DESS CCI : examen 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.

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

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.

Procédures et tableaux

Rappelons quelques points de la convention d'appel utilisée par le compilateur gcc pour ARM :

1. L'adresse de retour dans l'appelante est stockée dans le registre **lr**.
2. Les quatre premiers paramètres sont passés dans les registres r0 à r3. La procédure appelée peut modifier ces paramètres.
3. Le résultat des fonctions est retourné à la place du premier paramètre, dans le registre **r0**.
4. La sauvegarde des registres est à la charge de la procédure appelée, excepté pour le registre **ip**.

On veut écrire une procédure capable de faire la somme d'une liste d'entiers.

2 Boucle et pointeur

Dans un premier temps, on suppose que la liste d'entiers est rangée dans un tableau stocké dans la section **data**. La procédure somme_tab a deux paramètres : l'adresse à laquelle le cumul des valeurs entières doit être rangé et l'adresse du tableau.

```
#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).

3 Procédure à nombre quelconque d'arguments

Les entiers à sommer sont maintenant passés comme une liste d'arguments passés à une procédure à nombre quelconque d'arguments : somme_liste.

Dans la déclaration de **somme_liste**, la présence de **...** inidque une liste d'un nombre quelconque de paramètres sans noms.

```
void somme_liste (unsigned long *somme, ...)
{
somme_tab (somme, (long *) (&somme + 1));
}

int main (int argc, char *argv[], char*envp[])
{
unsigned long r = 5;
somme_liste (&s,1,2,3,4,5,6,0); /* un appel à 7+1 arguments */
somme_liste (&r,2,4,6,8,0);     /* un appel à 5+1 arguments */
somme_liste (&r,2,3,4,0);       /* un appel à 4+1 arguments */
}
```

Voici la traduction en langage d'assemblage de la procédure **somme_liste** :

```
        .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.

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.

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

Question d : Décrire comment fonctionne la procédure **bl somme_liste**. Commenter son code en langage d'assemblage : expliquer le rôle (pas la sémantique [1]) de chacune des sept dernières instructions.

[1] Exemple avec ldr r0, [fp #4]. Sémantique : r0 $\leftarrow$ Mem[fp +4]. Rôle : r0 correspond à ..., on y charge le contenu de ...