Cours “Compilation avancée”

Examen de rattrapage

Exercice 1

Transformer le filtrage en cascade de tests élémentaires et rendre explicite les fermetures des fonctions `ext_lex` et `ordre` dans le fragment de code Caml ci-dessous.

```
type ordre = Égal | Inférieur | Supérieur

let extension_lexicographique ordre =
  let rec ext_lex = function
      ([], []) -> Égal
    | (_, []) -> Supérieur
    | ([], _) -> Inférieur
    | (h1::t1, h2::t2) ->
        let c = ordre(h1, h2) in if c = Égal then ext_lex(t1, t2) else c
  in ext_lex
```

Exercice 2

On considère la boucle suivante, qui applique une rotation du plan à un ensemble de points dont les coordonnées sont dans les tableaux `x[]` et `y[]`:

```
for i = 0 to N do
    new_x = x[i] * a - y[i] * b
    new_y = x[i] * b + y[i] * a
    x[i] = new_x
    y[i] = new_y
done
```

Après passage en code intermédiaire et allocation de registres, on obtient le code suivant pour le corps de la boucle:

```
        f1 ← load rx          /* A1 */
        f1 ← f1 * fa          /* A2 */
        f2 ← load ry          /* A3 */
        f2 ← f2 * fb          /* A4 */
        f1 ← f1 - f2          /* A5 */
        f3 ← load rx          /* A6 */
        f3 ← f3 * fb          /* A7 */
```

```
f4 ← load ry          /* A8 */
f4 ← f4 * fa          /* A9 */
f3 ← f3 + f4          /* A10 */
rx ← rx + 8           /* A11 */
ry ← ry + 8           /* A12 */
store(f1, rx - 8)     /* A13 */
store(f3, ry - 8)     /* A14 */
```

1. Construire le graphe de dépendances de données pour ce code. (On prendra les latences habituelles: 2 cycles pour `load`, 4 pour les opérations arithmétiques flottantes, 1 pour les autres opérations.) Marquer sur chaque arc le type de dépendance: read-after-write, write-after-read, write-after-write, ou dépendance sur la mémoire.

2. Obtenir un ordonnancement pour ce code en lui appliquant l'algorithme de *list scheduling*. Donner pour chaque cycle la liste des instructions activables, et justifier le choix de l'instruction émise lorsque plusieurs sont activables. (On suppose que le processeur ne peut lancer qu'une instruction par cycle.)

3. Quelle est la durée minimale d'exécution de ce code en supposant que le processeur a un nombre arbitraire d'unités de calcul et peut lancer un nombre arbitraire d'instructions à chaque cycle?

Exercice 3

On rappelle que le degré $D(r)$ d'un noeud r dans un graphe non orienté est le nombre de noeuds adjacents à r dans le graphe. Dans le cas d'un graphe d'interférence, le degré $D(r)$ d'un pseudo-registre r est donc le nombre de pseudo-registres qui interfèrent avec r.

Preston Briggs (1992) a proposé le critère suivant pour décider quand une instruction "move" $r \leftarrow r'$ peut être éliminée en identifiant r et r' dans tout le programme:

$$r \text{ et } r' \text{ n'interfèrent pas, et de plus } S(r) + S(r') < N$$

Dans ce critère, N est le nombre de registres disponibles et $S(r)$ est le nombre de noeuds adjacents "significatifs" de r: le nombre de pseudo-registres de degré $\geq N$ qui interfèrent avec r.

Montrer que ce critère est sûr vis-à-vis de l'allocation de registres, c'est-à-dire que si le graphe d'interférences initial est N-colorable, alors le graphe obtenu en identifiant deux noeuds r et r' satisfaisant le critère de Briggs est également N-colorable.