Travaux dirigés n° 4 : Initiation au Langage Machine
Cette planche de TD concerne le langage machine simplifié LM0 vu en cours au chapitre II " Initiation au Langage Machine ". Ci-dessous sont rappelés les éléments essentiels des syntaxe et fonctionnement du langage LM0, puis suivent les exercices proprement dits.

Rappels sur le langage LM0

Le microprocesseur MP0 considéré en cours est une version simplifiée et idéalisée des vrais processeurs. Il possède plusieurs registres parmi lesquels cinq nous intéresse ici :

1. Deux registres de données D_0 et D_1 qui stockent un octet chacun ;
2. Deux regisres d'adresses A_0 et A_1 qui stockent un octet (représentant une adresse en mémoire) ;
3. Un registre d'état contenant les bits Z et N.
   – Quand $Z = 1$, cela signifie que la dernière instruction a produit un résultat égal à zéro ;
   – Quand $N = 1$, cela signifie que la dernière instruction a produit un résultat négatif.

Il existe trois modes d'adressages, c'est-à-dire trois façons différentes d'accéder à des valeurs stockées en mémoire.

1. Adressage immédiat : si on connaît explicitement la valeur n, on écrit " $\#n$ " pour représenter cette valeur ;
2. Adresssage direct : on ne connaît pas explicitement la valeur n, mais on connaît l'adresse m où elle est stockée, alors la notation " m " (sans le dièse) signifie que l'on considère m comme une adresse ;
3. Adressage indirect : ni la valeur ni l'adresse de la donnée ne sont connues, en revanche cette dernière est stockée dans un des registres d'adresses A_i ($i = 0, 1$). Alors la notation " (A_i) " désigne la valeur stockée en mémoire à l'adresse contenue dans A_i. Notons que (D_0) ou (D_1) n'a aucun sens puisque D_0 et D_1 contiennent des données et non des adresses.

Les instructions du langage LM0 ont (presque toutes) la forme suivante " `nom source, destination` " avec comme restriction qu'au moins un(e) des deux arguments (opérandes), `source` ou `destination`, soit un registre de données ou d'adresses (note : il est également possible d'avoir " (A_i) ", $i = 0, 1$). Notez bien que ces instructions modifient les valeurs des bits d'état Z et N.

1. Instruction de transfert : " `MOVE source, destination` " signifiant que l'on affecte la valeur désignée par `source` à `destination`. Par exemple,
   – `MOVE #10, A1` : on place la valeur 10 dans A_1 (en particulier, 10 doit ici représenter une adresse) ;
   – `MOVE D0, 123` : on place la valeur contenue dans D_0 dans l'emplacement mémoire d'adresse 123 ;
   – `MOVE (A0), D1` : on place la valeur contenue dans l'emplacement mémoire située à l'adresse stockée dans A_0 dans le registre D_1 ;

– `MOVE #10, 110` : cette écriture n'est pas valide. Pour placer la valeur 10 à l'emplacement d'adresse 110, il faut exécuter une séquence de deux instructions :
`MOVE #10, D0`
`MOVE D0, 110`
Les bits d'états sont modifiés par `MOVE` :

$Z = 1$ si la valeur transférée est 0 et $N = 1$ si la valeur transférée est négative.

2. Instructions arithmétiques et logiques : " `OP source, destination` " permet d'effectuer l'opération " `destination OP source` ", puis d'affecter le résultat obtenu à `destination`. Les différentes opérations possibles pour `OP` sont
   – `ADD` : addition ;
   – `SUB` : soustraction ;
   – `MUL` : multiplication ;
   – `DIV` : division entière ;
   – `AND` : conjonction logique ;
   – `OR` : disjonction logique ;
   – `NEG` : négation logique (cette opération n'a qu'une seule opérande jouant à la fois les rôles de la source et de la destination).
   Par exemple " `SUB #3, D0` " : si D_0 contient initialement 2, alors, après cette instruction, il contient $-1(= 2 - 3)$, et ici, le bit Z prend la valeur 0 et le bit N la valeur 1.
3. Instruction de comparaison : `CMP source, destination` qui effectue le même traitement que `SUB` sans mise à jour du contenu de `destination` (par contre les bits Z et N sont modifiés comme pour `SUB` ;
4. Instructions de saut :
   (a) Saut inconditionnel : `JMP destination` permet de passer directement à l'instruction `destination` dans un programme ;
   (b) Sauts conditionnels :
      i. `JEQ destination` : on passe à l'instruction `destination` à la condition que $Z = 1$ (c'est-à-dire que le résultat de la dernière opération effectuée est nul) ;
      ii. `JNE destination` : on passe à l'instruction `destination` à condition que $Z = 0$ (le résultat de la dernière opération effectuée n'est pas nul) ;
      iii. `JLT destination` : on passe à l'instruction `destination` à condition que $N = 0$ (le résultat de la dernière opération effectuée est négatif) ;
      iv. `JLE destination` : on passe à l'instruction `destination` à condition que $N = 0$ ou $Z = 1$;
      v. `JGT destination` : on passe à l'instruction `destination` à condition que $N = 1$;
      vi. `JGE destination` : on passe à l'instruction `destination` à condition que $N = 1$ ou $Z = 1$.

Exemples :

1. Supposons que la valeur d'une variable x soit stockée dans D_0. L'algorithme suivant
```
Si x>3,
      alors affecter x-5 à x ;
      sinon affecter x+10 à x ;
```

```
fin si ;
suite du programme...
```
se traduit en LM0 comme suit :
```
100 : CMP #3, D0
102 : JGE #108
104 : SUB #5, D0
106 : JMP #110
108 : ADD #10, D0
110 : suite du programme...
```

2. Supposons encore que la valeur de la variable x soit stockée dans D_0. Considérons l'algorithme
```
Tant que x > 10 Faire
      Affecter x-1 à x ;
Fin Tant que ;
suite du programme...
```
Ce dernier se retranscrit en LM0 ainsi qu'il suit :
```
100 : CMP #10, D0
102 : JGE #108
104 : SUB #1, D0
106 : JMP #100
108 : suite du programme...
```

Exercices

Une chaîne de caractères est codée en C grâce à un tableau de caractères dont le dernier caractère de la chaîne est suivi par un 0. Par exemple : si "abc" est stocké dans un tableau déclaré par char t[10] ; alors t[0] contient 'a' (97), t[1] contient 'b' (98), t[2] contient 'c' (99) et t[3] contient 0. (Entre parenthèses sont données les valeurs dans la table ASCII des caractères.) Nous allons écrire les instructions en LM0 qui permettent de :

1. récupérer la longueur d'une chaîne.
2. concaténer deux chaînes.
3. vérifier si deux chaînes sont identiques.

Exercice n^o 1 : Calcul de la longueur d'une chaîne

Écrire un (bout de) programme en LM0 qui calcule la longueur d'une chaîne de caractères (la longueur finalement calculée sera contenue dans le registre D0 et on suppose que le début de la chaîne se situe à l'adresse 200).

Solution 1

100 : MOVE #0, D0 (D0 est le compteur de caractères de la chaîne)
102 : MOVE #200, A0 (on place l'adresse 200 dans le registre d'adresses)
104 : CMP #0, (A0) (on compare le caractère situé à l'adresse contenue dans A0 avec le caractère de fin de chaînes 0)
106 : JEQ #114
108 : ADD #1, D0

110 : ADD #1, A0 (on passe à la case mémoire immédiatement voisine de celle située à l'adresse contenue dans A0, à savoir, A0+1)
112 : JMP #104
114 : ... (D0 contient la longueur de la chaîne)

Exercice n^o 2 : Concaténation de deux chaînes

Écrire un (bout de) programme effectuant la concaténation de deux chaînes en en langage machine LM0. (On suppose que la première chaîne débute à l'adresse 150, la seconde à l'adresse 175 et le résultat de la concaténation débute à l'adresse 200.)

Solution 2

100 : MOVE #150, A0 (la 1ère chaîne débute à l'adresse 150)
102 : MOVE #200, A1 (la chaîne résultat débute à l'adresse 200)
104 : CMP #0, (A0) (le caractère est-il nul (fin de chaîne) ?)
106 : JEQ #116
108 : MOVE (A0), (A1) (on place le caractère courant de la première chaîne dans le résultat)
110 : ADD #1, A0 (on passe à la case mémoire suivante pour la première chaîne)
112 : ADD #1, A1 (idem pour la chaîne résultat)
114 : JMP #104
116 : MOVE #175, A0 (la seconde chaîne débute à l'adresse 175)
118 : CMP #0, (A0) (le caractère est-il nul (fin de chaîne) ?)
120 : JEQ #130
122 : MOVE (A0), (A1) (on place le caractère courant de la second chaîne dans la chaîne résultat)
124 : ADD #1, A0 (on passe à la case mémoire suivante pour la première chaîne)
126 : ADD #1, A1 (idem pour la chaîne résultat)
128 : JMP #118
130 : MOVE #0, (A1) (on place le caractère de fin de chaîne dans la chaîne résultat)

Exercice n^o 3 : Test d'égalité entre deux chaînes

Écrire un (bout de) programme effectuant le test d'égalité entre deux chaînes en langage machine LM0. (On suppose que la première chaîne débute à l'adresse 150, la seconde à l'adresse 200. D0 contiendra 1 si les deux chaînes sont égales et 0 dans le cas contraire.)

Solution 3

100 : MOVE #150, A0 (la 1ère chaîne débute à l'adresse 150)
102 : MOVE #200, A1 (la 2ème chaîne débute à l'adresse 200)
104 : CMP #0, (A0) (est-on au bout de la 1ère chaîne ?)
106 : JEQ #122 (oui : il faut tester si on est également au bout de la seconde)
108 : CMP #0, (A1) (est-on au bout de la seconde chaîne ?)
110 : JEQ #130 (oui : les deux chaînes diffèrent)
112 : CMP (A0), (A1) (les caractères courants sont-ils égaux ?)
114 : JNE #130 (non : les deux chaînes diffèrent)
116 : ADD #1, A0 (on passe au caractère suivant pour la première chaîne)

118 : ADD #1, A1 (on passe au caractère suivant pour la seconde chaîne)
120 : JMP #104
122 : CMP #0, (A1) (est-on au bout de la seconde chaîne ?)
124 : JEQ #130 (non : les deux chaînes diffèrent)
126 : MOVE #1, D0 (on indique que les deux chaînes sont identiques)
128 : JMP #132
130 : MOVE #0, D0 (on indique que les deux chaînes diffèrent)
132 : ... (D0 contient 1 si les chaînes sont égales et 0 dans le cas contraire)