M2P CCI : examen Langage Machine, Novembre 2012

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

Table des matières

1 Tableaux et fonctions (90mn)

On considère le programme suivant qui recherche la valeur et l'indice de l'élément minimum d'un tableau :

```
#include <stdio.h>
#include <string.h>

char format1 [13] = "tab[0] = %d\n";
char format2 [24] = "mini = %hu , indice %u\n";
unsigned short tab [9] = {3,6,8,7,2,1,9,0,5};
unsigned short *p = tab;

void chercher_mini (unsigned int n,               // n dans r0
                    unsigned short **ptr_mini)    // ptr_mini dans r1
{
   register unsigned short min;      // a stocker dans r4
   register unsigned short *p_min;   // a stocker dans r5
   register unsigned short *p;   // a stocker dans r6

   min = tab[0];
   p_min = tab;

  for (p=tab+1; p<tab+n;p++)
  {
    if (*p < min)
      {
      min = *p;
      p_min = p;
      }
```

```
  }
  *ptr_mini = p_min;
}

int main (void)
{
   register unsigned short m; // a stocker dans r6
   register unsigned int indice;        // a stocker dans r7

   printf (format1,*p);
   chercher_mini (8,&p);
   m = *p;
   indice = p-tab;
   printf (format2, m,indice);

   return 0;
}
```

La convention d'appel de chercher_mini est la suivante :

- le premier paramètre (n) est passé dans le registre r0.
- le deuxième paramètre (ptr_mini) est passé dans r1.
- L'adresse de retour est passée dans lr.

1.1 Section .data (15mn)

Traduire en langage d'assemblage ARM les déclarations de format1, format2, tab et p.

L'initialisateur de format1 est une chaîne de douze caractères (\n représente un seul caractère), celui de tab un ensemble de 9 entiers. **Pourquoi** le tableau format1 est-il déclaré de taille 13 alors que tab est bien de taille 9 ?

1.2 Chercher_mini (parcours de tableau avec pointeur) (40mn)

Voici un squelette de la traduction de chercher_mini en langage d'assemblage ARM à compléter :

```
                .global chercher_mini
                .text
chercher_mini:
prologue :      stmfd   sp!, {r4-r9}     @ sauver r4 a r9 dans la pile
corps :         ...                      @ traduire ici min = tab[0]
                                         @ jusqu'a *ptr_mini = p_min
```

```
epilogue:       sp!,{r4-r9}              @ restaurer r4 a r9
                ...
```

Traduire les instructions **min = tab[0] et p_min = tab**.

Traduire la totalité de la boucle for (if inclus).

Traduire la fin de la fonction.

1.3 Main (Appel de fonction, arithmétique sur les pointeurs) (20mn)

Traduire en langage d'assemblage l'instruction **chercher_mini (8,&p)**.

Traduire en langage d'assemblage l'instruction **indice = p-tab**. Attention : si deux pointeurs p1 et p2 repèrent respectivement tab[i] et tab[j], alors l'expression p1-p2 représente i-j.

1.4 Parcours du tableau par indice (15mn)

Voici une autre version de chercher_mini utilisant une variable de boucle de type indice.

```
void chercher_mini (unsigned int n,                // n dans r0
                    unsigned short **ptr_mini)     // ptr_mini dans r1
{
   register unsigned short min;       // a stocker dans r4
   register unsigned indice_min;      // a stocker dans r5
   register unsigned int i;           // a stocker dans r6

   min = tab[0];

  for (i=0;i<n;i++)
  {
    if (tab[i] < min)
      {
      min = tab[i];
      indice_min = i;
      }
  }
  *ptr_mini = tab + i;
}
```

Répondre aux questions suivantes sans rajouter de variable de boucle de type pointeur :

- **Traduire** l'instruction if (corps du if inclus).
- **Traduire** l'instruction ***ptr_mini = tab + i**.

2 Fonction mystère (branchements et appels, if) (30mn)

Voici une traduction manuelle d'une fonction lettre dont le prototype est le suivant : **void lettre (char c)**.

```
        .global lettre

        .data
table:  .word  voy_hexa
        .word  cons_hexa
        .word  cons_hexa
        .word  cons_hexa
        .word  voy_hexa
        .word  cons_hexa
        .word  cons
        .word  cons
        .word  voy
        .word  cons
        .word  cons
        .word  cons
        .word  cons
        .word  cons
        .word  cons
        .word  voy
        .word  cons
        .word  cons
        .word  cons
        .word  cons
        .word  cons
        .word  voy
        .word  cons
        .word  cons
        .word  cons
        .word  voy
        .word  cons

       .text
m1:    .asciz  "N'est pas une minuscule\n"
m2:    .asciz  "consonne\n"
m3:    .asciz  "voyelle\n"
m4:    .asciz  "consonne hexa\n"
m5:    .asciz  "voyelle hexa\n"
```

```
        .global lettre
        .balign  4

lettre: stmfd  sp!, {r0-r2,lr}

        ldr    lr,= fin

        cmp    r0, #'a'
        blo    pas_min
        cmp    r0, #'z'
        bhi    pas_min

        sub    r1,r0,#'a'
        mov    r1, r1, LSL #2
        ldr    r2,= table
        ldr    pc, [r2,r1]

pas_min:
        ldr  r0,= m1
        b     printf

cons_hexa:
        ldr  r0,= m4
        b   printf

cons:
        ldr  r0,= m2
        b   printf

voy_hexa:
        ldr  r0,= m5
        b   printf

voy:
        ldr  r0,= m3
        b   printf

fin:    ldmfd  sp!, {r0-r2,lr}
        mov    pc,lr

.ltorg
```

Cette fonction affiche un message décrivant une information sur la nature du caractère reçu en paramètre.

Qu'affiche-t-elle pour

1. '0'
2. 'Z'
3. 'a'
4. 'b'
5. 'e'
6. 'g'
7. 'i'

Expliquer pourquoi

- Printf n'est exécuté qu'une seule fois par appel de lettre alors qu'il n'y a aucun branchement entre deux appels de printf et le branchement à printf s'effectue par b et non par bl.
- Lr doit être sauvé dans le prologue et restauré dans l'épilogue.

Pourrait-on effectuer le branchement à printf sans utiliser de branchement relatif (ni b ni bl) ?

- Si oui, avec quelle instruction ou séquence d'instructions ARM ?
- Si non, expliquer pourquoi ce n'est pas possible.

Donner une séquence de code ARM se comportant de la même manière que la fonction lettre mais n'utilisant aucun tableau. Les deux méthodes sont-elles comparables en nombre d'instructions exécutées (justifier brièvement votre conclusion) ?