M2P CCI : examen Langage Machine, Novembre 2007

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

Table des matières

1 Introduction (15mn)

On considère un programme qui calcule une suite de nombres à partir d'une valeur initiale, la range dans un tableau nommé valeurs, et en calcule la somme.

La procedure di2_p__mu3pl1_imp calcule la prochaine valeur de la suite à partir de la valeur courante :

```
/* fichier di2_p__mu3pl1_imp.h */
void di2_p__mu3pl1_imp (unsigned short x, unsigned short *y);
```

Les fichiers di2_p__mu3pl1_imp.c et di2_p__mu3pl1_imp.s (fichier source et traduction manuelle) sont données section 3. Ils sont incomplets : une partie de code détaillée dans l'un est omise dans l'autre.

La procedure calculer contient la boucle qui appelle di2_p__mu3pl1_imp et remplit le tableau. Le fichier calculer.c est détaillé section 4.

Le fichier main.c contenant le programme principal est donné en annexe.

```
/* fichier calculer.h */
#define LONGUEUR_MAX 51

extern unsigned short v2;
```

```
extern unsigned long somme;
extern unsigned short valeurs [LONGUEUR_MAX];

void calculer (unsigned short initial);
```

Voici d'autre part un petit rappel sur le jeu d'instructions ARM : un décalage à gauche (**LSL**[1]) ou à droite (**LSR**[2]) peut être appliqué à l'opérande droit d'une instruction et le suffixe **S** indique que l'instruction modifie les indicateurs (NZCV).

1. **mov r1, r2, LSL #3** prend une copie du contenu de r2, la décale de trois bits à gauche, la stocke dans r1 et ne change pas les indicateurs.
2. **addS r1, r2, r3, LSR #2** prend une copie du contenu de r3, la décale de deux bits à droite, l'ajoute au contenu de r2, stocke la somme dans r1 et met à jour les indicateurs.

2 Questions

Indication de barême : selon l'estimation approximative de la durée des questions.

La convention d'appel de procédure utilisée est la suivante : passage des 4 premiers paramètres dans les registres r0 à r3, les autres dans la pile. Dans la traduction en langage d'assemblage, vous pouvez omettre le détail des sauvegardes de registres dans le prologue et l'épilogue (un commentaire suffit). Les registres à utiliser pour stocker les variables locales sont les registres r5 et suivants.

2.1 Traduction des déclarations des variables (15mn)

Traduire en langage d'assemblage les déclarations des variables v2, valeurs et somme.

Traduire en langage d'assemblage l'affectation ***y = r**.

2.2 Base 2 et traduction inverse asm → C (40 mn)

Les instructions **and r2,r1,r0** en langage d'assemblage et **r2 = r1 & r0** en C affectent à r2 le **ET bit à bit** [3] entre r1 et r0. Chaque chiffre de la représentation des entiers en base deux est considéré comme un booléen (0 pour Faux et 1 pour Vrai).

Le bit de rang i de r2 est 1 si et seulement si le bit de rang i de r1 est 1 **et** le bit de rang i de r0 est 1. Si le bit de rang i de (au moins) un opérande est 0, le bit de rang i du résultat est aussi 0. Pour un **ou** bit à bit, le bit du résultat est 1 si (au moins un) des bits des opérandes est 1. En voici un exemple :

[1] LSL correspond à « en langage C.
[2] LSR correspond à » en langage C.
[3] Le **OU** bit à bit s'écrirait or r2,r1,r0 et r2 = r1 | r0.

```
@ Exemple de OU bit a bit   0xb7 est égal à 0xb3 | 0x16
@     1011 0011  r1                 r1 = 0xb3
@  |  0001 0110  r0                 r0 = 0x16
@    ----------
@     1011 0111  r2                 r2 = 0xb7
@     . ..  ...       colonnes avec r1_i == 1 ou  r0_i == 1
```

Calculer et écrire en hexadécimal les expressions ET bit à bit suivantes : $0xa$ ET_{bb} 0x1, 0xb ET_{bb} 0x1, $0xc$ ET_{bb} $0x1$, 0xd ET_{bb} 0x1.

Quelle est la propriété commune des entiers naturels tels que leur ET bit à bit avec l'entier 1 est non nul ?

A quelles opérations arithmétiques sur un entier naturel correspondent le décalage d'un bit à droite et d'un bit à gauche ?

Vous disposez dans la section 3 de la traduction en langage d'assemblage du calcul de la valeur de r en fonction de celle de x. **Ecrivez la séquence de code C** dont elle est la traduction.

Le calcul est décrit à l'aide de constructeur(s) algorithmique(s) classique(s) (**if**, **while**, **do** ... **while**) et d'opérations arithmétiques usuelles sur les entiers naturels : addition (+), soustraction (-), multiplication (*) , division (/) et modulo (%)[4].

La séquence C d'origine ne contient ni **goto** ni opérateur de décalage (« ou ») [5]

2.3 Traduction d'une boucle (50mn)

Traduire en langage d'assemblage la procédure calculer.

Réécrire en C la boucle en remplaçant le parcours du tableau par pointeur par un parcours du tableau par indice (en utilisant une seule variable de boucle).

3 Code de la procédure di2_p__mu3pl1_imp

Le calcul de r à partir de x a été omis dans le fichier di2_p__mu3pl1_imp.c.

```
/* fichier di2_p__mu3pl1_imp.c (incomplet) */
#include <stdio.h>
#include "di2_p__mu3pl1_imp.h"

void di2_p__mu3pl1_imp (unsigned short x, unsigned short *y)
```

[4] Rappel : modulo signifie reste de la division entière.

[5] Vous pouvez écrire l'équivalent en if ... goto, puis retrouver les constructions algorithmiques dont ils sont la traduction.

```
  {
  unsigned short r;
  ...    /* calcul de r a partir de x omis */
  *y = r;
  }
```

La traduction de l'affectation ***y** = **r** a été omise dans le fichier di2_p__mu3pl1_imp.s

```
@ fichier di2_p__mu3pl1_imp.s

                    .global di2_p__mu3pl1_imp
                    .text

@ convention d'appel : x dans r0 et y dans r1
@
@ La convention d'appel stipule que les contenus des registres
@ r0 a r3 peuvent etre detruit par la procedure appelee
@  ----->  on peut omettre la sauvegarde de r2


di2_p__mu3pl1_imp:   andS  r2, r0, #1      @ calcul r = f(x)  : debut
                     bne   pas_nul
nul:                 mov   r2, r0, LSR #1
                     b     sortie
pas_nul:             add   r2, r0, r0, LSL #1
                     add   r2, r2, #1      @ calcul r = f(x)  : fin
sortie:              ...                   @ traduction de *y = r omise
```

4 Code de la procédure calculer

```
/* fichier calculer.c */
#include <stdio.h>
#include "di2_p__mu3pl1_imp.h"
#include "calculer.h"

unsigned short v2;

unsigned long somme = 0;
unsigned short valeurs [LONGUEUR_MAX];

void calculer (unsigned short initial)
{
register unsigned short *pval;
register unsigned short v1;
```

```
pval = valeurs;
v1 = initial;

while ((pval < (valeurs + LONGUEUR_MAX)) && (v1 != 1))
  {
  di2_p__mu3pl1_imp (v1,&v2);
  *pval = v2;
  somme += v2;
  pval++;
  v1 = v2;
  }
}
```

5 Annexe : code du programme principal

```
#include <stdio.h>
#include "calculer.h"

unsigned short a;

void afficher (void)
  {
  int i;
  for (i=0; i<LONGUEUR_MAX; i++)
    {
    printf ("Valeur [%d] = %u\n",i,valeurs[i]);
    if (valeurs [i] == 1) break;
    }
  printf ("sigma = %lu\n",somme);
  }

int main (int argc, char *argv[], char *envp[])
{
   unsigned int l;

   sscanf (argv[1], "%u", &l);
   a = l;
   calculer (a);
   afficher ();
   return 0;
}
```