Master CCI

Langage machine

Contrôle continu écrit 2014

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 Modulo 3 : constructions algorithmiques (45mn)

On veut calculer le reste de la division par 3 du contenu de la variable entière x, en n'utilisant ni x/3, ni x%3, mais des soustractions.

Traduire en langage d'assemblage ARM les déclarations de variables ainsi que les instructions entre les appels de sscanf et printf (exclus) avec les contraintes suivantes :

1. Bien que l'instruction ARM de division (div) existe mais vous devez essayer de ne pas l'utiliser.
2. L'ordre des déclarations de variables stockées en mémoire doit être respecté.

```
unsigned short resultat;
unsigned int x;

int main (int argc, char *argv[])
{
   register unsigned int delta;          // a stocker dans r1
   register unsigned int etapes;         // a stocker dans r3
   register unsigned short int mod3;     // a stocker dans r2

   sscanf (argv[1]," %x",&x);
   // Traduire en ARM depuis ici ...

   delta = 0x30000000;
```

```
   mod3=x;
   etapes=0;

   while (mod3>=3)
     {
     if (mod3>=delta)
       mod3 = mod3 - delta;
     else
       delta = delta/2;
     etapes++;
     }
   resultat = mod3;
  // ... jusqu'ici

   printf ("mod3=%d calcul en %u etapes\n",mod3,etapes);
   return 0;
}
```

2 Modulo 3 : variables et pointeurs (20mn)

Voici un autre algorithme (très inefficace) de calcul de modulo 3, basé sur une liste circulaire de structures chaînées entre elles : le champ suivant de c_i repère la structure $c_{(i+1)\%3}$.

```
unsigned int x;
unsigned short mod3;

struct cellule {
  unsigned short  mod;
  struct cellule *suivant;
};

extern struct cellule c0;   // avec extern les declarations ne définissent
extern struct cellule c1;   // que le type des structures et ne reservent
extern struct cellule c2;   // pas de memoire de stockage

struct cellule c0 = {0,&c1};  // vraies declaration avec reservation
struct cellule c1 = {1,&c2};  // de mémoire et valeurs intiales
struct cellule c2 = {2,&c0};


int main (int argc, char *argv[])
{
   register struct cellule *p;   // un pointeur de ce genre de structure

   sscanf (argv[1]," %x",&x);
   printf ("x=0x%8x (%11d)i, mod3(x) = %u\n",x,x,x%3);
```

```
   p=&c0;
   while (x>0)
     {
     p = (*p).suivant;
     x--;
     }
   mod3 = (*p).mod;
   printf ("%u\n",mod3);

   return 0;
}
```

Traduire en langage d'assemblage ARM

1. les réservations de place associées aux déclaration des variables x,mod, c0 à c2.
2. l'instruction p=&c0
3. l'instruction p = (*p).suivant
4. l'instruction mod3 = (*p).mod

3 Branchements, adresses et base 2, divers (25mn (sans le bonus))

Un programme C gère trois variables entières sur 32 bits : a, b et niter. Ce programme contient deux constructeurs algorithmiques classiques imbriqués. Voici un squelette de sa traduction en langage d'assemblage ARM :

```
@ Associations variables <-> registres : a <-> r0, b <-> r1, niter <-> r2
@ En cas de besoin de temporaire, utiliser r3

main:       ...                 @ code d'initialisation de a et b omis
            mov   r3,#0
            b     etiq4         @ b est synonyme de bal (branch always)
etiq1:      blo   etiq2
            sub   r1,r1,r2      /*   A    */
            b     etiq3         /*   B    */
etiq2:      sub   r2,r2,r1      /*   C    */
etiq3:      add   r3,r3,#1      /*   D    */

etiq4:      cmp   r1,r2
            bne   etiq1

            mov   r0,#0         @ ces 2 isntructions mov sont la traduction
            mov   pc,lr         @ de return 0 en fin de main
```

3.1 Traduction inverse de constructeurs algorithmiques

(Question bonus) Quelle(s) séquence(s) de 2 instructions sera ou seront exécutée(s) avant l'instruction blo ? Par quelle instruction seront positionnés les indicateurs Z,N,C,V utilisés dans l'évaluation de la condition lower ?

(Question bonus) Donner une suite d'instructions C équivalente à ce programme, (d'abord avec, puis sans if ... goto) [1]

3.2 Notion de branchement et base 2

Quelle est l'instruction de branchement conditionnel à utiliser pour que la condition de saut soit l'inverse de celle testée par blo ?

Ecrire en C la déclaration des variables a et b.

Supposons que l'instruction blo etiq2 soit stockée à l'adresse 0x00100010. **A quelle adresse** hexadécimale correspond l'étiquette etiq2 ?

L'instruction blo est un branchement relatif. Son paramètre etiq2 sera traduit en un entier relatif Delta inclus dans l'intruction blo. Delta, encodé sur 24 bits suivant la méthode classique du complement à 2, est exprimé en nombre d'instructions dont il faut se déplacer.

Calculer Delta.

De combien (environ) d'instructions au maximum pourrait-on se déplacer vers l'avant avec une instruction telle que b ou blo ?

Considérez la proposition suivante : on peut substituer à l'instruction b etiq3 une pseudo-instruction ldr (forme ,=) sans modifier le comportement du programme. ?

— Si elle vraie, **écrire** cette pseudo-instruction (avec une courte explication du pourquoi)
— Sinon expliquez **pourquoi c'est impossible**

Il manque une directive de section à ce squelette de traduction en langage d'assemblage ARM. **Donner cette directive**

Quelle est la différence de comportement entre les instructions add et addS ?

Par quelle instruction [2] pourrait-on remplacer l'instruction cmp sans changer le comportement du programme ?

1. Toute ressemblance avec un algorithme de calcul connu n'est absolument pas fortuite.
2. en langage d'assemblage ARM