Master CCI

Langage machine

Contrôle continu écrit 2015

Durée 1h30, documents autorisés, calculatrices et ordinateurs interdits

Table des matières

Attention : ne perdez pas de temps à analyser le principe des algorithmes et focalisez-vous sur leur traduction en langage d'assemblage ARM.

1 Variables et constructions algorithmiques (45mn)

On considère un extrait de code f et sa traduction en langage d'assemblage ARM. Les fichiers f.c et f.s sont tous les deux incomplets. L'objectif est de les reconstituer en entier, chaque partie de code manquant dans l'un étant présente dans l'autre.

Utiliser les registres r0 et r1 pour les variables x et y, et les registres r2 à r4 comme stockage temporaire.

```
/* fichier f.c */
...  a=...;              /* Retrouver à quoi correspondent les ... */
...  b=...;
char c;
...  res;

void f (void)
{
  register ... x;    /* a stocker dans le registre r0 */
  register ... y;    /* a stocker dans le registre r1 */

  x=a;
  y=b;
```

```
  ...
    if (x<y) {
       x=x+a;
    } else {
       y=y+b;
    }
  ...
 res = x;
}

/* fichier f.s */
                    .data
a:                  ....    15     /* retrouver quelle directive de réservation utilis
b:                  ....    12

                    .bss
/* Ajouter ici la déclaration de la variable c */
res:                .skip    2


                    .text
                    @ r0 : x   r1 : y   r2, r3, r4 : temporaires

f:                  stmfd  sp!,{r0-r4}  /* debut standard de fonction */

                    ldr   r3,=a
                    ldrh  r0,[r3]

                    ldr   r4,=b
                    ldrh  r1,[r4]

etq1:               cmp   r0,r1
                    beq   etq2

@ Debut de la traduction de if omise
                    ...
@ Fin   de la traduction de if omise

                    b     etq1
etq2:               ...                @ res = x

                    ldmfd sp!,{r0-r4}   /* fin standard de fonction */
                    mov   pc,lr
```

Déduire (expliquer brièvement le raisonnement) du fichier.s **de quel type**(parmi les 6 du tableau 3) sont les **variables** a,b,res,x et y (type commun aux cinq variables). **Compléter** leurs déclarations dans f.c et f.s. **Traduire** ensuite la déclaration de la

variable c dans la section bss.

Traduire en langage d'assemblage l'instruction res = x.

Traduire en langage d'assemblage la construction algorithmique if.

Retrouver la partie de code de f.c qui englobe la construction if. L'objectif est de reconstituer un code C standard dépourvu d'instruction goto.

2 Mémoire, base 2 et boucle (45mn)

2.1 Boucle et mémoire(35mn)

On considère la fonction calcul suivante et sa traduction partielle (sans la gestion de la boucle) :

```
unsigned int    x = 0x12345678;

unsigned int calcul (void)
{
  register int i;
  register unsigned int valeur;
  register unsigned int lu;
  register unsigned char *pc;

/* (unsigned char *) ne spécifie aucun accès mémoire                */
/* C'est un forceur de type pour dire au compilateur de considérer  */
/* &x comme si x avait été déclarée de type unsigned char           */

  pc = (unsigned char *) &x;
  valeur = 0;

  i = 3;
  while (i>=0) {
   lu = *(pc+i);

/* valeur << 8 : prendre une copie du contenu de valeur et le décaler */
/* de 8 bits vers la gauche */
    valeur = lu + (valeur << 8);

    i--;
  }
  return valeur;
}

                .data
```

```
x:              .word    0x12345678

@ r1 : pc   r2: valeur   r3 : lu  r4 : i
@ valeur de retour de calcul dans r0

calcul:         stmfd    sp!, {r1,r4}

                ldr      r1,=x              @ pc = &x
                mov      r2, #0             @valeur = 0

                mov      r4,#3              @ i = 3

                ...                         @ à compléter

                ldrb     r3,[r1,r4]
                ...                         @ à compléter
                sub      r4,r4,#1

                ...                         @ à compléter

                mov      r0,r2              @ return valeur
                ldmfd    sp!, {r1,r4}
                mov      pc,lr

                .ltorg
```

Compléter la traduction de calcul en langage d'assemblage

La valeur retournée par la fonction dans r0 est celle de x : 0x12345678. En **déduire** (avec explications) si la machine est de type big ou little endian (cf définition en annexe).

Proposer une séquence d'instructions ARM n'utilisant aucune variante de ldr et qui charge la constante 0xabcdef1234 dans le registre r0.

Que peut-on affirmer à propos du saut en fin de calcul (expliquer brièvement vos choix) :

1. (a) conditionnel
   (b) inconditionnel
   (c) propriété impossible à déterminer
2. (a) absolu
   (b) relatif
   (c) propriété impossible à déterminer
3. (a) saut en avant
   (b) saut en arrière
   (c) propriété impossible à déterminer

2.2 Base 2 (10mn)

Soient r0 et r2 deux variables de type entier naturel et la séquence de code suivante :

```
mvn     r1,r0
add     r1,r1,#1
add     r2,r0,r1
```

Donner la valeur de r2 en fin de séquence pour les valeurs suivantes (écrites en base 10) de r0 : 10, 260, 1040 et 1054632.

3 Annexe 1 : tableau des types d'entiers de stdint.h

Après utilisation de la directive #**include** <**stdint.h**>, on peut utiliser les types C d'entiers de taille certaine intxx_t et uintxx_t regroupés dans le tableau ci-dessous, avec leurs synonymes classiques [1] pour un processeur ARM 32 bits.

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

4 Annexe 2 : conventions little et big endian

Un entier E stocké sur plus de 8 bits en mémoire occupe des octets d'adresses consécutives (A,A+1,A+2 et A+3 pour un entier codé sur 32 bits). L'adresse A de l'entier est celle de son premier octet.

La représentation de l'entier en binaire est donc découpée en paquets de 8 bits stockés chacun dans un octet de mémoire selon un ordre croissant ou décroissant.

Dans la convention Big Endian, l'ordre est décroissant : le premier octet (A) contient les bits de poids forts de l'entier et le dernier octet (A+3) les bits de poids faibles de E.

La convention opposée est dite Little Endian : les bits de poids faibles occupe le premier octet (A) et ceux de poids forts le dernier (A+3).

1. dont la taille effective peut varier d'une machine à l'autre