Corrigé TD n° 4 : Fonctions et procédures

On utilise dans ce TD les jeux d'instructions du NIOSII et de l'ARM.

1. Procédures simples

Soit le programme C suivant

```
int a, b, c ;
main(){
B=10; C=15;
a= max (b,c) ;
}

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

a) Ecrire le programme appelant et la fonction en assembleur MIPS32. (Les variables A, B, C sont rangés dans des cases mémoire successives).

/* les registres r4 et r5 sont utilisés en entrées et r2 pour récupérer le résultat.

```
Main :                          // NB : main est une fonction.
     addi r29,r29,-4            // push adresse de retour
     sw r31,0(r29)
     la r8,A          // adresse de A dans R8
     lw r4, 4(r8)           // valeur de B dans r4
     lw r5, 8(r8)           // valeur de C dans r5
     jal fmax
     sw r2, 0(r8)    //valeur du Max rangée dans la variable A
     lw r31,0(r29)              //pop adresse de retour
     addi r29,r29,4
     jr r31



fmax :    add r2,r4,r0
          slt r9,r5,r4
          bgtz r9,suite  //branchement si r5 < r4
          add r2,r5,r0   // r5 est le max
suite :   jr r31

Variante (mieux)
fmax :    move r4,r2     //pseudo instruction
          slt r9,r4,r5   // r9 =1 si r4 est le min
          movn r9,r2,r5  // r5 est le max
```

```
suite :   jr r31
```

b) Même question en assembleur ARM en passant les paramètres par registres, puis en les passant par la pile.

ARM – Passage par registres

```
        LDR R0, adresseB
        LDR R1, adresseC
        BL max
        STR R0, adresse A

Max :   STMFD R13!,{R14}        // sauve l'adresse de retour dans la pile (R13 est le SP)
        CMP R0, R1
        MOVLT R0, R1
        LDMFD R13!,{R15}        // charge l'adresse de retour dans CP (R13 est le SP)
```

ARM – passage des paramètres par la pile

```
        LDR R0, adresseB
        STR R0, [R13, #-4]!   // empile la valeur de R0
        LD R1, adresse C
        STR R1, [R13, #-4]!   // empile la valeur de R1
        BL max
        LDR R0, [R13, #4]     // récupère la valeur de R0
        STR R0, adresse A
        ADD R13, 13, #8       //nettoie la pile (les deux paramètres)

Max :   STMFD R13!,{R0-R1, R14}     // sauve les registres et l'adresse de retour dans la
                                    pile (R13 est le SP)
        LDR R0, [R13, #16 ]         //      récupère la valeur de A
        LDR R1, [R13, #12]          // récupère la valeur de B
        CMP R0, R1
        MOVLT R0, R1
        STR R0, [R13, #16]          // range le résultat
        LDMFD R13!,{R0-R1, R15}     // restaure les registres et charge l'adresse de retour
                                    dans CP (R13 est le SP)
```

Pointeur après entrée dans la procédure	R0 sauvegardé
	R1 sauvegardé
	Adresse de retour
Pointeur avant d'entrer dans la procédure	B
	A
Etat initial de la pile	

2. Procédures imbriquées

Soit le programme C suivant, qui trie un tableau de N octets signés en utilisant des procédures.

```
#include <stdio.h>
char v[10];

main()
{
v[0]=100;v[1]=80;v[2]=70;v[3]=60;v[4]=55;
v[5]=40;v[6]=35;v[7]=30;v[8]=25;v[9]=10;

tri (v,10);
}

change (char v[], int k, int m)
{
int temp;
temp=v[k];
v[k]=v[m];
v[m]=temp;
}

tri (char v[], int n)
{
        int i, j;
        for (i=n-1;i>0; i--){
                for (j=i-1; j>=0   ; j--){
                    if (v[j] >v [i])       change (v,j, i);
                     }
                }
}
```

Ecrire le programme le programme principal et les deux procédures en code NIOSII et en code ARM en passant les paramètres par registres.

NB : cet exercice aborde la question des procédures imbriquées. Il est évident que la programmation serait plus simple en intégrant directement le code de change dans la procédure de TRI.

CODE MIPS

```
main:
      addi $sp,$sp,-4
      sw $ra,0($sp)
      la $t0,V
```

```
      li $t1,10
   jal TRI
      lw $ra,0($sp)
      addi $sp,$sp,4
      jr $ra


TRI:
      addi $sp,$sp,-4
      sw $ra,0($sp)
      addi $t2,$t1,-1       #// initialisation de i
L1:   addi $t3,$t2,-1       #initialisation de j=i-1
      add $t4,$t2,$t0
L2: add $t5,$t3,$t0
      lb $t6,0($t4)         #v(i)
      lb $t7,0($t5)         #v(j)
      bge $t6,$t7, suite
      jal change
suite : addi $t3,$t3,-1     #j–
      bgez $t3,L2
      addi $t2,$t2,-1 #i–
      bgtz $t2,L1
      lw $ra,0($sp)
      addi $sp,$sp,4
      jr $ra


change:
           add $t4, $t0, $t2          # adresse de v[k]
           add $t5, $t0, $t3          # adresse de v[m]
           lb $t6, ($t4)         # r1 contient v[k]
           lb $t7, ($t5)         # r2 contient v[m]
           sb $t6, ($t5)
           sb $t7, ($t4)
           jr $ra
```

CODE ARM

Programme principal

```
ADR R4, adresse de V[0]      // chargement adresse de début du tableau (pseudo-
MOV R5, 10                                          instruction)
BL     TRI
```

TRI :

```
        STMFD       R13 !, {R14}
```

```
            SUB R6, R5, #1      // initialisation de i
            SUB R7, R6, #1      //initialisation de j
L1          LDR   R8, (R4+R6)   // V[i]
L2          LDR   R9, (R4+R7)   // V[j]
            CMP R8,R9
            BLLT  Change                // paramètres dans R4, R6, R7
            SUBS R7,R7,#1
            BGE   L2
            SUBS R6,R6, #1
            BGT L1
            LDMFD       R13!, {R15}

CHANGE
            STMFD       R13 !, {R14}
            LDR R1, (R4+R6)             // R1 contient v[k]
            LDR R2, (R4+R7)             //R2 contient v[m]
            STR R1, (R4+R7)
            STR R2, (R4+R6)
            LDMFD       R13!, {R15}
```

3. Annexes

Instructions pour évaluation des conditions et branchements

MIPS32

Registres

Nom registre	Numéro	Utilisation	Sauvegarde sur appel
$zero	**0**	**Valeur 0**	
$at	**1**	**Réservé pour assembleur**	
$v0-$v1	**2--3**	**Résultats et évaluation expression**	**non**
$a0-$a3	**4--7**	**arguments**	**oui**
$t0-$t7	**8--15**	**valeurs temporaires**	**non**
$s0-$s7	**16--23**	**valeurs sauvegardées**	**oui**
$t8-$t9	**24-25**	**temporaires supplémentaires**	**non**
$gp	**28**	**pointeur global**	**oui**
$sp	**29**	**pointeur de pile**	**oui**
$fp	**30**	**pointeur de trame**	**oui**
$ra	**31**	**adresse de retour**	**oui**

ARM

CMP, TST	CMP Rs1, Rs2	Rs1-Rs2 → Rcc

	TST Rs1, Rs2	Rs1 and Rs2 → Rcc
Instructions arithmétiques avec suffixe S : ADDS, SUBS, etc	SUBS Rd, Rs1, Rs2	Rd ← Rs1 – Rs2 Positionne Rcc
Bcond (LT, LE, GT, GE, EQ, NE…)	Bcond, déplacement	Si cond, alors CP ← NCP +déplacement
BL	BL déplacement	R14 ← NCP CP ← NCP + déplacement
BLcond	BLcond déplacement	Si cond, alors { R14 ← NCP CP ← NCP +déplacement}

Conventions logicielles

Pour ARM

- R0 à R3 : registres de travail, arguments des procédures
- R4 à R8 : variables dans des registres (à sauvegarder par les fonctions appelées)
- R9 : variable registre, base statique
- R10 : variable registre, limite pile
- R11 : pointeur de trame
- R12 : registre de travail
- R13 : pointeur de pile
- R14 : registre de lien, registre de travail
- R15 : compteur de programme