

Un indicateur d'adresses contient une taille d'adresses et un type d'adresses (fig. 7). La taille d'adresses indique le nombre de bits fourni par une sous-adresse. La taille d'adresses est aussi indiquée en bits. Un accent circonflexe est utilisé comme séparateur. Un indicateur d'adresses est placé devant une sous-adresse (fig. 8).

```
JUMP    32^adresse_de_saut
CALL    A16^adresse_de_saut
JUMP,NE R8^adresse_de_saut
MOVE.32 32^{A0}+A16^{D0}+A8^déplacement,D1
MOVE.16 32^{A5}+R8^étiquette,D4
```

Fig. 8. Exemples avec des indicateurs d'adresses

Le type d'adresses indique comment l'adresse est interprétée. Il distingue l'adressage relatif ou absolu. Si en plus, une adresse ne fournit pas la taille d'adresses entière, il faut indiquer le type d'extension à la taille d'adresses entière. Pour l'adressage absolu, une extension peut se faire avec ou sans signe. Pour l'adressage relatif, le compteur d'adresse est ajouté au déplacement étendu de signe.

Les indicateurs d'adresses existent aussi chez les langages d'assemblage des fabricant. Cependant, ils sont difficilement reconnaissables à cause de leur syntaxe non uniforme. Les différentes tailles d'adresses sont souvent indiquées par des préfixes ou des suffixes (p.ex.: LBRA, BRA.S, dépl(A0,D3.L), BNE ETIQUETTE:W). Les types d'adresses sont indiqués par des codes opératoires ou des caractères spéciaux (p.ex.: BSR, ETIQUETTE(PC), @ETIQUETTE). On peut d'ailleurs constater avec satisfaction, que seulement les instructions de saut différencient les modes d'adressage relatif et absolu par deux codes opératoires différents (BRANCH et JUMP). Mais cette distribution du sens d'une instruction sur différentes parties d'une instruction complique et empêche en fin de compte des extensions.

3 Notation des modes d'adressage

Selon le texte précédent, il est primordial de formuler des règles claires pour indiquer explicitement un mode d'adressage. Pour cela, le programmeur a à l'esprit un modèle simplifié du processeur qui contient principalement la structure des registres et l'architecture générale des locations mémoire.

3.1 Registres et locations mémoire

Les registres et les locations mémoire sont constitués d'un certain nombre de bits. Les multiples de 8 bits (= 1 octet) sont usuels chez les microprocesseurs. Ainsi, on obtient des tailles de registre de 8, 16 et 32 bits. Les locations mémoire successives sont numérotées et on appelle leur numéro adresse. Si un mot de 16 bits est placé dans deux locations mémoire successives, deux ordres d'octets différents existent selon le microprocesseur (fig. 9).

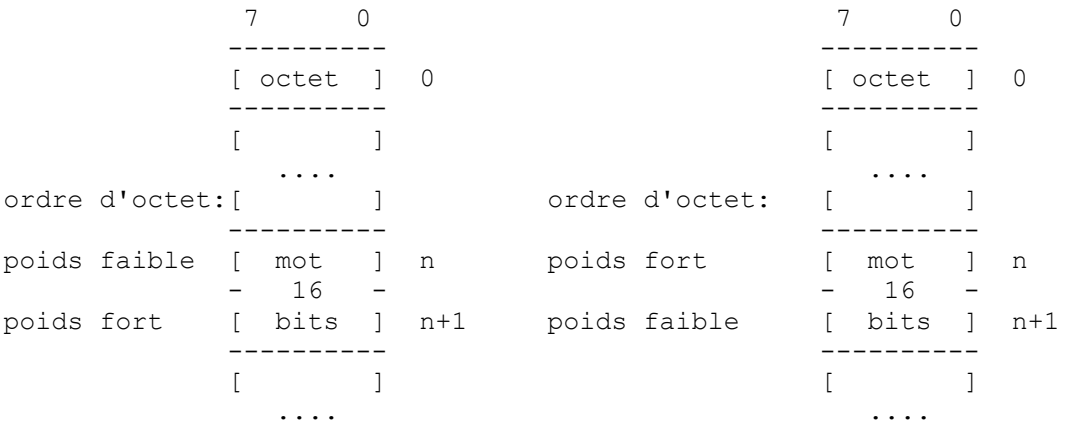

Fig. 9. Différents ordres d'octet

Si l'on adresse une suite d'octet par bit, l'ordre d'octet avec l'octet du poids fort en premier s'avère défavorable car les bits sont numérotés différemment. En plus, la numérotation des bits dans un registre est différente de celle dans une suite de locations mémoire.

3.2 Espaces d'adressage

On peut voir les registres et les locations mémoire faisant partie d'un espace d'adressage de registres ou d'espace mémoire. L'espace d'adressage de registres contient souvent un petit nombre de locations mémoire, mais est directement intégré sur la puce du microprocesseur. Ainsi, le temps d'accès est court. Mais on ne parle pas de locations mémoire mais plutôt de registres, parce que leur nombre est faible et ils bénéficient d'une situation technique particulière. En plus, chaque registre reçoit un nom réservé. Si possible, les noms de registre du fabricant sont repris. Sinon, seulement les symboles cités à la fig. 10 sont réservés en CALM.

```
PC    compt. d'adr. au moment de l'exéc. du prog. (Program Counter)
SP    pointeur de pile (Stack Pointer)
F     registre d'indicateurs arithmétique (Flags)
S     registre d'état (System, Status)
APC   valeur du compteur d'adresse de l'assembleur au début de
      l'instruction dans laquelle se trouve APC
TRUE  -1   (tous les bits à 1)
FALSE 0    (tous les bits à 0)
```

Fig. 10. Symboles réservés et prédéfinis

L'espace mémoire par contre est très grand et est construit à l'extérieur. Les locations mémoire sont numérotées et peuvent obtenir des symbols définis par l'utilisateur.

Quelques microprocesseurs peuvent adresser d'autres espaces d'adressage: entrée/sortie et données. Afin de pouvoir dinstinguer ces espaces d'adressage de l'espace mémoire, on place un signe dollar pour l'espace d'adressage d'E/S et un pourcent pour l'espace d'adressage de données devant l'expression d'adresse (fig. 11).

```
MOVE B,A           registre -> registre
MOVE ADMEMOIRE,A   espace mémoire -> registre
MOVE $ADENTREE,A   espace d'adressage d'E/S -> registre
MOVE %ADDONNEES,A espace d'adressage de données -> registre
```

Fig. 11. Exemples: espaces d'adressage 3.3 Adressage direct

Comme nous avons vu auparavant, les registres et les locations mémoire sont constitués d'un certain nombre de bits. Les états de ces bits représente un mot binaire qu'on appelle contenu de ce registre ou de cette location mémoire. Cependant, dans les langages d'assemblage on indique souvent uniquement le nom d'un contenu, alors qu'on se réfère au contenu de celui-ci. On dit: "incrémente D" au lieu: "incrémente le contenu du registre D".

Cette référence implicite est toujours valable en CALM et pour tous les quatre espaces d'adressage mentionnés. On indique directement le nom réservé du registre ou de l'adresse. Ceci est appelé adressage direct (fig. 12).

```
ADD  B,A           contenu(B) + contenu(A) -> contenu(A)
ADD  ADMEMOIRE,A   contenu(ADMEMOIRE) + contenu(A) -> contenu(A)
ADD  16'1234,A     contenu(16'1234) + contenu(A) -> contenu(A)
```

Le second exemple est identique au troisième si ADMEMOIRE = 16'1234.

Fig. 12. Exemples: adressage direct 3.4 Adressage immédiat

Le signe dièze devant une constante, un symbole, ou une expression complexe, supprime la référence implicite et indique l'adresse construite immédiatement (et pas son contenu). Ceci est appelé addressage immédiat (fig. 13). Le dièze correspond au signe de pointeur (^, @) dans les langages de haut niveau.

```
ADD  #16'1234,A      16'1234 + contenu(A) -> contenu(A)
ADD  #ADMEMOIRE+1,A  ADMEMOIRE + 1 + contenu(A) -> contenu(A)
JUMP ADMEMOIRE       ADMEMOIRE -> contenu(PC)
MOVE #ADMEMOIRE,PC   ADMEMOIRE -> contenu(PC)
```

Fig. 13. Exemples: adressage immédiat

3.5 Adressage indirect

Si l'on veut utiliser le contenu d'une adresse quelconque de nouveau comme adresse, il faut placer des accolades autour de l'expression d'adresse en question. Ceci est appelé adressage indirect (fig. 14).

```
ADD  {HL},A          contenu(contenu(HL)) + contenu(A) -> contenu(A)
MOVE #16'1234,{HL}   16'1234 -> contenu(contenu(HL))
MOVE #16'1234,{ADMEMOIRE}  16'1234 -> contenu(contenu(ADMEMOIRE))
```

Fig. 14. Exemples: adressage indirect

En fait, il n'y a pas de différence fonctionnelle si l'on utilise un registre ou une location mémoire pour l'adressage indirect. Mais beaucoup de processeurs ne permettent qu'avec de registres un adressage indirect.

3.6 Adressages combinés

Avec l'adressage direct, immédiat et indirect, nous avons déjà défini les éléments de base pour construire des adressages complexes quelconques. Les opérations possibles entre les différents termes d'adresse sont l'addition, la soustraction et la multiplication (fig. 15).

```
MOVE #16'FE,{A0}+8     254 -> contenu(contenu(A0)+8)
MOVE #10'98,{A0}-8     98 -> contenu(contenu(A0)-8)
MOVE #8'177,{A0}*8     127 -> contenu(contenu(A0)*8)
MOVE #2'1011,{A0}*     11 -> contenu(contenu(A0)*taille_de_données)
MOVE R0,{{SB}+5}+3     cont.(R0) -> cont.(contenu(contenu(SB)+5)+3)
MOVE {A0}+{D0}+10,A1   cont.(cont.(A0)+cont.(D0)+10) -> cont.(A1)
MOVE #{A0}+{D0}+10,A1  contenu(A0)+contenu(D0)+10 -> contenu(A1)
```

Fig. 15. Exemples: adressages combinés

Notez que l'adressage indirect est simplement une notation explicite de l'adressage direct, mais peut être appliquer sans limites. La référence implicite (c.à.d. le contenu de l'adresse calculée) est appliquée à l'expression d'adresse entière. Si un signe dièze se trouve devant l'expression d'adresse entière, cette référence implicite est annulée.

3.7 Adressages spéciaux

L'adressage relatif est en fait une notation plus courte d'un adressage combiné comme c'est illustré à la fig. 16.

Quelques modes d'adressage supplémentaires ont pris une certaine importance et ont obtenu pour cela leur propre notation.

Dans un adressage de bit, la première expression spécifie l'adresse d'octet. L'expression après les deux-points est l'adresse de bit. L'adresse de bit zéro correspond au bit 0 dans l'octet adressé. Si l'adresse d'octet pointe dans l'espace mémoire, l'adresse de bit peut dépasser l'octet adressé (après ou avant l'octet).

notation	description	désignation
MOVE R^ADRESSE,D0	MOVE {PC}+DEPLACEMENT,D0	adressage relatif
MOVE {A0+},D0	MOVE {A0},D0	modif. automatique
	ADD #taille_de_données,A0	(incrémentation)
MOVE D0,{-A0}	SUB #taille_de_données,A0	modif. automatique
	MOVE D0,{A0}	(décrémentation)
CLR D0:#1	met bit 1 à zéro dans D0	adressage de bit
PUSH R1..R3|R5	PUSH R1	liste de registre
	PUSH R2	
	PUSH R3	
	PUSH R5	
MOVE R0,R1:#1..#4	transfère bits 0 à 3 de R0 à R1 (bits 1 à 4)	liste de bit

Fig. 16. Exemples: adressages spéciaux

4 Pseudo-instructions

Les pseudo-instructions sont des commandes à l'assembleur. Elles dirigent la génération de code, l'assemblage et le listage conditionnel, les macros, etc.. Chaque pseudo-instruction commence avec un point (fig. 17).

Une étiquette peut être placée uniquement devant les pseudo-instructions .ASCII, .ASCIZ, .BLK, .FILL, .n et .STRING. L'ordre des octets dans les pseudo-instructions .16, .32, etc., dépend du processeur chargé (.PROC).

```
   Listage
   .TITLE chaîne de caractères  commence une nouv. page avec le titre.
   .CHAP chaîne de caractères  ins. le texte ind. dans le sous-titre.
   .END                    termine un programme source.
   .TEXT                   ignore jusqu'à .ENDTEXT les lignes suivant.
                           qui sont copiées dans le fichier listage.
   .ENDTEXT                termine .TEXT.
   .LIST expression        permet le list. des lignes suiv. si l'exp.
                           est vraie. On peut imbriq. les .LIST.
   .ENDLIST                termine une section de listage condition.
   .LAYOUT                 définit les param. pour l'asp.gén. du list.
   Assemblage
   .BASE expression        déf. la nouv. base par défaut pour les nb.
   .START expression       définit l'adresse de début.
   .EXPORT symbole1, symbole2, ...définit les symboles exportés.
   .IMPORT symbole1, symbole2, ...définit les symboles importés.
   Compteur d'adresse
   .APC expression         choisit un des compt. d'adr. de l'ass.(APC)
   .LOC expression         ass. une nouv.val. au compt. de l'ass.(APC)
   .ALIGN expression       aj. la val. de l'APC à la proch. adr. mult.
   .EVEN                   ajuste la val. de l'APC à une val. paire.
   .ODD                    ajuste la val. de l'APC à une val. impaire.
   .BLK.8 expression (nombre) additionne à l'APC le produit de la
   .BLK.8.16.32 expression  taille des données par le nombre.
   Insertion de fichiers
   .INS fichier            insère le fichier de source indiqué.
   .REF fichier            insère la table des symboles mentionnée.
   .PROC fichier           insère la description du proc. indiqué.
   Génération de code
   .n   expression, expression, ...insère la valeur donnée dans le
   .8.32 expression8, expression32, expression8, ... programme objet.
   .FILL.n expression (nombre), expression  gén. nb. fois la val.ind.
   .ASCII "texte ascii"    insère les codes ASCII des caractères
                           imprimables dans l'objet.
   .ASCIZ "texte ascii"    = .ASCII, aj. le code ASCII nul à la fin.
   .STRING "texte ascii"   = .ASCII, insère la long.(8 bits) au début.
   Assemblage conditionnel et macros
   .IF  expression         les lignes d'instr. suiv. jusqu'au .ELSE
                           ou .ENDIF corres. sont ass. si l'exp. est
                           vraie (TRUE). Peuvent être imbriquées.
   .ELSE                   les lignes d'instr. suiv. jusqu'au .ENDIF
                           sont ass. si l'exp. du .IF était faux.
   .ENDIF                  termine une section d'un .IF ou .ELSE.
   .MACRO nom,paramètre1,...commence la définition d'une macro avec le
                           nom de la macro et une liste fac. de param.
   .ENDMACRO               termine la définition d'une macro.
   .LOCALMACRO symbole1,... déclare les symb. locaux dans une macro.
```

Fig. 17. Pseudo-instructions CALM

5 Utilisation de CALM - pas si facile

Chaque nouveau langage de programmation a besoin de temps pour être accepté. Mais contrairement aux langages de programmation de haut niveau, un langage d'assemblage commun dépend du processeur.

5.1 Documentation

La notation CALM des instructions d'un processeur est actuellement documentée dans des dites cartes de référence. Dans ces cartes de référence se trouvent toutes les instructions d'un processeur sous une forme condensée.

Mais des informations supplémentaires comme codes machines, temps d'exécution, particularités, etc., d'un microprocesseur se trouvent uniquement dans la documentation du fabricant. Ainsi, l'utilisateur est obligé de connaître les deux notations. Comme aide, les désignations des codes opératoires du fabricant sont aussi indiquées dans les cartes de référence CALM.

5.2 Une notation CALM - plusieurs codes machines

Les microprocesseurs modernes 16/32 bits ont plusieurs instructions qui sont 100% identiques. La notation CALM montre ceci d'une manière exemplaire. Alors la question se pose: Quel code machine doit l'assembleur générer?

Normalement, on laisse l'assembleur chosir le code le plus compact et ainsi le plus rapide. Sinon, les caractéristiques sont déterminées par les opérandes. Le code machine résultant dépend de la taille et du type de l'opérande, ainsi que des indicateurs d'adresses et de données additionnelles.

5.3 Assembleur

Il est évident qu'une programmation en CALM ne sert à rien si les outils correspondants (c.à.d. assembleur, éditeur de liens) manquent. Actuellement, des assembleur CALM générant un code machine non translatable sont disponible auprès de l'auteur pour pratiquement tous les microprocesseurs.

5.4 Format d'objet

La génération du code machine reste extrêmement complexe, car pour chaque processeur n'existe pas seulement un propre format de code machine, mais aussi un format d'objet spécifique. Cette diversité empêche pratiquement une généralisation de l'assembleur. Toujours est-il qu'un format d'objet uniforme a été défini [2].

6 Exemples

Dans les exemples suivants, quelques instructions en notation CALM sont comparées à ceux du fabricant. Ce ne sont pas des programmes complèts, mais plutôt un choix de vingt instructions des processeurs iAPX86, M68000 et NS32000.

La notation d'assemblage dépend fortement des subtilités de l'assembleur utilisé. Nous avons utilisés ici les assembleurs des fabricants.

6.1 iAPX86

L'iAPX86 d'Intel est à cause de sa segmentation un des microprocesseurs les plus complexes. L'assembleur ASM-86 nécessite de nombreuses indications concernant ces segments (ASSUME, NEAR, déclaration des variables, etc.). Pour cette raison, les exemples suivants sont incompléts, car ces indications manquent ici.

A cause de cette architecture, un langage d'assemblage ne peut pas être simple pour ce processeur (fig. 18). CALM indique avec chaque instruction le registre de segment utilisé. De plus, on fixe avec un indicateur de données la portée des instructions de saut (à l'intérieur ou à l'extérieur du segment courant). Une notation simplifiée est introduite pour le décalage automatique des registres de segments ([CS] pour {CS}*16).

CALM	Intel
MOVE.16 #16'1000,BX	MOV BX,1000H
MOVE.16 AX,${DX}	OUT DX,AX
MOVE.16 #[ES]+NEXT,DX	LEA DX,ES:NEXT
MOVE.8 DL,BH	MOV BH,DL
MOVE.8 [DS]+DATA,AL	MOV AL,DATA
MOVE.16 [CS]+{SI},AX	MOV AX,CS:[SI]
MOVE.8 AH,F	SAHF
PUSH.16 SF	PUSHF
MOVE.8 [DS]+{BX}+{AL},AL	XLAT CONV_TAB
MOVE.32 [DS]+TARGET,DSBX	LDS BX,TARGET
CONV.A8.16 AX	CBW
INC.8 AL	INC AL
DIV.A16 CX,DXAX,AX,DX	IDIV CX
OR.16 #ERROR,[DS]+STATUS	OR STATUS,ERROR
RL.8 AL	ROL AL,1
TEST.16 #MASK,AX	TEST AX,MASK
JUMP,LO R^LOW	JB LOW
CALL.32 20^ROUTINE	JSR FAR ROUTINE
DJ.16,NE CX,ADDRESS	LOOP ADDRESS
TRAP.32,VS	INTO

Fig. 18. Exemples: iAPX86

6.2 M68000

La notation des instructions du M68000 de Motorola ne diffère que de très peu de CALM (fig. 19). Mais on voit particulièrement bien avec ce processeur que la compatibilité ascendante est bien valable pour le code machine (malheuresement seulement partiellement), mais pas pour le langage d'assemblage: Les possibilités d'adressage supplémentaires du M68020 ont obligés Motorola de changer la notation de quelques modes d'adressage.

CALM	Motorola
MOVE.32 #16'1000,D1	MOVE.L #$1000,D1
MOVE.32 #A8^WERT,D0	MOVEQ #WERT,D0
MOVE.32 #{A6}+100,A3	LEA 100(A6),A3
MOVE.8 D4,D6	MOVE.B D4,D6
MOVE.8 {A4}+4,D2	MOVE.B 4(A4),D2
MOVE.16 {A0}+A16^{D4}+2,{A2}+32^{A3}+4	MOVE 2(A0,D4),4(A2,A3.L)
MOVE.16 D0,F	MOVE D0,CCR
PUSH.16 SF	MOVE SR,-(A7)
PUSH.32 #TABLE	PEA TABLE
PUSH.32 D0..D3|D5	MOVEM.L D0-D3/D5,-(A7)
CONV.A8.16 D4	EXT.W D4
INC.8 D1	ADDQ.B #1,D1
DIV.A16 #10,D6	DIVS #10,D6
OR.16 #MASK,{A4}+A16^{D4}-4	OR #MASK,-4(A4,D4.W)
RL.32 #8,D4	ROL.L #8,D4
TCLR.8 {A0+}:D4	BCLR D4,(A0)+
JUMP,LO R^LOW	BLO LOW
CALL U^ROUTINE	JSR ROUTINE
DJ.16,NMO D0,ADDRESS	DBRA D0,ADDRESS
TRAP,VS	TRAPV

Fig. 19. Exemples: M68000

6.3 NS32000

En réalité il s'agit ici d'une famille complète, appelée par National Semiconductors "NS32000 series". Elle consiste de trois membres (NS32032, NS32016 et NS32008) 100% compatibles qui se différencient que par la largeur du bus externe.

Il faut surtout mentionner la symétrie parfaite des modes d'adressages (fig. 20). L'adressage relatif est permis partout et est codé automatiquement.

CALM		NS	
MOVE.32	#16'1000,R1	MOVD	H'1000,R1
MOVE.32	#A4^WERT,R0	MOVQD	WERT,R0
MOVE.32	#{R6}+100,{SB}-20	ADDR	100(R6),-20(SB)
MOVE.8	R4,R6	MOVB	R4,R6
MOVE.8	{SB}+4,R2	MOVB	4(SB),R2
MOVE.16	{{FP}+2}+4,{SB}+{R0}*1+2	MOVW	4(2(FP)),2(SB)[R0:B]
MOVE.8	R0,F	LPRB	R0,UPSR
PUSH.16	SF	SPRW	PSR,TOS
PUSH.32	#R^TABLE	ADDR	TABLE,TOS
PUSH.32	R0..R3|R5	SAVE	[R0,R1,R2,R3,R5]
CONV	R4.8,R2.32	MOVZBD	R4,R2
INC.8	R1	ADDQB	1,R1
DIV.A32	#10,R6	QUOD	10,R6
OR.16	#MASK,{{SB}-2}+{R5}*8+4	ORW	MASK,4(-2(SB))[R5:Q]
RL.32	#-10,R4	ROTD	-10,R4
TCLR	{R0}+20:{{SB}+10}+4.A32	CBITD	4(10(SB)),20(R0)
JUMP,LO	R^LOW	BLO	LOW
CALL	U^ROUTINE	JSR	@ROUTINE
AJ.32,NE	#2,{R0}+4,ADDRESS	ACBD	2,4(R0),ADDRESS
TRAP,VS		FLAG	

Fig. 20. Exemples: NS32000

7 Conclusions

CALM est un langage d'assemblage commun qui se prête à tous les microprocesseurs, mais aussi pour les miniprocesseurs et les calculateurs. La notation CALM est orientée vers le futur puisqu'on peut l'étendre d'une façon ponctuelle et fonctionelle.

La séparation nette des différentes parties d'une instruction permet une adaptation individuelle à des processeurs les plus diverses. Des extensions spécifiques au processeur sont toujours possibles. En plus, on comprend une instruction dans la notation CALM souvent mieux qu'après de longues pages d'explications.

On a testé la notation CALM avec 24 processeurs pour voir si l'on peut exprimer tous les processeurs d'une manière satisfaisante. Comme résultat, des cartes de référence CALM ont été établi pour les processeurs suivants: Z80, 65x02, 680x, 6805, 6809, 8048, 8051, 808x, iAPXx86, NS32000, 680xx, etc.

Nous avons essayé de montrer sur quelques pages les éléments principaux de CALM. La programmation en assembleur en soi ne deviendra pas plus facile dans la notation CALM, mais elle sera plus compréhensible pour l'utilisateur, car elle est construite d'après de règles logiques.

8 Bibliographie

[1] DIN 66283, Allgem. Assemblierer-Sprache für Mikroproz. CALM Norme nat. all., Beuth Verlag GmbH, CP 1145, D-1000 Berlin 30

[2] "The Microprocessor Universal Format for Object Modules", Prop. Stand.: IEEE P695 Working Group. IEEE Micro, 8.1983, p. 48-66.

[3] Nicoud, J.D. and Fäh, P. Common Assembly Language for Microprocessors CALM, Document interne. LAMI-EPFL, INF-Ecublens, CH-1015 Lausanne, Dec. 1986. English version of DIN 66283.

[4] Nicoud, J.D. and Fäh, P. Explanations and Comments, Related to the Common Assembly Language for Microprocessors. LAMI-EPFL.

[5] Zeltwanger, H. "Genormte Assemblersprache für Ps", ELEKTRONIK, 35ème année (1986), no. 8, p. 66-71.

[6] Nicoud, J.D. Calculatrices, Volume XIV du Traité d'Electricité, Lausanne: Presses polytechniques romandes, 1983.

[7] Nicoud, J.D. and Wagner, F. Major Microprocessors, A unified approach using CALM. North Holland, 1987.

[8] Strohmeier, A. Le matériel informatique, concepts et principes Lausanne: Presses polytechniques romandes, 1986.

[9] Fäh, P. "Die (Un-)Logik von Assemblersprachen", Elektroniker, 26ème année (1987), no. 5, p. 97-100.