Fiche HAI705I - Compilation

Notions de cours

- Compilation à la volée
    - traduction dynamique “Just in Time”
    - Deux étapes : compil bytecode (portabilité) ➡ compil native (efficacité)
    - Nécessite de pouvoir compiler à l’exécution
    - Moins coûteux que de compiler depuis la source
    - VisualWorks (Smalltalk), LLVM, machine virtuelle de .NET, machines virtuelles de Java

- Compilation dynamique
    - Bibliothèques dynamiques chargées une seule fois en mémoire
    - Dépendance de l’exécutable vis-à-vis de bibliothèques dynamiques
    - Exécutable « standalone » (aucune dépendance)
    - Exécutable plus « lourd » (bibliothèques intégrées)
- CISC
    - « Complex Instruction Set Computer »
    - Plus vieux
    - Jeu étendu d’instructions complexes
    - Chaque instruction peut effectuer plusieurs opérations élémentaires ;
    - Jeu d’instructions comportant beaucoup d’exceptions ;
    - Instructions codées sur une taille variable.
- RISC
    - « Reduced Instruction Set Computer » ;
    - Jeu d’instructions réduit ;
    - Chaque instruction effectue une seule opération élémentaire ;
    - Jeu d’instructions plus uniforme ;
    - Instructions codées sur la même taille et s’exécutant dans le même temps (un cycle d’horloge en général)

PP

- Une fonction f retourne un résultat avec une variable (implicite) f
- Les variables ne sont pas systématiquement initialisées, et on a donc besoin de valeurs par défaut pour tous les types de données
- Le langage est typé
- Le langage est très réduit et peu expressif
- le langage est Turing-complet : tout algorithme peut être exprimé en utilisant ce langage

Exemple :

```
var n : integer
f (n : integer) : integer
var i : integer
if n = 0 then
  f : = 1
else
  f : = 1;
  i : = 1;
  while i ≤ n do
    f : = f × i;
    i : = i + 1

n : = read();
write(f (n))
```

MIPS

- Comporte 32 registres généraux (r0 .. r31)
  - r0 contient tout le temps 0
  - a0 .. a3, ra : réservés pour le passage d'arguments
  - v0 .. v1 : renvoi de résultat
  - s0 .. s7 : sauvegardés par l'appelé
  - t0 .. t9 : sauvegardés par l'appelant
  - sp, fp, gp : pointeurs vers la pile et les données
  - k0 .. k1 : réservés au noyau
  - at : réservé à l'assembleur
- trois types principaux d'instructions :
  - Les instructions de transfert entre registres et mémoire
    - lw et sw
  - Les instructions de calcul
    - (li dest src) (addi dest, src, cst) (move dest, src) (neg dest, src) (add dest, src1, src2),, (set on less than) (slt dest, src1, src2), sub, mul, div, sle, sgt, sge, seq, sne.
      - slt = 1 si le contenu de src1 est inf à src2, 0 sinon
  - Les instructions de saut

        - (j adr), bgtz, bgez, bltz, blez, beq, blt, bne, jal, jr, syscall,
- Exemple

```
.data
positive: .asciiz "positive\n"
negative: .asciiz "negative\n"

.text
main: li $v0 , 5
      syscall
      li $t0 , 0
      blt $v0 , $t0 , neg
      li $v0 , 4
      la $a0 , positive
      syscall
      j end
neg : li $v0 , 4
      la $a0 , negative
      syscall
end:
```

UPP

- Les types sont supprimés
- Les variables globales et locales sont distinguées
- Les opérateurs arithmétiques et ceux d'accès aux tableaux sont remplacés par les opérations MIPS, lw et sw
- Toute variable sera de taille fixe : 1 mot = 4 octets = 32 bits

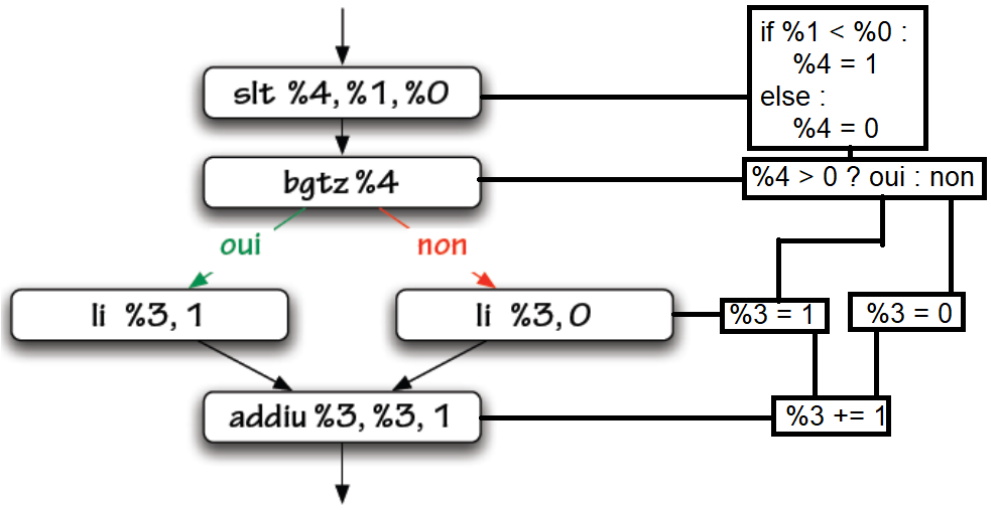
```
function f(n):
if n > 100 then
    f := n - 10
else
    f := f(f(n+11))
```

RTL

- Expressions et instructions structurées sont décomposées en instructions élémentaires organisées en graphe de flot de contrôle
- Les variables locales sont remplacées par des pseudo-registres
- Exemple

```
function f (%0) : %1
var %0, %1, %2, %3
entry f6
exit f0
f6 : li %1, 0 → f5
f5 : blez %0 → f4, f3
f3 : addiu %3, %0, -1 → f2
f2 : call %2, f (%3) → f1
f1 : mul %1, %0, %2 → f0
f4 : li %1, 1 → f0
```

- Graphe de flot de contrôle



ERTL

- La convention d'appel est le résultat d'un contrat entre plusieurs parties :
  - l'appelant (« caller ») ;
  - l'appelé (« callee ») ;
  - le système d'exécution (« runtime system ») ;
  - le système d'exploitation (« operating system »).
- Procédures et fonctions ne sont plus distinguées ;
- L'adresse de retour devient un paramètre explicite ;
- L'allocation et la désallocation des trames de pile devient explicite ;
- Les registres « callee-save » sont sauvegardés de façon explicite.
- newframe : début de trame, $sp est décrémenté ;

- delframe : fin de trame, $sp est incrémenté ;
- « move » d'un registre physique vers un pseudo-registre, et vice versa.
- À propos des registres paramètres $a0-$a3
    - Ils sont à sauvegarder (dans des pseudo-registres) s'il y a des appels de procédures dans la procédure en question et que ces paramètres sont utilisés après l'un de ces appels
    - Les pseudo-registres utilisés pour la sauvegarde de ces paramètres peuvent être réalisés :
        - Soit par des registres physiques « callee-save »
        - Soit en le « spillant » (en le mettant sur la pile)
- Exemple

```
procedure f (1)
var %0, %1, %2, %3, %4, %5, %6
entry f 11
f11 : newframe → f10
f10 : move %6, $ra → f9
f9 : move %5, $s1 → f8
f8 : move %4, $s0 → f7
f7 : move %0, $a0 → f6
f6 : li %1, 0 → f5
f5 : blez %0 → f4, f3
f3 : addiu %3, %0, -1 → f2
f2 : j → f20
f20 : move $a0, %3 → f19
f19 : call f (1) → f18
f18 : move %2, $v0 → f1
f1 : mul %1, %0, %2 → f0
f0 : j → f17
f17 : move $v0, %1 → f16
f16 : move $ra, %6 → f15
f15 : move $s1, %5 → f14
f14 : move $s0, %4 → f13
f13 : delframe → f12
f12 : jr $ra
f4 : li %1, 1 → f0
```

Durée de vie et interférence

Notions :

- Il faut savoir en quels points du code chaque variable est vivante,
- Deux variables peuvent être réalisées par le même emplacement si elles n'interfèrent pas
- Les instructions « move » suggèrent des emplacements préférentiels
- Les registres $v0-$v1, $a0-$a3, $t0-$t9 (« caller-save »), $s0-$s7 (« callee-save ») et $ra sont allouable
- Variables :
    - Une variable v est vivante au point p (d'un programme) s'il existe un chemin menant de p à un point p′ où v est utilisée et si v n'est pas définie (affectée) le long de ce chemin
    - Une variable est morte au point p (d'un programme) si tous les chemins à partir de p mènent à des points p′ i où v est définie (affectée) et si v n'est pas utilisée le long de ces chemins
- Interférence :
    - Deux variables distinctes interfèrent si l'une est vivante à la sortie d'une instruction qui définit l'autre
    - Deux variables qui n'interfèrent pas peuvent être réalisées par un unique emplacement (registre physique ou emplacement de pile)
    - Inversement, deux variables qui interfèrent doivent être réalisées par deux emplacements distincts.
    - Supposons y vivante à la sortie d'une instruction qui définit x ; si la valeur reçue par x est certainement celle de y , alors il n'y a pas lieu de considérer que les deux variables interfèrent

Méthode :

1. Partir du bas et remonter --- **Variables vivantes et mortes**
2. Pour chaque instruction :
    a. Si une des variable précédemment vivante n'est pas définie dans l'instruction du haut on l'ajoute à la liste, sinon on l'enlève
    b. Si une variable est utilisée dans l'instruction on l'ajoute dans la liste
3. Aller de haut en bas --- **Interférence et Préférence**
4. Pour chaque **assignation :**
    a. Toute variable vivante après une assignation est en interférence avec celle qui vient d'être assignée
    b. Sauf si x := y alors x n'interfère pas avec y
    c. Dans cette exception (ou x est assigné par une seule variable y) alors x est en préférence avec y
5. Pour dessiner le graphe :
    a. on dessine chaque sommet
    b. on relie les sommets en interférence d'une couleur
    c. ceux en préférence d'une autre