Examen final d'assembleur–compilation

ÉNSIIE, semestre 3

vendredi 20 décembre 2019

Durée : 3h.
Tout document personnel autorisé (pas de prêt entre voisins).
Ce sujet comporte 7 exercices indépendants, qui peuvent être traités dans l'ordre voulu.
Il contient 4 pages.
Le barème est donné à titre indicatif, il est susceptible d'être modifié. Le total est sur 20 points.
Il va de soi que toute réponse devra être justifiée.

Exercice 1 : Assembleur MIPS (3 points)

1. Question de cours : quelles sont les différences entre langage assembleur et code machine ?
2. Écrire une suite d'instructions MIPS qui mettent dans le registre `$v0` le résultat du calcul du contenu de `$a0` à la puissance `$a1`. (Il faut donc multiplier `$a0` avec lui-même `$a1` fois. On pourra supposer que `$a1` est positif ou nul.)
3. Transformer la suite précédente en fonction avec label d'entrée `puiss` et retour à l'appelant.
4. Écrire une suite d'instructions qui appellent la fonction `puiss` pour mettre dans `$v0` le résultat de $2^3 + 3^2$.
5. Comment faire pour pouvoir retrouver la valeur initiale de `$ra` après les appels à `puiss` ?

Exercice 2 : Syntaxe (2 points)

Donner l'arbre de syntaxe abstraite des instructions Pseudo Pascal suivantes, ou expliquer pourquoi elles ne sont pas syntaxiquement correctes :

1. `t[i+1] := 2`
2. `t[t[i]] := t[i]`
3. `t[i := 1]`
4. `if c < 0 and y > 3 then begin x := 1 end else y := x`
5. `if c < 0 and y > 3 then begin x := 1 else y := x end`

Exercice 3 : Analyse syntaxique (2,5 points)

On considère la grammaire suivante :

$$\begin{aligned} I &\to bLe \\ &\mid a \\ L &\to IvL \\ &\mid I \end{aligned}$$

1. Construire l'automate déterministe LR(0) pour cette grammaire.
2. Cette grammaire est-elle LR(0) ?

Exercice 4 : Réécriture (2,5 points)

On considère trois opérateurs $\texttt{neu}, \texttt{inv}, \texttt{add}$ attendant respectivement 0, 1 et 2 arguments, ainsi que le système de réécriture suivant :

$$\begin{aligned} \texttt{inv}(\texttt{inv}(X)) &\to X \\ \texttt{add}(X, \texttt{inv}(X)) &\to \texttt{neu} \\ \texttt{add}(X, \texttt{neu}) &\to X \end{aligned}$$

1. Donner la ou les forme(s) normale(s) $\texttt{add}(\texttt{inv}(\texttt{inv}(x)), \texttt{add}(y, \texttt{inv}(y)))$.
2. Le système de réécriture est-il fortement terminant ?
3. Calculer les paires critiques du système. Lesquelles sont-elles joignables ?
4. Orienter les paires critiques non joignables de façon à ce que le système soit fortement terminant. Les nouvelles paires critiques sont-elles joignables ?
5. Compléter le système jusqu'à obtenir un système confluent.

Exercice 5 : Graphe de flot de contrôle (3 points)

Soit le programme Pseudo-Pascal suivant :

```
1 x :=  y * 2 + x;
2 y := (x - 2 * y) + z;
3 z := 2 * y + 1
```

1. Faire la sélection d'instruction pour transformer ce programme en UPP. On utilisera les règles de réécriture

$$\begin{aligned} \texttt{add}(X, \texttt{li}_k) &\to \texttt{addi}_k(X) \\ \texttt{mul}(X, 2^k) &\to \texttt{sll}_k(X) \\ \texttt{mul}(2^k, X) &\to \texttt{sll}_k(X) \end{aligned}$$

2. Transformer le programme en RTL.
3. Calculer la valeur symbolique des pseudo-registres au fur et à mesure des instructions.
4. Supprimer les calculs redondants et donner le programme RTL résultant.

Exercice 6 : Convention d'appel (3 points)

On considère le programme Pseudo-Pascal suivant :

```
1  program
2
3  var g : integer;
4
5  function f(n : integer) : integer;
6  var z : integer;
7  begin
8    if n < 1
9    then f := 2
10   else begin
11     g := f(n / 2);
12     z := n + g;
13     f := z * n
14   end
15 end;
16
17 procedure h(l, m : integer);
18 begin
19   g := l;
20   g := f(m)
21 end;
22
23 begin
24   h(2,3)
25 end.
```

On suppose qu'on utilise la convention d'appel MIPS comme vue en cours. On supposera que la variable locale `z` de la fonction `f` est stockée dans le registre sauvegardé par l'appelant `$t0`, et que la variable globale `g` est stockée dans le registre `$s0`. En tant que variable globale, `g` est partagée entre toute les fonctions et n'a donc pas besoin d'être sauvegardée.

1. Quels registres `f` doit-elle sauvegarder ? En déduire la trame de `f`.
2. Mêmes questions pour `h`.
3. Pour expliciter la convention d'appel, quelles instructions faut-il ajouter :

a) Au début de `f` ?

b) Avant l'appel à `f` dans `f` ?

c) Après l'appel à `f` dans `f` ?

d) À la fin de `f` ?

e) Au début de `h` ?

f) Avant l'appel à `f` dans `h` ?

g) Après l'appel à `f` dans `h` ?

h) À la fin de `h` ?

4. Peut-on optimiser l'appel à `f` dans `h` ?

Exercice 7 : Allocation de registres (4 points)

On considère le programme RTL suivant :

```
var %0, %1, %2, %3, %4, %5
entry l0
exit l4
l0: move %0, %1 -> l1
l1: add %3, %5, %2 -> l2
l2: bgtz %3 -> l3, l4
l3: move %4, %3 -> l5
l5: blez %5 -> l6, l7
l6: move %2, %4 -> l8
l7: move %2, %5 -> l8
l8: add %1, %5, %3 -> l9
l9: addi %5, %3, -2 -> l2
```

1. Dessiner le graphe de flot de contrôle correspondant.
2. Donner les variables vivantes en chacun des points du programme.
3. Quelles instructions pourraient-elles être éliminées ?
4. Dessiner le graphe d'interférence (avec les arêtes de préférence). On indiquera sur les arêtes le label d'une instruction qui a causé l'interférence ou la préférence.
5. Essayer de 2-colorier ce graphe en appliquant l'algorithme de George et Appel. On détaillera le déroulement de l'algorithme et on justifiera le critère utilisé lors d'une fusion.
6. En utilisant ce 2-coloriage, donner le programme RTL correspondant au programme initial, dans lequel on aura alloué les variables aux deux registres `$s0` et `$s1` et pour lequel on utilise `$t0` et `$t1` pour sauvegarder ou restaurer la valeur des variables éventuellement spillées.