Master CCI

Langage machine

Contrôle continu écrit 2012

Durée 1h30, documents autorisés, calculatrices et ordinateurs interdits

Table des matières

1 Base2 et accès à la mémoire (50mn)

1.1 Précisions et rappels sur les décalages (pas de question)

- En C, l'expression $a \ll b$ signifie valeur(a) décalée à gauche de valeur(b) bits. Réciproquement $a \gg b$ signifie valeur(a) décalée à droite de valeur(b) bits. Le décalage est de type arithmétique si l'entier a est de type relatif (int), et logique sinon (unsigned int).
- Lorsque l'opérande droit d'un calcul est un registre, les isntructions ARM permettent de lui appliquer une opération de décalage à gauche ou à droite. Le nombre de bits du décalage peut être une constante immédiate ou le contenu d'un registre. Le décalage est appliquée sur la copie du contenu du registre utilisée dans l'instruction de calcul : il ne modifie pas le contenu du registre utilisé.

1.2 Accès aux variables en mémoire (35mn)

On veut traduire le programme source1.c en un fichier source1.S (langage d'assemblage ARM) :

```
#include <stdio.h>

unsigned int x = 7;
unsigned int y;
unsigned int z;

int main (void)
{
  scanf("%u",&y);    // verification du code de retour de scanf omise
```

```
  printf ("x=%u, y=%u, z=%u\n",x,y,z);

  // traduire en langage d'assemblage ces deux instructions
  x = (y<<3) + y;
  z = x/4;

  printf ("x=%u, y=%u, z=%u\n",x,y,z);
  return 0;
}
```

Traduire la déclaration des variables en supposant que la taille d'un int ou unsigned int est un mot de 32 bits.

Traduire en langage d'assemblage ARM chacune des deux affectations entre les appels à printf. Traduire chaque affectation indépendament l'une de l'autre puis indiquer les éventuelles optimisations de code possible lorsque l'on traduit les deux instructions ensemble.

On considère à présent le programme source2.c :

```
#include <stdio.h>

unsigned char  x = 'a';
short int y;
int z;

int main (void)
{
  scanf("%hd",&y);    // verification du code de retour de scanf omise
  printf ("x=%c, y=%hd\n",x,y);

  // traduire en langage d'assemblage ces deux instructions
  x++;
  z = y*4;

  printf ("x=%c, y=%hd, z=%d\n",x,y,z);
  return 0;
}
```

Traduire en langage d'assemblage la déclaration des variables et les deux affectations en supposant que sizeof(short)=1 et sizeof(int)=4.

1.3 Base 2 (15mn)

On suppose que la machine représente les entiers relatifs selon la convention habituelle du complément à 2.

Donner en hexadécimal et en décimal le contenu de la variable x (de taille 32 bits) à la fin de cette séquence de code (expliquer pourquoi). Quelle est l'opération arithmétique réalisée ?

```
int x;
x = 0xcde12300;
x = x >> 8;
```

Que peut-on dire des valeurs relatives de x et y lorsque l'indicateur C (retenue finale) est à 1 après calcul de x-y par addition du complément à 2 de y :

- pour x et y de type int ?
- pour x et y de type unsigned int ?

2 Constructions algorithmiques (40mn)

On considère le squelette de programme C suivant :

```
#include <stdio.h>

#define N 10

int main (void)
{

  register unsigned int s;    // a stocker dans registre r7
  register unsigned int i;    // a stocker dans registre r6
  register unsigned int j;    // a stocker dans registre r5

  ... // instructions à retrouver
  printf ("%d\n",j);
  ... // instructions à retrouver

   return 0;
}
```

ainsi que sa traduction en langage d'assemblage ARM :

```
@ s : r7  i : r6  j : r5

@ Ne vous souviez pas de ce qui est avant debut:
            .global main
            .text

format:     .asciz   "%d\n"
            .balign 4
```

```
main:       stmfd  sp!,{r0-r7,lr}
            ldr    r8,= format

debut:      mov r5,#0
et1:        b    et6
et2:        mov  r7,#0
            mov  r6,#0
et3:        b     et5
et4 :       add  r7,r7,r6
            add  r6,r6,#1
et5:        cmp  r6,r5
            bls  et4

fin1:       @ debut de traduction de printf ("%d\n",j);
            mov  r0,r8
            mov  r1,r7
            bl   printf
            @ fin de traduction de l'appel de printf

debut2:     add  r5,r5,#1
et6:        cmp  r5,#10
            bls  et2

@ Ne vous souciez pas non plus de ces instructions
fin2:       ldmfd sp!,{r0-r7,lr}
            mov  r0,#0
            mov  pc,lr
```

Donner une séquence d'instructions C équivalente au code en langage d'assemblage compris entre les étiquettes debut1 et fin1 d'une part, et entre debut2 et fin2 d'autre part. Vous pouvez procéder en deux étapes : traduction intermédiaire en C avec goto, puis en C sans goto.

Ecrire en langage d'assemblage un code équivalent dans lequel les branchements conditionnels sont tous des branchements en avant.