Master CCI

Langage machine

Solution du contrôle continu écrit 2020

Durée 1h30, documents autorisés, calculatrices et ordinateurs interdits

Les durées servent de barême indicatif

Durée 1h30, documents autorisés, calculatrices et ordinateurs interdits

Les questions peuvent être traitées dans n'importe quel ordre (mais **??** et **??** portent sur la même portion de code) et sont d'importance égale.

Du point de vue timing, la partie base 2 est la plus susceptible de déraper : soyez attentif à ne pas perdre trop de temps dessus.

Table des matières

Conventions et contraintes

L'ordre dans lequel les variables sont déclarées doit être respecté dans la traduction en langage d'assemblage. Les variables déclarées sans attribut register **doivent** être stockées en mémoire (contenu mémoire mis à jour à chaque affectation).

Voici les types de variables entières utilisées dans le code des exercices. Par dérogation, considérez pour la traduction en langage d'assemblage le type char comme synonyme de uint8_t (entier naturel).

types d'entier relatif		taille	types d'entier naturel	
synonyme sur ARM	type	bits	type	synonyme sur ARM
	int8_t	8	uint8_t	unsigned char
short	int16_t	16	uint16_t	unsigned short
int	int32_t	32	uint32_t	unsigned int

Les constantes symboliques peuvent être définies par = en langage d'assemblage (TAILLE=2), après quoi elles peuvent être utilisées comme opérande dans une directive ou une instruction : (mov r0,#TAILLE).

Les directives de réservation de place acceptent des expressions simples composées d'entiers, de constantes entières symboliques et d'opérateurs arithmétiques de base : +,-,* (.skip 4*TAILLE).

1 Variables et accès mémoire (30mn)

1.1 Déclarations (10mn)

Traduire en langage d'assemblage les déclarations de variables suivantes :

```
uint8_t    x = 35;
uint8_t *pt2 = &y;
uint8_t    y;
register uint8_t *pt1;  // utiliser le registre r0
```

1.2 Accès aux variables (20mn)

Traduire en code ARM chacune des affectations suivantes :
- vous pouvez utiliser d'autres registres que r0 comme temporaires
- sans factorisation de code avec celui des autres affectations
- en supposant que d'autres instructions ont pu modifier les variables depuis leur déclaration : x et pt2 peuvent contenir autre chose que 35 et &y.

1. y=x+3;
2. pt1=&x;
3. *pt1=2;
4. x=*pt2;

```
            .data
x:          .byte   35
            .balign 4
pt2:        .word   y

            .bss
y:          .skip   1    @ .byte 0 à la rigueur acceptable

@ Pas réservation de mémoire pour pt1 qui est déclaré avec l'attribut register
@ pt1 EST le registre r1

@ r0 : pt1
@ r1: temporaire adresses
@ r5:  copie de valeur

@ y=x+3;    --> *&y = *&x
@ Mem[adresse_de_y] = Mem8[adresse_de_x] + 3
        ldr   r1,= x
        ldrb  r5, [r1]
        add   r5,r5,#3
        ldr   r1,= y
        strb  r5,[r2]

@ pt1=&x;  registre r0 <- adresse de x
```

```
        ldr   r0,=x

@ *pt1=2;  Mem8[registre r0] = 2
        mov   r5,#2
        strb  r5,[r0]

@ x=*pt2;       --> x = **&pt2   Mem8[adresse[x] <- Mem8[Mem32[adresse_de_pt2]]
        ldr   r1,=pt2
        ldr   r1,[r1]
        ldrb  r5,[r1]
        ldr   r2,=x
        strb  r5,[r2]
```

2 Constructions algorithmiques (25mn)

On considère des extraits d'un programme de conversion binaire.c (code complet détaillé en annexe), qui affiche la valeur de l'entier x en base 2. Contrairement à la méthode classique de conversion le programme détermine et affiche les chiffres binaires directement dans le bon ordre (chiffre des unités en dernier, affiché à droite).

Les délarations de variables dans main sont rappelées ci-dessous :

```
register int32_t val;      // utiliser r4
register char car;         // utiliser r5
register unsigned int c;   // utiliser r6
register uint32_t borne;   // utiliser r7
```

2.1 Traduction d'un if (10 mn)

Traduire *en code ARM la construction if de la deuxième boucle :*

```
if (val <0) {
   c=1;
} else {
   c=0;
}
```

```
if2:       cmp  r4,#0            @ if (val >= 0) goto endif2
           bge  else2
then2:     mov  r6,#1            @ c=1
           b    endif2           @ goto endif2
else2:     mov  r6,#0
endif2:
```

2.2 Traduction d'une boucle (15mn)

Traduire *en code ARM la deuxième boucle. L'ensemble du if et l'appel de printf seront remplacés chacun par un simple commentaire.*

```
while (borne > 0) {
   // code du if
   car = 0x30+c;
   val = val*2;
   borne = 2*borne;
   // appel de printf
}
```

```
@ Version avec condition après le corps

           b condw2
body2:
if2:
           @ insert if code here
endif2:
           add  r5,r6,#0x30   @ car=0x30+c   ou car='0'+c
           mov  r4,r4,LSL #1  @ val = val*2    (add r4,r4,r4 possible également)
           mov  r7,r7,LSL #1  @ borne = borne*2
           @ add call to printf here
endbody2:
condw2:    cmp r7,#0          @ if (borne > 0) goto body2
           bhi body2

@  version avec condition avant le corps

condw2:    cmp r7,#0          @ if (borne <= 0) goto endw2
           bls endw2
body2:     ...                @ same body
endbody2:  b   condw2         @ goto condw2
```

3 Représentation d'informations en binaire (35mn)

3.1 Fonctionnement du programme de conversion (section ??) (15mn)

Expliquer *brièvement*

1. *pourquoi la valeur du chiffre courant est déterminée par la condition val<0 et pourquoi val est multiplié et non divisé par 2.*
2. *le critère d'arrêt de la deuxième boucle (section* **??***) et préciser si l'attribut unsigned (uint32_t et non int32_t) de la variable borne est important ou non*
3. *la raison de l'ajout de la constante 0x30*

1. *Le principe consiste à extraire à chaque tour de boucle le bit de poids fort (à écrire en premier), qui donne le signe de l'entier relatif (*< 0 *si le bit est à 1). Une fois le bit affiché, la multiplication par 2 décale l'entier d'un bit à gauche pour que le bit suivant à afficher devienne à son tour bit de signe.*

2. *Soit i le numéro du bit de x en cours d'affichage, on maintient l'invariant borne $= 2^{31-i}$, d'où borne=1 pour le premier affichage. La multiplication par 2 décale d'un bit à gauche val (i décroît) et borne, ce qui maintient l'invariant. Au dernier bit, borne $= 2^{31}$ et retombera à 0 lors de la multiplication, arrêtant la boucle.*
   Si borne était déclarée comme un relatif, le bit de poids 31 à 1 de borne ferait interpréter le contenu de borne comme < 0 et la sortie de boucle se produirait un bit trop tôt.
3. *La constante 0x30 est '0', le code ASCII du caractère zéro, le code ASCII des chiffres décimaux n'étant pas égal à leur valeur entière.*

3.2 Addition binaire et indicateurs (20mn)

Compléter *l'addition sur 4 bits ci-dessous en détaillant les retenues*
Que valent *les indicateurs Z,N,C et V (préciser brièvement la méthode de lecture de C et V)*
D'après les indicateurs, le **résultat** *apparent est-il* **correct** *?*

Ecrire *en binaire le complément à 2 du premier opérande. Quel est la signification arithmétique du complément à 2 ?*

	1	0	1	1
+	0	1	1	0

---

	x_3=1	0	1	1
+	y_3=0	1	1	0
$C=c_4$=1	c_3=1	1	0	0
	$N=res_3$=0	0	0	1

C est la dernière retenue (c_4). C vaut 1 : le résultat apparent est incorrect si l'opération représente l'addition d'entiers naturels. Le résultat apparent est le vrai résultat (11+6=17) modulo 2^4, soit 1.

N est le bit de poids fort du résultat apparent : $N=res_3=0$. S'il s'agit d'une opération sur les entiers relatifs, celà indique un résultat apparent ≥ 0.

V est l'indicateur de débordement en artihmétique sur les entiers relatifs. Dans cette opération, V=0 : indique un résultat correct (-5 + +6 = +1).

V=0 au choix :
- *parce que les opérandes sont de signes différents ($x_3 \neq y_3$) : la valeur absolue du résultat ne peut excéder celle des opérandes et il est donc représentable.*
  La formule $\overline{x_3}.\overline{y_3}.res_3 + x_3.y_3.\overline{res_3}$ donne V=0.
- *parce que les deux dernières retenues sont identiques : $c_4 = c_3$.*

Z=0 parce qu'au moins un bit (res_0) du résultat est à 1 : le résultat apparent est non nul, quelle que soit l'interprétation (addition d'entier naturels ou d'entiers relatifs)

Le complément à 2 d'un entier relatif représente l'opposé pour un entier relatif (ou 2^n - l'entier naturel). Le complément à 2 de **101***1 est* **010***1 (bit de droite conservés jusqu'au premier 1 inclus). Autre méthode : C2(x) =* $2^n - x$ *: Ici* 1011_2 *représente* 11_{10} *son complément s'écrit comme l'entier naturel* $2^4 - 11 = 5$.

4 Annexe : source du programme de conversion en binaire

Pour donner un exemple d'exécution sans utiliser scanf, la variable x est initialisée avec une constante. La connaissance de cette constante ne peut évidement pas être utilisée pour simplifier le code de main.

```
int32_t x=0x39ab5675;

int main () {
   register int32_t val;      // utiliser r4
   register char car;         // utiliser r5
   register unsigned int c;   // utiliser r6
   register uint32_t borne;   // utiliser r7

   val = x;
   borne=1;

  if (val==0) {
     printf ("0");
  } else {

       while (borne >0 && val >0) {
         val = 2*val;
         borne = 2*borne;
       }

       while (borne > 0) {
         if (val <0) {
           c=1;
         } else {
           c=0;
         }
         car = 0x30+c;
         val = 2*val;
         borne = 2*borne;
         printf ("%c",car);
       }       // fin du while
   }           // fin du else
   printf ("\n");
```

```
   return 0;
}

mandelbrot> ./binaire
11100110101011010101100111010 1
mandelbrot>
```