DESS CCI : Corrigé Langage Machine, Septembre 2006

Deux heures, tous documents et calculatrices autorisés. Ordinateurs (PC) interdits.

1 Entiers et variables en mémoire

On considère les déclarations C suivantes.

```
short int s1 = -3;
long l1 = 0x12345678;
short int s2;
long l2;
```

On suppose que la section **data** début à l'adresse 0x1000 et la section bss à l'adresse 0x1100.

Question a : Donner en hexadécimal la représentation de -3.
0xfffd

Question b : Dessiner le contenu de la mémoire pour chacune des deux sections

```
             048c            159d             26ae          37af
1000         0xfd            0xff             00            00
             <       s1           >          <   alignement     >
1004         0x78            0x56             0x34          0x12
             <                  l1                              >

1100         00              00               00            00
             <       s2           >          <   alignement     >
```

Question c : Ecrire en langage d'assemblage une suite d'instruction qui copie le contenu de s1 dans s2 et celui de l1 dans l2.

```
ldr     r0,= s1
ldrsh   r1, [r0]
ldr     r0, =s2
strh    r1, [r0]

ldr     r0,= l1
ldr     r1, [r0]
ldr     r0, =s2
str     r1, [r0]
```

2 Boucle et pointeur

```
#define TAILLETAB 6

unsigned long tableau [TAILLETAB] = {2,4,6,8,10,0};

unsigned long s;

/*************************************************************************/
/* Calcul de la somme des éléments d'un tableau                          */
/* jusqu'au premier élément à 0 inclus                                   */
/* s       : adresse de l'emplacement de stockage de la somme            */
/* t       : adresse du tableau                                          */
/*************************************************************************/

void somme_tab (unsigned long *s, unsigned long *t)
{
*s = 0;
while (*t != 0)
  {
  *s = *s + *t;
  t++;
  }
}

int main (int argc, char *argv[], char*envp[])
{
somme_tab (&s, tableau);
return 0;
}
```

Question : Traduire le programme ci-dessus en langage d'assemblage en tenant en particulier compte du fait que les variables entières sont de type **naturel** (unsigned).

```
                .data
tableau:        .word  2
tableau:        .word  4
tableau:        .word  6
tableau:        .word  8
tableau:        .word  10

                .bss
s:              .skip  4

                .text
```

```
@         rappel stockage premier  paramètre :  s dans r0
@         rappel stockage deuxième paramètre :  t dans r1
@         r3, r4 utilisables comme registre de travail

somme_tab:
                    @  *s = 0
                    mov   r3, #0
                    str   r3, [r0]

                    @ while
                    ba     test_while

corps:              @ *s = *s + *t
                    ldr   r3, [r0]        @ r3 = *s
                    ldr   r4, [r1]        @ r4 = *t
                    add   r3, r3, r4      @ *s + *t
                    str   r3, [r0]        @ *s = r3

                    @ t++
                    add   r1, r1, #4

test_while:         @ contenu de *t déjà dans r4
                    @ on omet l'instruction  ldr  r4, [r1]
                    cmp   r4, #0
                    bne   corps

                    mov   pc, lr

main:
                    @ somme (&s, tableau)
                    @ r0 = &s
                    ldr    r0, =s
                    @ r1 = tableau
                    ldr    r1,= tableau
                    @ appel
                    bl     somme_tab

                    @retour
                    mov    pc, lr
\end{itemize}

\section{Procédure à nombre quelconque d'arguments}

\begin{verbatim}
        .global  somme_liste
```

```
somme_liste:
        mov     ip, sp
        stmfd   sp!, {r0, r1, r2, r3}
        stmfd   sp!, {fp, ip, lr, pc}
        sub     fp, ip, #20
        add     r1, fp, #8
        ldr     r0, [fp, #4]
        bl      somme_tab
        ldmea   fp, {fp, sp, pc}
```

Question a : Dessiner la pile au moment où le premier branchement à la procédure **bl somme_liste** est exécuté.

Les arguments &s, 1,2 et 3 sont stockés dans les registres r0 à r3. Les suivants sont empilés.

```
Sp -->          3
                4
                5
                6
                0
                ...
fp -->
```

Question a : Dessiner l'évolution de la pile (par rapport au dessin précédent) au moment où le premier branchement à la procédure **bl somme_tab** est exécuté.

La procédure empile le bloc de registres r0 à r3, puis les 4 mots standards (fp, ip, lr, pc).

```
Sp --->         ancien fp  ----------------------------
                ancien sp  ---------------------
                adresse de retour dans main
Fp --->         @ corps de somme_liste
                adresse de s
                1
                2
                3  <--- ancien sp -------------
                4
                5
                6
                0
                ...
                  <--- ancien fp  ---------------------
```

Question c : Traduire en langage d'assemblage le premier appel de **somme_liste** dans le corps de main.

```
@ empiler 0
mov   r0, #0
sub   sp, sp, #4
str   ro, [sp]

@ empiler 6
mov   r0, #6
sub   sp, sp, #4
str   ro, [sp]

@ idem pour 5 et 4

mov   r3, #3
mov   r2, #2
mov   r1, #1
ldr   r0,= s

bl    somme_liste
```

Question d : Décrire comment fonctionne la procédure **bl somme_liste**. Commenter son code en langage d'assemblage : expliquer le rôle de chacune des sept dernières instructions.

Le deuxième stmfd et le ldmea respectivement sauvegardent dans la pile et restaurent les anciens sommets de pile et l'adresse de retour (+ l'adresse du corps de somme_pile).

Le premier stmfd empile les premiers paramètres de telle sorte que tous les paramètres forment un tableau contigu stocké dans la pile.

Habituellement, on trouve l'instruction sub fp, ip, #4 pour déplacer le contenu de fp juste au-dessus du blocs des paramètres empilés par l'appelante. Ici, on se déplace de 4 mots de plus pour tenir compte de l'empilement des 4 premiers paramètres (4 + 4*4 donne 20). Le paramètre somme se trouve à l'adresse fp+4 et le suivant en fp+8.

L'instruction add passe dans r1 (deuxième paramètre de comme_tab) le contenu du deuxième paramètre, ici 1.

L'instruction ldr calcule l'adresse du paramètre somme et le stocke dans r0 (premier paramètre de somme_tab)..

L'instruction bl efectue le branchement aller vers comme_tab.