



Vérifiez votre énoncé: les 6 entêtes doivent être +1/1/*xx*+... +1/6/*xx*+

Ing1 2016 – CMP2 – 1h30 – Juin 2014 *Sans document ni machine*

Noircir les cases plutôt que cocher. Renseigner les champs d'identité. Les questions marquées du symbole ♣ peuvent avoir plusieurs réponses justes. Toutes les autres questions n'ont qu'une seule réponse juste; si plusieurs réponses sont valides, sélectionner la plus restrictive (par exemple s'il est demandé si 0 est *nul*, *non nul*, *positif*, ou *négatif*, sélectionner *nul*). Il n'est pas possible de corriger une erreur. Les réponses justes créditent; les incorrectes pénalisent; et les blanches et réponses multiples valent 0.

Nom et prénom :

.....
.....
.....
.....

Cochez votre identifiant (de haut en bas):

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

☐0 ☐1 ☐2 ☐3 ☐4 ☐5 ☐6 ☐7 ☐8 ☐9

1 Contrôle

Q.1 Avez-vous bien vérifié les en-têtes des 6 pages de ce sujet, comme indiqué en haut de cette première page ?

☐ Oui

☐ Non

2 Généralités

Q.2 Parmi les étapes suivantes, laquelle ne fait pas partie du *back end* (partie terminale) d'un compilateur ?

☐ Les optimisations relatives au langage cible

☐ La sélection d'instructions

☐ L'analyse sémantique

☐ L'analyse de vivacité

3 Typage

Q.3 Utiliser des références à la place de pointeurs en C++ produit du code

☐ moins sûr.

☐ plus rapide.

☐ équivalent en vitesse d'exécution.

☐ plus lent.

Q.4 Quel est le type de la fonction `f` définie ci-dessous en Objective Caml ?

```
let f x = x ();;
```

☐ Ce code est invalide

☐ `'a -> unit`

☐ `(unit -> 'a) -> 'a`

☐ `'a -> 'b`



Q.5 Quel est le type de la fonction `g` définie ci-dessous en Objective Caml ?

```
let g x = x x;;
```

- | | |
|---|--|
| ☐ <code>'a -> unit</code> | ☐ <code>(unit -> 'a) -> 'a</code> |
| ☐ Ce code est invalide | ☐ <code>'a -> 'b</code> |

4 Désucrage

Q.6 Quel est le problème avec le désucrage des boucles `for` du langage Tiger proposé ci-dessous ?

```
for i := min to max  
do body
```

~

```
let var i      := min  
    var limit := max  
in  
  while i <= limit  
  do  
    (body; i := i + 1)  
end
```

- ☐ La portée de `i` est modifiée.
- ☐ Le cas où `min = INT_MIN` n'est pas correctement traité.
- ☐ Les bornes `min` et `max` ne sont évaluées qu'une seule fois.
- ☐ Le cas où `max = INT_MAX` n'est pas correctement traité.

5 Mémoire

Q.7 Le *frame pointer* sert à

- | | |
|--|--|
| ☐ accéder aux variables globales du programme. | ☐ manipuler des données allouées dynamiquement sur le tas en automatisant la gestion de la mémoire. |
| ☐ accéder à l'instruction courante. | |
| ☐ accéder aux variables locales d'une fonction. | |

Q.8 Quel mécanisme permet d'allouer de la mémoire dynamiquement sur la pile en C++ ?

- | | |
|--|--|
| ☐ <code>new</code> | ☐ <code>malloc</code> |
| ☐ <code>alloca</code> | ☐ <code>std::allocator</code> |

6 Traduction

Q.9 Lequel de ces langages est un langage de machine à pile ?

- | | |
|---|--|
| ☐ Le langage LLVM | ☐ Le langage RTL de GCC |
| ☐ Le bytecode Java | ☐ Le langage Tree |



Q.10 On considère le programme Tiger ci-dessous.

```
1 let
2   function f() =
3     let
4       function f1() = ()
5       function f2() = f1()
6     in
7     end
8 in
9 end
```

En supposant que le *static link* d'une fonction est situé à l'adresse pointée par son frame pointer (*fp*), quelle valeur *f2* doit elle passer en argument comme static link à *f1* lors de l'appel de la ligne 5 ?

- | | |
|---|--------------------------------------|
| ☐ <i>fp</i> | ☐ <i>*fp</i> |
| ☐ Ce programme ne compile pas. | ☐ <i>**fp</i> |

Q.11 Quel code Tiger a produit le code HIR ci-dessous ?

```
label 11
cjump ne const 1 const 0 name 12 name 10
label 12
sxp const 0
jump name 11
label 10
```

- | | | | |
|--------------------------------------|---------------------------------------|--|--------------------------------|
| ☐ (1 <> 0; 0) | ☐ if 1 then () | ☐ while 1 do () | ☐ 1 & 0 |
|--------------------------------------|---------------------------------------|--|--------------------------------|

Q.12 Quelle est la traduction de l'expression Tiger *a := a + 1*, vers une représentation intermédiaire haut niveau (HIR) exprimée dans le langage Tree ? On considérera que *a* est une variable stockée sur la pile à l'adresse *fp - 4* où *fp* désigne le frame pointer.

- | |
|--|
| ☐ <code>move(mem(temp a), binop(add, mem(temp a), const 1))</code> |
| ☐ <code>move(binop(add, temp fp, const -4),</code>
<code>binop(add, binop(add, temp fp, const -4), const 1))</code> |
| ☐ <code>move(mem(binop(add, temp fp, const -4),</code>
<code>binop(add, mem(binop(add, temp fp, const -4)), const 1))</code> |
| ☐ <code>move(temp a, binop(add, temp a, const 1))</code> |

Q.13 Parmi les exemples de variables ci-dessous, laquelle peut on placer dans un registre ?

- | |
|--|
| ☐ Une variable tableau |
| ☐ Une variable utilisée plusieurs fois comme <i>l-value</i> |
| ☐ Une variable avec des utilisations non-locales |
| ☐ Une variable dont on prend l'adresse |

7 Canonisation

Pour chacun des programmes HIR ci-dessous, sélectionnez le programme « canonisé » (i.e., LIR) qui lui correspond. Par convention les termes '*s**' désignent des instructions (*statement*), '*e**' des expressions, '*l**' des étiquettes (*label*) et '*t**' des temporaires. Nous utilisons les notations du projet Tiger de l'EPITA : *sxp* au lieu de *exp* chez Andrew Appel, *seq* avec un nombre d'éléments arbitraire, etc.



Q.14 `seq(s1, seq(s2, s3), s4)`

- | | |
|---|---|
| ☐ <code>seq(s1, s3, s4)</code> | ☐ <code>seq(seq(seq(s1, s2), s3), s4)</code> |
| ☐ <code>seq(s1, s2, s3, s4)</code> | ☐ <code>eseq(seq(s1, s2, s3), s4)</code> |

Q.15 `mem(eseq(s, e))`

- | | |
|---|---|
| ☐ <code>seq(s, sxp(mem(e)))</code> | ☐ <code>eseq(move(temp t, e), mem(e))</code> |
| ☐ <code>seq(s, sxp(e))</code> | ☐ <code>eseq(s, mem(e))</code> |

Q.16 `sxp(call(label l, e1, eseq(s, temp t))`

- | |
|---|
| ☐ <code>seq(move(temp t1, e1), s, sxp(call(label l, temp t1, temp t)))</code> |
| ☐ <code>seq(s, sxp(call(label l1, e1, temp t)))</code> |
| ☐ <code>seq(move(temp t1, e1), move(temp t2, s), sxp(call(label l, temp t1, temp t)))</code> |
| ☐ <code>seq(s, move(temp t1, e1), sxp(call(label l, temp t1, temp t)))</code> |

8 Génération de code

Q.17 Quel code Tiger a produit le code MIPS ci-dessous ?

```
sw      $fp, -4 ($sp)
move    $fp, $sp
sub     $sp, $sp, 8
sw      $ra, ($fp)
li      $ra, 1
blt     $ra, 2, 10
12:     j      14
10:     li     $a0, 51
jal     tc_print_int
j       12
14:     lw     $ra, ($fp)
move    $sp, $fp
lw      $fp, -4 ($fp)
jr      $ra
```

- | | |
|---|--|
| ☐ <code>while 1 < 2 do print_int(51)</code> | ☐ <code>if 1 < 2 then print_int(51)</code> |
| ☐ <code>(1 < 2; print_int(51))</code> | ☐ <code>1 < 2 print_int(51)</code> |

Q.18 Quel(s) registre(s) n'est (ne sont) pas sauvegardé(s) sur la pile à l'entrée d'une fonction puis restauré(s) à la sortie de celle-ci ?

- | | |
|---|---|
| ☐ Le frame pointer | ☐ L'adresse de retour |
| ☐ Les registres <i>caller-save</i> | ☐ Les registres <i>callee-save</i> |

Q.19 Parmi les caractéristiques ci-dessous, laquelle est typique d'une machine CISC ?

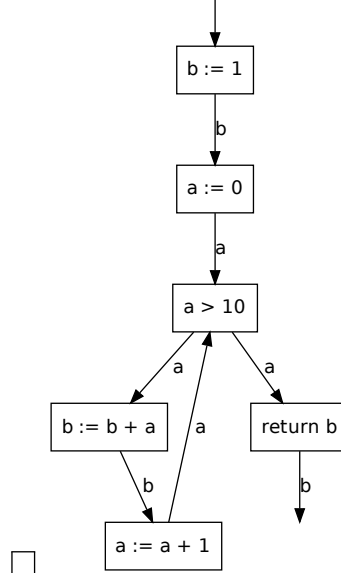
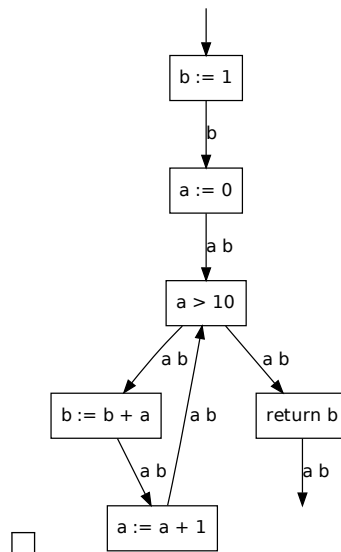
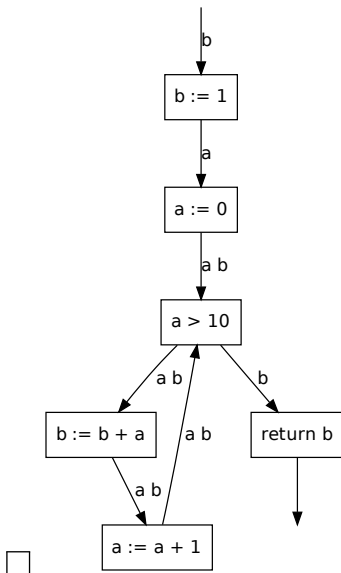
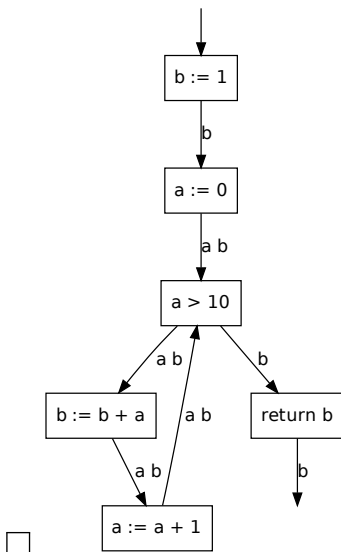
- | | |
|---|--|
| ☐ Des instructions machine de taille fixe | ☐ La présence de 32 registres |
| ☐ Une seule classe de registres entier/pointeurs | ☐ Les opérations arithmétiques pouvant utiliser tant les registres que la mémoire |

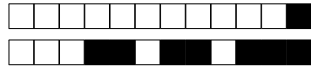


9 Analyse de vivacité

Q.20 Quel graphe de vivacité correspond au programme ci-dessous ?

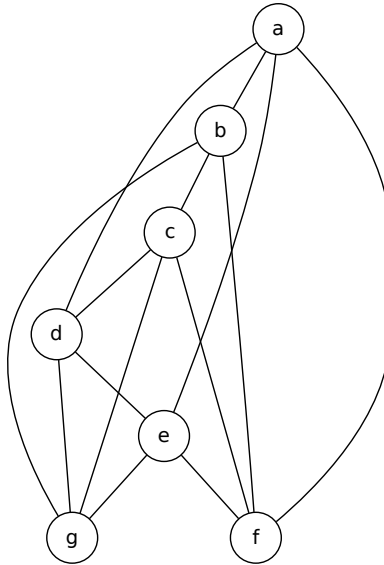
```
b := 1
a := 0
L1: if a > 10 goto L2
    b := b + a
    a := a + 1
    goto L1
L2: return b
```





10 Allocation de registres

Q.21 On s'intéresse ici à coloration de graphe par simplification. Le graphe ci-dessous...



- ☐ ... est coloriable avec 4 couleurs sans faire appel à une étape de *spilling*.
- ☐ ... n'est pas coloriable avec 4 couleurs.
- ☐ ... est coloriable avec 4 couleurs en réalisant un *spill* effectif.
- ☐ ... est coloriable avec 4 couleurs grâce à un *spill* potentiel (mais pas effectif).

Fin de l'épreuve.