

4 – Programmation structurée

Objectifs

L'objectif de ce TD est la programmation structurée (manipulation de la pile, appels de sous-programmes, sauvegarde de variables locales, passage de paramètres).

Rappels

Le modèle de pile utilisé dans les exercices est le modèle Full Descending (conformément à l'ARM Procedure Call Standard). La manipulation de la pile se fait par les instructions :

STR Rd, [SP, #-4]!	pour empiler un registre
STMFD SP!, {reglist}	pour empiler une liste de registres
LDR Rd, [SP], #4	pour dépiler un registre
LDMFD SP!, {reglist}	pour dépiler une liste de registres

1.1 Sauvegarde de contexte

a) Etant donné le code suivant :

```
A:          .word      1
B:          .word      2
C:          .word      3
main:
            LDR        R8, A
            LDR        R9, B
            LDR        R4, C
            MOV        R10, #1
            BL         fonction1
            ADD        R0, R0, R8
            ADD        R0, R0, R9
            ADD        R0, R0, R10
wait :      B          wait

fonction1:
            STMFD      SP!, {R8, R9, R10, LR}
            MOV        R10, #3
            LDR        R9, C
            ADD        R8, R10, R9
            BL         fonction2
            MUL        R0, R8, R0
            LDMFD      SP!, {R8, R9, R10, LR}
            MOV        PC, LR

fonction2:
            STMFD      SP!, {R8, LR}
            MUL        R8, R4, R4
            ADD        R0, R8, R8
            LDMFD      SP!, {R8, LR}
            MOV        PC, LR
```

Je Quel devrait être le résultat correct de ce programme?

Que se passerait-il si on omettait les instructions LDMFD et STMFD dans fonction1?

Que se passerait-il si on omettait les instructions LDMFD et STMFD dans fonction2?

On suppose maintenant que les instructions LDMFD et STMFD sont présentes dans fonction1 et fonction2, représenter l'état de la pile juste avant l'exécution de l'instruction ADD R0, R8, R8 ?

b) Analyser le code suivant :

```
foo:
    STMFD    SP!, {R8, LR}
    MOV      R8, #4
    ADD      R1, R8, R4
    BL       bar

bar:
    MOV      R8, R4
    ADD      R8, R8, R1
    MOV      R1, R8
    LDMFD    SP!, {R8, LR}
    MOV      PC, LR
```

Expliquer quels sont les problèmes et rajouter les instructions manquantes pour que les deux sous programmes fonctionnent.

1.2 Passage de paramètres

Soit le programme C suivant :

```
main() {
    int a;
    a = max (56, 49);
}

int max (int x, int y) {
    if (x > y) return x;
    else return y;
}
```

Ecrire le programme appelant main et la fonction max en assembleur ARM :

- en passant les paramètres par registres, sans sauvegarde de contexte.
- en passant les paramètres par la pile, sans sauvegarde de contexte.
- en passant les paramètres par la pile, avec sauvegarde du contexte.
- Analyser le code assembleur produit par le compilateur à partir du code C précédent.

1.3 Somme des carrés

On souhaite écrire un programme en assembleur ARM qui prend en entrée un tableau X de dimension N et réalise le calcul : $X[0]^2 + X[1]^2 + \dots + X[n-1]^2$

Ecrire, en respectant les contraintes qui vous sont imposées :

- a) Un sous programme CARRE qui reçoit en paramètre *par la pile* un nombre X et renvoie dans le registre **R0** la valeur X^2 .
- b) un sous-programme SOMME_CARRE qui reçoit en paramètre, dans le registre **R0**, l'adresse d'un tableau et renvoie, dans le registre **R1**, la somme des carrés des éléments de ce tableau. Cette somme sera calculée en utilisant le sous-programme CARRE.
- c) le programme principal qui appelle le sous-programme SOMME_CARRE pour calculer la somme des carrés d'un tableau (de 10 éléments qu'il faudra initialiser) et dont l'adresse est dans le registre **R0**. Au retour du sous-programme, on veut que **R0** contienne toujours la valeur de départ. Le résultat sera récupéré dans **R1**.