M2P CCI : examen Langage Machine, Novembre 2017

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

Table des matières

Conventions communes à toutes les questions

Toutes les variables locales des fonctions et procédures sont stockées dans des registres (le numéro de registre à utiliser est indiqué en commentaire). Seules les variables déclarées à l'extérieur des fonctions et sans l'attribut register sont stockées en mémoire. Types uint32_t, uint16_t, uint8_t : entiers naturels sur 32, 16 et 8 bits.

1 Endianness (fontions et tableaux) (1h15)

Dans la traduction des fonctions, respecter l'allocation des registres décrite en annexe et la convention d'appel suivante :

— Tous les paramètres sont passés par les registres et non en mémoire
— Le premier paramètre (à gauche) est passé dans le registre r0
— Le deuxième paramètre est passé dans le registre r1

1.1 Présentation sans question (5mn)

On considère 3 tableaux d'entiers naturels de tailles 16, 32 et 64 bits, représentés selon la convention Big ou Little Endian de la machine (détails en annexe 5).

```
uint64_t t64[2] = { 0x0123456789abcdef, 0xefcdab8967452301 };
uint32_t t32[4] = { 0x01234567, 0x89abcdef, 0x67452301, 0xefcdab89 };
```

```
uint16_t t16[8] = { 0x0123, 0x4567, 0x89ab,0xcdef,
                    0x6745, 0x2301, 0xefcd, 0xab89 };
```

Les 3 paramètres de la procédure reverse_endian (cf annexe 1) sont :

1. tab : adresse du tableau
2. indice : numéro de l'élément de tableau à modifier
3. taille (d'élément) : sizeof(type), type $\in$ {uin64_t, uint16_t,uint8_t}

Reverse_endian permet de changer la convention de représentation d'un élément de tableau [1] :

— Après reverse_endian(t32,2,4), l'élément t32[2] contiendra 0x01234567
— Après reverse_endian(t16,0,2), l'élément t16[0] contiendra 0x2301

1.2 Traduction de swap_bytes (20mn)

Traduire en langage d'assemablage ARM le corps (boucle for) de swap_bytes. Le prologue et l'épilogue ne sont pas demandés.

1.3 Traduction de reverse_endian (50mn)

Le code de reverse_endian comprend un prologue (sauvegarde des registres), le corps proprement dit, et un épilogue (restauration des registres et retour).

La pile n'étant utilisée que pour la sauvegarde des registres modifiés dans le corps de la fonction, seul sp sera utilisé (et pas fp).

Réécrire en C le corps de reverse_endian en supprimant tous les opérateurs d'indexation (crochets [ ]) : 5mn.

Traduire en langage d'assemblage l'appel de swap_bytes (10mn)

Traduire le reste du corps (20mn)

Ecrire le prologue et l'épilogue (15mn) :

1. établir au préalable la liste des registres à sauvegarder
2. n'utiliser que les instructions sub, str, et mov.

2 Listes : structures, procédures et pile (45mn)

La convention d'appel de la fonction listmin (voire annexe 2) est la suivante :

1. L'adresse de retour est passée dans le registre lr

1. ce genre de code est notament utile pour la mise en réseau de machines d'endianness $\neq$

2. Tous les paramètres explicites sont passés dans la pile, le premier (cl) étant en sommet de pile.

Cette fonction parcourt une liste circulaire donnée en premier paramètre et modifie le pointeur passé en deuxième paramètre pour qu'il piointe sur l'éléement de liste de plus petite valeur.

Le type doublet_t est défini comme suit :

```
typedef struct doublet {
struct doublet *next;
uint16_t value;
} doublet_t;
```

2.1 Déclaration de variables (20mn)

Le fichier data.c déclare deux listes chaînées circulaires créées statiquement composées respectivement de :

1. 5 éléments d0 à d4
2. un seul élément single_cl

```
/* Extern : type specification only : no memory allocation    */

extern doublet_t d1,d2,d3,d4,d5;

/* Circular Linked list 5,3,1,2,4 */

doublet_t d0 = {&d2,5};
doublet_t d1 = {&d0,4};
doublet_t d2 = {&d4,3};
doublet_t d3 = {&d1,2};
doublet_t d4 = {&d3,1};

doublet_t *circlist = &d0;

/* Single element circular list */
doublet_t single_cl = {&single_cl,6};

doublet_t *ptrmin;
```

Traduire en langage d'assemblage les déclarations de variables précédentes.

2.2 Appel de listmin (10mn)

La routine ci-dessus est une version raccourcie du programme principal donné en annexe.

```
void main ()
{
  afficher ();
  listmin(circlist,&ptrmin);
  circlist=&d2;
  afficher ();
}
```

Traduire en code ARM le corps cette fonction main (le prologue et l'épilogue de main ne sont pas demandés.

2.3 Listmin (15mn)

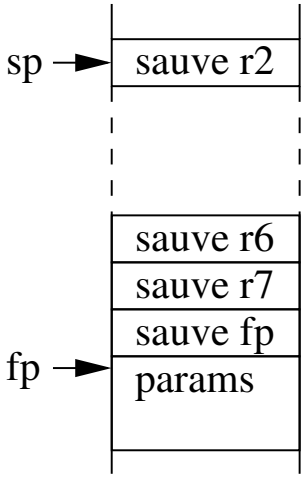
Voici le prologue et l'épilogue de listmin, ainsi qu'un dessin de la pile au début du corps.

```
            .text
@ void listmin (doublet_t *cl, doublet_t **pmin)

listmin:
prologue:   stmfd  sp!,{r2-r7,fp}
            add    fp,sp,#28

corps:      /* non détaillé , dessin de pile ici */

epilogue:   ldmfd  sp!,{r2-r7,fp}
            mov    pc,lr
```



Traduire l'extrait de code C suivant :

```
valmin = (*c1).value;
*pmin = cl;
```

3 Annexe 1 : code de sawp_bytes et reverse_endian

Pour la traduction en lanagage d'assemblage ARM, la convention d'affectation des variables locales aux registres est imposée (cf tableaux suivant,les déclarations de variables locales).

```
void swap_bytes(uint32_t size,
            uint8_t *address) {

  register uint8_t   tmp1;
  register uint8_t   tmp2;
  register uint32_t     i;
  register uint32_t tm1mi;

/* Si un temporaire supplémentaire
est nécessaire, utiliser r7 */
```

choix regs imposé			
tmp1	tmp2	i	tm1mi
r3	r4	r5	r6

```
  for (i=0; i<size/2; i++) {
    tm1mi = size - 1 - i;
    tmp1 = address[i];
    tmp2 = address[tm1mi];
    address[i] = tmp2;
    address[tm1mi] = tmp1;
  }
}
```

```
void reverse_endian (void *tab,
                     uint32_t indice,
                     uint32_t taille) {

  register uint64_t  *ad64;
  register uint32_t  *ad32;
  register uint16_t  *ad16;
  register uint8_t   *ad8;
```

choix regs imposé			
ad64	ad32	ad16	ad8
r4	r5	r6	r7

```
  ad64=tab;
  ad32=tab;
  ad16=tab;

  if (taille==8) {
    ad8 = (uint8_t *) & (ad64[indice]);
  } else if (taille==4) {
    ad8 = (uint8_t *) & (ad32[indice]);
  } else if (taille==2) {
    ad8 = (uint8_t *) & (ad16[indice]);
  } else
    return;

  swap_bytes(taille,ad8);
}
```

4 Annexe 2 : code de gestion de liste

```
void listmin (doublet_t *cl, doublet_t **pmin) {
  register uint16_t valmin;        /* utiliser r4 */
  register doublet_t *pdoublet;    /* utiliser r5 */

  if (cl == NULL) return;

  valmin = (*cl).value;           /* ou vamin = cl -> value */
  pdoublet = cl;
  *pmin = cl;
```

```
  do {
     pdoublet = pdoublet -> next;
     if (pdoublet -> value < valmin) {
       valmin = pdoublet -> value;
       *pmin= pdoublet;
     }
  } while (pdoublet != cl);
}

void afficher () { /* non détaillé * }

void main ()
{
  afficher ();
  listmin(circlist,&ptrmin);
  afficher ();
  circlist=&d2;
  listmin(circlist,&ptrmin);
  afficher ();
  listmin(&single_cl,&ptrmin);
  printf ("single_cl = %p, ptrmin = %p, val = %u\n",&single_cl,ptrmin, single_cl.valu
}
```

5 Annexe 3 : endianness et contenu des tableaux

Dans la représentation Big Endian, le premier octet de l'entier contient la paire de chiffres hexacimaux de poid forts et le dernier octet ceux de poids faibles. Avec la convention Little Endian, le premier octet contient au contraire les chiffres de poids faibles.

Contenu des premiers octets de t64 si stocké en 0x60000								
Endian	60000	60001	60002	60003	60004	60005	60006	60007
Big	0x01	0x23	0x45	0x67	0x89	0xab	0xcd	0xef
Little	0xef	0xcd	0xab	0x89	0x67	0x45	0x23	0x01
Contenu des premiers octets de t32 si stocké en 0x30000								
Endian	30000	30001	30002	30003	30004	30005	30006	30007
Big	0x01	0x23	0x45	0x67	0x89	0xab	0xcd	0xef
Little	0x67	0x45	0x23	0x01	0xef	0xcd	0xab	0x89
Contenu des premiers octets de t16 si stocké en 0x10000								
Endian	10000	10001	10002	10003	10004	10005	10006	10007
Big	0x01	0x23	0x45	0x67	0x89	0xab	0xcd	0xef
Little	0x23	0x01	0x67	0x45	0xab	0x89	0xef	0xcd