Université Laval
Faculté des sciences et de génie
Département d'informatique et de génie logiciel
IFT-3101

Danny Dubé
Version : Automne 2013

Exercices reliés au chapitre 8

Exercices

Voici les exercices que je recommande de faire :

— **Exercice 8.2.1**. (Dans la 1ère édition, les exercices 9.1, 9.2 sont similaires.)
Note : Supposez que les variables sont aux positions suivantes en mémoire :

Variable	**Adresse**
x	128
y	132
a	104
b	108
c	112

— **Exercices 8.2.2**, **8.2.3**, **8.2.4** et **8.2.5**.

— **Exercice 8.3.1**.

— **Exercice 8.3.2**.
Note : Supposez que les variables sont aux positions suivantes par rapport au pointeur de pile `SP` *:*

Variable	**Adresse**
x	28
y	24
a	4
b	8
c	12

— **Exercice 8.4.1**
Note : le manuel mentionne que la matrice contient des éléments de 8 octets, mais cette information nous sera de peu d'utilité car la dimension de la matrice (n) n'est pas connue.

— **Exercice supplémentaire 1.** Générer le code pour la machine de la section 8.3 (9.2 dans la 1ère édition) pour le programme suivant :

```
/* Code du programme principal */
call f
call g
halt

/* Code de f */
x ++
return

/* Code de g */
x --
call f
return
```

Réponses

Notez que pour améliorer la lisibilité, nous utiliserons 'mov' à la place de 'ld' afin d'être plus près de la syntaxe d'assembleur la plus connue. De façon générale, notez que le but de l'exercice est de comprendre la logique et pas de respecter à la lettre la syntaxe du langage machine imaginaire qui est sommairement présenté dans le manuel.

Exercice 8.2.1

— (a)

```
MOV  128, #1
```

— (b)

```
MOV  128, 104
```

— (c)

```
MOV  R1, 104
ADD  R1, #1
MOV  128, R1
```

— (d)

```
MOV  R1, 104
ADD  R1, 108
MOV  128, R1
```

— (e)

```
MOV  R1, 108
MUL  R1, 112
MOV  128, R1
ADD  R1, 104
MOV  132, R1
```

Exercice 8.2.2

— (a)

```
MOV  R1,    i
MUL  R1,    #4
MOV  x,     a(R1)
MOV  R2,    j
MUL  R2,    #4
MOV  y,     b(R2)
MOV  a(R1), y
MOV  b(R2), x
```

— (b)

```
MOV  R1, i
MUL  R1, #4
MOV  x,  a(R1)
MOV  R1, i      ⟵ Éliminable !
MUL  R1, #4     ⟵ Éliminable !
MOV  y,  b(R1)
MOV  R1, x
MOV  R2, y
MUL  R1, R2
MOV  z,  R1
```

— (c)

```
MOV  R1,    i
MUL  R1,    #4
MOV  x,     a(R1)
MOV  R2,    j
MUL  R2,    #4
MOV  y,     b(R2)
MOV  a(R1), y
```

Exercice 8.2.3

```
MOV  R1,     &q
MOV  y,      0(R1)
ADD  q,      #4
MOV  R1,     &p
MOV  0(R1),  y
ADD  p,      #4
```

Exercice 8.2.4

```
     MOV   R1,  x
     SUB   R1,  y
     BLTZ  R1,  L1
     MOV   z,   #0
     BR    L2
L1 : MOV   z,   #1
```

Exercice 8.2.5

```
     MOV   s,   #0
     MOV   i,   #0
L1 : MOV   R1,  i
     SUB   R1,  n
     BGTZ  R1,  L2
     ADD   s,   i
     ADD   i,   #1
     BR    L1
L2 :
```

Exercice 8.3.1

```
ADD  SP,          SP,          #caller.recordSize
MOV  0(SP),       #here + 16
BR   procP.codeArea
SUB  SP,          SP,          #caller.recordSize
ADD  SP,          SP,          #caller.recordSize
MOV  0(SP),       #here + 16
BR   procQ.codeArea
SUB  SP,          SP,          #caller.recordSize
BR   0(SP)
ADD  SP,          SP,          #caller.recordSize
MOV  0(SP),       #here + 16
BR   procR.codeArea
SUB  SP,          SP,          #caller.recordSize
BR   0(SP)
BR   0(SP)
```

Exercice 8.3.2

— (a)

```
MOV  28(SP),  #1
```

— (b)

```
MOV  R1,      4(SP)
MOV  28(SP),  R1
```

— (c)

```
MOV  R1,      4(SP)
ADD  R1,      #1
MOV  28(SP),  R1
```

— (d)

```
MOV  R1,      4(SP)
ADD  R1,      8(SP)
MOV  28(SP),  R1
```

— (e)

```
MOV  R1,      8(SP)
MUL  R1,      12(SP)
MOV  28(SP),  R1
MOV  R1,      4(SP)
ADD  R1,      28(SP)
MOV  24(SP),  R1
```

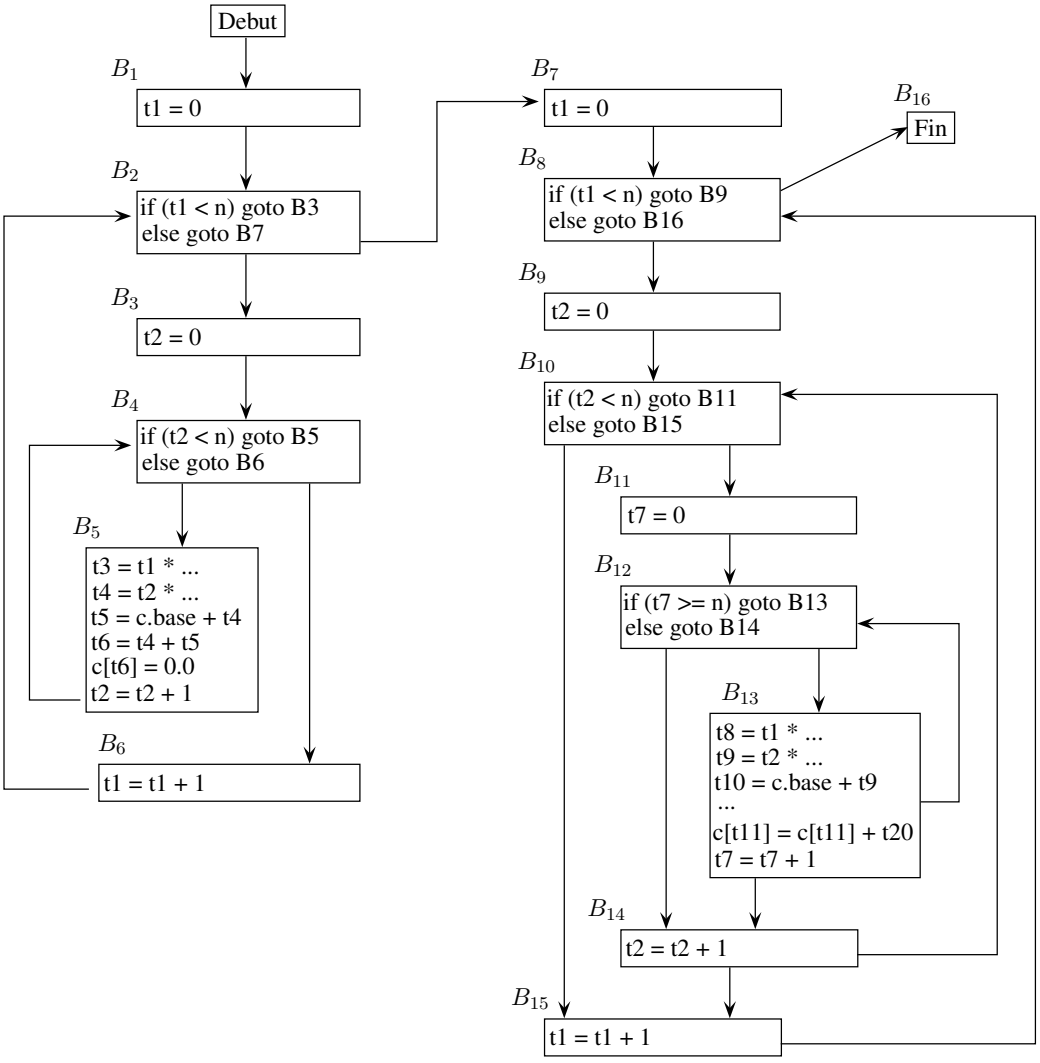
8.4.1

— (a) Génération de code :

```
t1 = 0

// For i ...
goto lcheck_i_loop
lstart_i_loop:

    // For j...
    t2 = 0
    goto lcheck_j_loop
    lstart_j_loop:
    // c[i][j] = 0.0
    t3 = t1 * c.type.elem.width
    t4 = t2 * c.type.elem.elem.width
    t5 = c.base + t4
    t6 = t5 + t3
    c[t6] = 0.0

    // j++
    t2 = t2 + 1
    lcheck_j_loop:
    if (t2 < n) goto lstart_j_loop

// i++
t1 = t1 + 1
lcheck_i_loop:
if (t1 < n) goto lstart_i_loop
```

```
t1 = 0
goto lcheck_i_loop
lstart_i_loop:
    t2 = 0
    goto lcheck_j_loop
    lstart_j_loop:
        t7 = 0
        goto lcheck_k_loop
        lstart_k_loop:
            // c[i][j]
            t8 = t1 * c.type.elem.width
            t9 = t2 * c.type.elem.elem.width
            t10 = c.base + t9
            t11 = t10 + t8

            // a[i][k]
            t12 = t1 * a.type.elem.width
            t13 = t7 * a.type.elem.elem.width
            t14 = a.base + t12
            t15 = t14 + t12

            // b[k][j]
            t16 = t7 * b.type.elem.width
            t17 = t2 * b.type.elem.elem.width
            t18 = b.base + t16
            t19 = t18 + t17

            // a[i][k] * b[k][j]
            t20 = a[t15] * b[t19]

            // c[i][j] = c[i][j] + a[i][k]*b[k][j]
            c[t11] = c[t11] + t20

            // k++
            t7 = t7 + 1
            lcheck_k_loop:
            if (t7 < n) goto lstart_k_loop

        // j++
        t2 = t2 + 1
        lcheck_j_loop:
        if (t2 < n) goto lstart_j_loop

    // i++
    t1 = t1 + 1
    lcheck_i_loop:
    if (t1 < n) goto lstart_i_loop
```

— (b) Diagramme de flot de contrôle



Notez que selon la technique utilisée pour générer le code des boucles, le résultat peut varier quelque peu.

— (c) Boucles
  — B_4, B_5
  — B_2, B_3, B_4, B_5, B_6
  — B_{12}, B_{13}
  — B_{10}, B_{11}, B_{12}, B_{13}, B_{14}
  — B_8, B_9, B_{10}, B_{11}, B_{12}, B_{13}, B_{14}, B_{15}

Exercice supplémentaire 1

Supposons que la fonction principale s'appelle 'main' ; que `#StackStart` représente l'adresse où la pile commence ; que `#p.length` donne la taille sur la pile de la procédure `p` ; et que `&L` retourne l'adresse de l'étiquette 'L'.

```
MOV SP, #StackStart      // Initialiser le pointeur de pile

// call f
ADD SP, #main.length     // Déplacer le pointeur de pile après le bloc de 'main'
MOV *SP, &Lretourmain    // Empiler l'adresse de retour
BR  Lf                   // Aller au début de la procédure 'f'
Lretourmain:
SUB  SP, #main.length    // Replacer le pointeur de pile au début du bloc de 'main'

// call g
ADD SP, #main.length     // Déplacer le pointeur de pile après le bloc de 'main'
MOV *SP, &Lretourmain2   // Empiler l'adresse de retour
BR  Lg                   // Aller au début de la procédure 'g'
Lretourmain2:
SUB SP, #main.length     // Replacer le pointeur de pile au début du bloc de 'main'

halt

// Corps de 'f'
Lf:
ADD x, #1                // x++
BR  *0(SP)               // Lire l'adresse de retour sur la pile

// Corps de 'g'
Lg:
SUB  x, #1               // x--

// call f
ADD SP, #g.length        // Déplacer le pointeur de pile après le bloc de 'g'
MOV *SP, &Lretourg       // Empiler l'adresse de retour
BR  Lf                   // Aller au début de la procédure 'f'
Lretourg:
SUB SP, #g.length        // Replacer le pointeur de pile au début du bloc de 'g'

// return
BR  *0(SP)               // Lire l'adresse de retour sur la pile
```