

T.D. 11 – Corrigé

La pile et les sous-programmes

Exercice 1

Soit le programme et les valeurs initiales ci-dessous :

```

001000 48E7C080  MOVEM.L D0/D1/A0,-(A7)

001004 7004      MOVEQ.L #4,D0
001006 7205      MOVEQ.L #5,D1
001008 41F84000  LEA      $4000,A0
00100C 6100000A  BSR      TAB
001010 2002      MOVE.L  D0,D2

001012 4CDF0103  MOVEM.L (A7)+,D0/D1/A0
001016 4E75      RTS

```

Valeurs initiales : D0 = \$12345678 A0 = \$11112222

 D1 = \$ABCDEF00 A7 = \$FFFFFFFC

 \$FFFFFF8 02 24 32 AF 00 00 20 00

1. Quelles instructions du programme vont modifier la pile ?

Quatre instructions vont modifier la pile :

```

001000 48E7C080  MOVEM.L D0/D1/A0,-(A7)    ①

001004 7004      MOVEQ.L #4,D0
001006 7205      MOVEQ.L #5,D1
001008 41F84000  LEA      $4000,A0
00100C 6100000A  BSR      TAB                    ②
001010 2002      MOVE.L  D0,D2

001012 4CDF0103  MOVEM.L (A7)+,D0/D1/A0    ③
001016 4E75      RTS                            ④

```

2. Pour chacune d'entre elles, indiquez le contenu de la pile.

État de la pile avant l'instruction ①	État de la pile après l'instruction ①	État de la pile après l'instruction ②	État de la pile après l'instruction ③	État de la pile après l'instruction ④
?	?	0000	0000	0000
?	?	1010	1010	1010
?	A7 → 1234	A7 → 1234	1234	1234
?	5678	5678	5678	5678
?	ABCD	ABCD	ABCD	ABCD
?	EF00	EF00	EF00	EF00
0224	1111	1111	1111	1111
32AF	2222	2222	2222	2222
A7 → 0000	0000	0000	A7 → 0000	0000
2000	2000	2000	2000	2000
?	?	?	?	A7 → ?

3. À quelle adresse se trouve la prochaine instruction à exécuter après le RTS ?

La prochaine instruction à exécuter se trouve à l'adresse \$2000. Il s'agit de l'adresse qui se trouve au sommet de la pile juste avant l'exécution du RTS.

Exercice 2

Soit la fonction ci-dessous rédigée en langage C :

```
long GetSol(short a, short b, short c)
{
    long delta;

    delta = b*b - 4*a*c;

    if (delta < 0)
        return 0;

    if (delta == 0)
        return 1;

    return 2;
}
```

1. Donnez l'équivalent en assembleur de l'instruction C suivante, sachant que les arguments sont passés à la fonction par la pile (le premier argument sera empilé en dernier) : `GetSol(5, 2, 1);`

Le mot clé `short` permet de déclarer des variables codées sur 16 bits signés. Il faudra donc empiler les arguments **a**, **b** et **c** de la fonction **GetSol()** avec une taille de 16 bits.

```

                                move.w  #1,-(a7)      ; argument c → pile
                                move.w  #2,-(a7)      ; argument b → pile
                                move.w  #5,-(a7)      ; argument a → pile
                                jsr      GetSol         ; Appel du sous-programme GetSol
RETOUR
```

L'instruction `jsr GetSol` empile l'adresse de retour sur 32 bits (adresse située juste après le `jsr`) puis saute au sous-programme **GetSol**. Le contenu de la pile après l'instruction `jsr` (avant l'exécution de la première instruction du sous-programme **GetSol**) est donc le suivant :

	?	
	?	
	?	
	?	
	?	
	?	
A7 →	RETOUR _{31:16}	← 16 bits de poids fort de l'adresse de retour
	RETOUR _{15:0}	← 16 bits de poids faible de l'adresse de retour
	0005	← a
	0002	← b
	0001	← c

2. La valeur de retour se trouvant dans le registre **D0**, donnez l'équivalent en assembleur de la fonction **GetSol ()**. Vous commencerez par placer les arguments **a**, **b** et **c** dans les registres **D1**, **D2** et **D3**.

```

GetSol      ; Sauvegarde les registres d1, d2 et d3 dans la pile.
            ; Cette sauvegarde se fait sur 32 bits afin de pouvoir
            ; modifier entièrement les registres.
            movem.l d1-d3,-(a7)

            ; Les arguments situés dans la pile sont copiés dans
            ; les registres d1, d2 et d3 : a → d1, b → d2 et c → d3.
            ; Attention : les arguments n'ont pas été dépilés.
            ; (Il n'est pas possible de les dépiler ici, car au sommet
            ; de la pile se trouve la sauvegarde des registres.)
            movem.w 16(a7),d1-d3

            ; Calcul de delta.
            muls    d2,d2      ; b²      → d2
            muls    d3,d1      ; ac      → d1
            lsl.l   #2,d1      ; 4ac     → d1
            sub.l   d1,d2      ; b²-4ac  → d2

            ; Saut au label NUL si delta = 0
            beq.s   NUL

            ; Saut au label POSITIF si delta > 0
            bgt.s   POSITIF

NEGATIF     ; Gestion du cas où delta est négatif.
            ; 0 solution → d0, puis sortie du sous-programme.
            moveq.l #0,d0
            bra     QUIT

NUL         ; Gestion du cas où delta est nul.
            ; 1 solution → d0, puis sortie du sous-programme.
            moveq.l #1,d0
            bra     QUIT

POSITIF     ; Gestion du cas où delta est positif.
            ; 2 solutions → d0, puis sortie du sous-programme.
            moveq.l #2,d0

QUIT        ; Gestion de la sortie du sous-programme.

            ; Restaure les valeurs initiales des registres d1, d2 et d3.
            movem.l (a7)+,d1-d3

            ; Ajustement de la pile (libère la pile des arguments).
            ; - Place l'adresse de retour en bas de la pile.
            ; - Place le pointeur de pile sur l'adresse de retour.
            move.l  (a7),6(a7)
            addq.l  #6,a7

            ; Dépile l'adresse de retour et la place dans le PC.
            ; (= saut à l'adresse située après un jsr GetSol.)
            rts

```