2 – Programmation élémentaire

Objectifs

L'objectif du TD est de se familiariser avec quelques concepts de base en programmation : la structuration de programmes assembleur, les structures conditionnelles, les boucles.

1.1 Structure et analyse d'un programme assembleur

Un programme assembleur est composé d'instructions, de déclarations et éventuellement de directives pour l'assemblage du programme. Ces dernières permettent de préciser par exemple la présence de variables ou de procédures externes. Pour des raisons également d'assemblage, les instructions et les déclarations ne sont pas mélangées mais définies au sein de sections de code et de données.

Dans le listing fourni ci-dessous, vous pouvez apercevoir ces différentes zones. Vous pouvez également identifier 3 zones de découpage vertical du programme correspondant à la zone étiquette (labels), instructions ou directives, paramètres et commentaires.

```
Labels      Instructions   Paramètres                Commentaires
                                                     @ Section données
RES:        .word          0
N:          .word          5
NUM1:       .word          3, -17, 27, -12, 322
                                                     @ Section code
            LDR            r1, N
            ADR            r2, NUM1
            MOV            r0, #0
LOOP:       LDR            r3, [r2]
            ADD            r0, r0, r3
            ADD            r2, r2, #4
            SUB            r1, r1, #1
            CMP            r1, #0
            BGT            LOOP
            STR            r0, RES
```

Exercices :

a) Analyser ce programme et déterminer son comportement, instruction par instruction en ajoutant des commentaires.

b) Quelle est le contenu de chaque registre à la fin de l'exécution ?

c) Donner un équivalent de ce programme assembleur en langage C.

d) Modifier le programme assembleur précédent pour qu'il effectue la somme des N premiers entiers, sans utiliser de tableau `NUM1`.

1.2 Structures conditionnelles

Un exemple de structure conditionnelle est donné par l'expression *if … else* en langage C.

Exemple : *if (a>b) c=a else c=b ;*

Une traduction possible en assembleur ARM est la suivante:

```
                                        @ Section données
A:          .word       …
B:          .word       …
C:          .word       …
                                        @ Section code
            LDR         r0, A           @ r0 <- A
            LDR         r1, B           @ r1 <- B
            CMP         r0, r1          @ compare A et B
            BLE         ELSE            @ saut à l'instruction
                                        @ suivant ELSE si ≤
            MOV         r2, r0          @ .. C = A..
            B           ENDIF           @ saut à l'instruction suivant
                                        @ ENDIF
ELSE        MOV         r2, r1          @ … else C = B
ENDIF       STR         r2, C           @ C <-r2
```

Dans l'instruction `BLE ELSE`, `B` est une instruction de branchement, elle permet de sauter à l'exécution de l'instruction qui suit l'étiquette `ELSE` (`MOV r2, r1`). Le suffixe `LE` dans l'instruction `BLE` permet d'exécuter ou non ce branchement si le résultat renvoyé par l'instruction `CMP` est `LE` (Less or Equal). De la même façon, le suffixe `NE` permet une exécution conditionnelle, mais cette fois si le résultat de l'instruction `CMP` est `NE` (Not Equal).

Exercices:

a) Donner une traduction possible en assembleur ARM de l'expression :

*if (a==b) c=a else c=b*8 ;*

b) Ecrire un algorithme, puis un programme assembleur qui renvoie dans le registre r3

- le contenu de r0/8 si sa valeur est divisible par 8.
- la valeur 0 sinon.

c) Ecrire un algorithme, puis un programme assembleur qui renvoie le maximum de trois valeurs contenues dans r0, r1, r2 dans le registre r3.

d) Il est possible d'écrire plus efficacement la structure conditionnelle de l'exemple (*if (a>b) c=a else c=b ;*) sans utiliser de branchements, en utilisant la forme conditionnelle des instructions. Donner le code assembleur correspondant et comparer les deux versions.

1.3 Structures itératives

a) On considère un simple programme de calcul de somme des N premiers entiers. Un tel programme peut s'écrire en C de la façon suivante :

total = 0;
for (i=1 ; i<N ; i++)
total = total + i;

Donnez un équivalent du programme en assembleur ARM

b) Traduire la boucle suivante en assembleur ARM :

for (i=0 ; i<10 ; i++) {A[i] = 0}