M2P CCI : Corrigé Langage Machine, Novembre 2007

1 Déclarations des variables

Les variables initialisées sont déclarées dans data, les autres dans bss. Un short occupe deux octets, un long en occupe 4.

```
   LONGUEUR_MAX = 51

     .global v2
     .global somme
     .global valeurs

         .bss
         .balign 2    @ facultatif : premier objet déclaré dans bss
v2:      .skip  2
valeurs: .skip 2*LONGUEUR_MAX

         .data
somme:   .word 0
```

La convention d'appel stipule que y est passé dans r1. On suppose que r est stocké dans le registre r5.

```
     @ r0 : x, r1 : y, r5 : r

     strh  r5, [r1]   @ *y = r
```

2 Base 2 et asm vers C

```
Sur 32 bits :
0x0000000a & 0x00000001  donne 0x00000000
0x0000000b & 0x00000001  donne 0x00000001
0x0000000c & 0x00000001  donne 0x00000000
0x0000000d & 0x00000001  donne 0x00000001

  1010    1011     1100    1101
& 0001  & 0001   & 0001  & 0001
  ----    ----     ----    ----
  0000    0001     0000    0001
```

La propriété commune de tous les entiers tels que le Et bit à bit avec l'entier 1 est non nul est d'avoir 1 en bit de poids faible. Autrement dit, il s'agit de tous les entiers

impairs (entier modulo 2 non nul).

Le décalage d'un bit à droite (respectivement à gauche) d'un entier naturel est une opération de division entière (respectivement multiplication) par deux.

Les deux premières instructions effectuent un branchement à pas_nul si l'entier est impair. L'instruction mov effectue une division par deux et l'addition une multiplication par trois.

Voici un code if ... goto équivalent :

```
          if  ((x%2) != 0) goto pas_nul;       @ andS
nul:      x = x >> 1;                          @ mov : calcule /2
          goto sortie;
pas_nul:  x = x + x << 1 ;                     @ add : calcule *3
          x = x+1;                             @ add
sortie:
```

On reconnait une structure classique si ... alors ... sinon et la définition d'une suite connue : la suite de Syracuse.

```
si x impair                  @ en C :    if ((x%2) != 0)
   alors
      x = 3x+1                                  x = 3*x+1;
   sinon                                     else
      x = x/2                                   x = x/2;
```

3 Traduction de la boucle

Traduction de la boucle : code en if ... goto.

D'après la convention d'appel + le paramètre est stocké dans un registre : r0 + les variables locales sont stockées dans des registres

Variante avec un autre type de convention : voir à la fin

```
       pval = valeurs; v1 = initial;
       goto test
corps: di2_p__mult3pl1_imp (v1,&v2);
       *pval = *&v2;
       *&somme += *&v2;
       pval++;
       v1 = *&v2;
test:  if (pval >= (valeurs + LONGUEUR_MAX)) goto fin;
       if (v1 != 1) goto corps;
```

Traduction en langage d'assemblage.

```
            @ r0 : initial   r5 : pval  r6 : v1  r7,r8,r9 : temporaires
                 .global calculer

                 .text
calculer:        @ sauvegardes de r5 à r9 + lr omises
                 ldr  r5,= valeurs         @ pval = valeurs
                 mov  r6, r0               @ v1 = initial

                 @ pas besoin de sauvegarder r0 parce que
 @ initial n'est pas utilise dans la suite
corps:           mov  r0, r6               @ x = v1
                 ldr  r1,= v2              @ y = &v2
                 bl   di2_p__mu3p1_imp     $ di2_p__mu3p1_imp(v1,&v2)
                 ldr  r9,= v2              @ r9 = & v2
                 ldrh r9, [r9]             @ r9 = *&v2
                 strh r9, [r5]             @ *pval = *&v2

                 ldr  r7,= somme           @ r7 = &somme
                 ldr  r8, [r7]             @ r8 = *&somme
                          @ *&v2 deja dans r9
                 add  r8, r8, r9           @ r8 += *&v2
                 str  r8, [r7]             @ *&somme += *&v2

                 add  r5, r5, #2           @ pval++

                 ldr  r7,= v2              @ r7 = &v2
                 ldrh r6, [r7]             @ v1 = *&v2

test:            ldr  r7,= valeurs           @ r7 = valeurs
                 add  r7, r7, #2*LONGUEUR_MAX @ r7 = valeurs+ LONGUEUR_MAX
                 cmp  r5, r7
                 bhs  fin                  @ if ( ... >= ... ) goto fin
                 cmp  r6, #1
                 bne  corps                @ if (v1 != 1) goto corps

fin:             @ restauration de r5 à r9 + lr omise
                 mov  pc,lr

                 .ltorg
```

Ecriture de la boucle avec une variable de type indice.

```
int i;

for (i=0; i < LONGUEUR_MAX; i++)
   {
```

```
      di2__mult3p11_imp (v1, &v2);
      valeurs[i] = v2;
      somme += v2;
      v1=v2;
      }
```

En supposant maintenant une convention dans laquelle les variables locales sont stockées dans un bloc mémoire associé à la procédure, le code doit ressembler en gros à ceci :

```
       *&pval = valeurs; *&v1 = initial;
       goto test
corps: di2_p__mult3pl1_imp (*&v1,&v2);
       **&pval = *&v2;
       *&somme += *&v2;
       (*&pval) ++;
       *&v1 = *&v2;
test:  if (*&pval >= (valeurs + LONGUEUR_MAX)) goto fin;
       if (*&v1 != 1) goto corps;
```

```
            @ r0 : initial   r5 : *&pval  r6 : *&v1  r7,r8,r9 : temporaires
                 .global calculer

                 .bss
privee_f:        .skip 2     @ variable locale pval
                 .skip 2     @ variable locale v1
 .skip 24    @sauvegarde registres r5 a r9 + lr
param_f:         @ neant
                 @ le premier parametre est dans le registre r0

                 DELTA_CALCULER_PVAl = -30
                 DELTA_CALCULER_V1   = -28

                 .text
calculer:        @ sauvegardes de r5 à r9 + lr omis
                 ldr ip,=param_calculer

                 ldr  r5,= valeurs         @ *&pval = valeurs
                 str  r5, [ip, ##DELTA_CALCULER_PVAl]

                 mov  r6, r0               @ *&v1 = initial
                 strh  r6, [ip, ##DELTA_CALCULER_V1]

corps:           ldr  r0, [ip, ##DELTA_CALCULER_V1]   @ x = *&v1
                 ldr  r1,= v2                         @ y = &v2
                 bl   di2_p__mu3p1_imp     $ di2_p__mu3p1_imp(v1,&v2)
```

```
                ldr  r9,= v2              @ r9 = & v2
                ldrh r9, [r9]             @ r9 = *&v2
                ldr  r5, [ip, ##DELTA_CALCULER_PVAl] @ r5 = *&pval
                strh r9, [r5]             @ **&pval = *&v2

                ldr  r7,= somme           @ r7 = &somme
                ldr  r8, [r7]             @ r8 = *&somme
                                @ *&v2 deja dans r9
                add  r8, r8, r9           @ r8 += *&v2
                str  r8, [r7]             @ *&somme += *&v2

                ldr  r5, [ip, ##DELTA_CALCULER_PVAl] @ (*&pval) ++
                add  r5, r5, #2
                str  r5, [ip, ##DELTA_CALCULER_PVAl] @

                ldr  r7,= v2                        @ r7 = &v2
                ldrh r6, [r7]                       @ r6 = *&v2
                strh r6, [ip, ##DELTA_CALCULER_V1] @ *&v1 = *&v2

test:           ldr  r7,= valeurs           @ r7 = valeurs
                add  r7, r7, #2*LONGUEUR_MAX @ r7 = valeurs+ LONGUEUR_MAX
                ldr  r5, [ip, ##DELTA_CALCULER_PVAl] @  r5 = *&pval
                cmp  r5, r7
                bhs  fin                  @ if ( ... >= ... ) goto fin
                cmp  r6, #1
                bne  corps                @ if (v1 != 1) goto corps

fin:            @ restauration de r5 à r9 + lr omise
                mov  pc,lr

                .ltorg
```