Examen Session 2
Traduction et Compilation de Programmes
février 2017
Durée 1H

Aucun document autorisé

Instructions **à lire attentivement !**

1. Les exercices sont indépendants.
2. On vous donne des indications de temps.
3. Vous répondrez dans les cadres et rendrez l'examen entier **agraphé**, et à **l'intérieur d'une copie d'examen anonyme**. Attention à bien reporter le numéro d'anonymat dans le cadre prévu ci-dessous (en gros).
4. Merci de justifier vos réponses.

Numéro d'anonymat :

Exercice 1 – Génération de code 3-adresses (20min)

On rappelle la syntaxe abstraite du mini-language while/Mu :

$S(Smt)$	::=	$x := e$	*affectation*
	|	`skip`	*ne rien faire*
	|	$S_1; S_2$	*sequence*
	|	`if` b `then` S_1 `else` S_2	*test*
	|	`while` b `do` S `done`	*boucle*

avec e une expression numérique (entiers, +, ...) et b une expression booléenne (`true`, or, ...).

La figure 1 disponible en annexe donne les règles de génération de code 3 adresses que vous devez utiliser pour l'exercice. **Dans toute la suite on considère que les constantes numériques utilisées sont suffisamment petites pour être codées sur 5 bits en complément à 2.**

Question #1

Remplir le code **généré par les règles** pour le programme suivant :

```
i:=10;
while(i>0)
  i = i - 1;
```

;;Automatically generated code
;;(stat (assignment x = (expr (atom 10)) ;)) ;;2 lignes (x−>temp_1)

;;(stat (while_stat while (expr (atom ((expr (expr (atom 0)) < (expr (atom i)))))) (
stat_block { (block (stat (assignment x = (expr (expr (atom x)) − (expr (atom 1))) ;)))
}))))
l_while_begin_1: *;; calcul du test 0−i (5 lignes)*

BRzp l_cond_neg_2
AND temp_5 , temp_5 , 0
ADD temp_5 , temp_5 , 1
BR l_cond_end_2
l_cond_neg_2:

AND temp_5 , temp_5 , 0
l_cond_end_2:
BRz l_while_end_1
;;(stat (assignment x = (expr (expr (atom x)) − (expr (atom 1))) ;)) ;;5/6lignes

;;; 1 branchement ici

l_while_end_1:
;;; the end

Question #2

Quel serait le code généré pour le programme suivant ? *Pour aller plus vite, vous pouvez numéroter des blocs dans le programme précédent et uniquement reporter ces numéros.*

```
i:=10;
do
  i = i - 1;
while(i>0);
```

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

Question #3

Écrire **et JUSTIFIER** la règle de génération de code pour le `do... while` :

(Stm)do S while b	

Exercice 2 – Data Flow (20min)

On rappelle qu'*une variable est vivante en sortie d'un bloc si il existe un chemin partant de ce bloc qui mène à une utilisation de cette variable sans redéfinition de cette variable.*

On redonne aussi les équations data-flow pour calculer les variables vivantes à l'entrée (In) et à la sortie de chaque bloc (Out) :

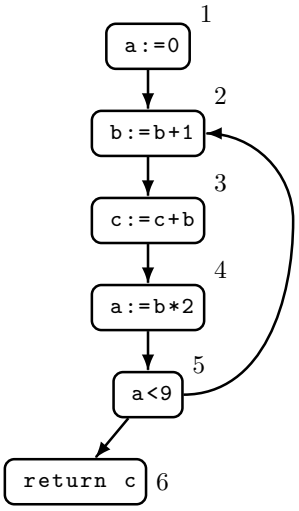
$$Out(\ell) = \begin{cases} \emptyset & \text{si } \ell\text{est final} \\ \bigcup\{In(\ell') |\ \ell'\text{successeur dans le graphe de flot}\} \end{cases}$$

$$In(\ell) = \big(Out(\ell) \backslash kill_{LV}(\ell)\big) \cup gen_{LV}(\ell)$$

avec :

- $kill_{LV}(\ell)$ l'ensemble des variables assignées dans le bloc.
- $gen_{LV}(\ell)$ une variable apparaissant en membre de droite du bloc ℓ (dont l'usage implique qu'elle *doit être définie* dans un bloc précédent).

On donne le graphe de flot de contrôle suivant :



Question #1

Donner les valeurs des ensembles *Gen* et *Kill* pour chaque bloc dans le tableau ci dessous. Le bloc 6 est final.

Question #2

Calculer les ensembles de variables vivantes en sortie (Out) et en entrée (In) de chaque bloc, en remplissant le tableau suivant : **attention, faites l'analyse en arrière, i.e. en commençant par le bloc 6, vous ferez moins d'itérations**.

			Étape 1		Étape 2		Étape 3		Étape 4	
ℓ	$kill(\ell)$	$gen(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$	$In(\ell)$	$Out(\ell)$
6										
5										
4										
3										
2										
1										

Exercice 3 - Génération de code & Allocation de registres (20 min)

Soit le programme Mu suivant :

```
x:=3;
y:=2+x;
z:=x+y;
t:=x+y+z;
```

La production de code 3 adresses du TP4 produit le code décrit dans le tableau ci-dessous, et l'analyse de durée de vie du TP5 produit les durées de vie représentées dans le tableau (chaque étoile dans une ligne signifie "le temporaire est vivant en entrée de cette ligne de code) :

code	t1	t2	t3	t4	t5	t6	t7	t8	t9	t10
`AND t1 , t1 , 0`										
`ADD t1 , t1 , 3`	*									
`ADD t2 , t1 , 0`	*									
`AND t3 , t3 , 0`		*								
`ADD t3 , t3 , 2`		*	*							
`ADD t4 , t3 , t2`		*	*							
`ADD t5 , t4 , 0`		*		*						
`ADD t6 , t2 , t5`		*			*					
`ADD t7 , t6 , 0`		*			*	*				
`ADD t8 , t2 , t5`		*			*					
`ADD t9 , t8 , t7`							*	*		
`ADD t10 , t9 , 0`									*	
`HALT`										*

Question #1

Tracer le graphe d'interférence.

Question #2

Le coloriage à 2 couleurs suivant est-il valide ?

— Bleu : $t2, t6, t7$

— Rouge : $t1, t3, t4, t5, t8, t9, t10$

. . . . . .

. . . . . .

. . . . . .

Question #3

On décide de "spiller" $t2$. Les autres registres restent coloriés à Rouge (registre R1) ou Bleu (registre R2) comme précédemment. Donner le code finalement produit. R6 pointeur de pile, R5 et R7 pour la gestion des variables spillées.

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

. . . . . .

(Stm) `x := e`	`dr <- GenCodeExpr(e)`
	`if x has a location loc:`
	`  code.add(instructionADD(loc,dr,0))`
	`else:`
	`  storeLocation(x,dr)`

`while` b `do` S `done`	`ltest,lendwhile=newLabels()`
	`code.addLabel(ltest)`
	`dr <-GenCodeExpr(b)  #dr is the last written register`
	`code.add(InstructionBRz(lendwhile) #if 0 jump to end`
	`GenCodeSmt(S)  #else (execute S`
	`code.add(InstructionBR(ltest))  #and jump to the test)`
	`code.addLabel(lendwhile)`

(constant expression) c	`#not valid if c is too big`
	`dr <-newTemp()`
	`code.add(InstructionAND(dr, dr, 0))`
	`code.add(InstructionADD(dr, dr, c))`
	`return dr`

e ::= e_1-e_2	`t1 <- GenCodeExpr(e_1)`
	`t2 <- GenCodeExpr(e_2)`
	`dr <- newTemp()`
	`code.add(InstructionNOT(dr, t2))`
	`code.add(InstructionADD(dr, dr, 1))`
	`code.add(InstructionADD(dr, dr, t1))`
	`return dr`

e ::= $e_1 < e_2$	`dr <-newTemp()`
	`t1 <- GenCodeExpr (e1-e2)  #last write in register`
	`(lfalse,lend) <- newLabels()`
	`code.add(InstructionBRzp(lfalse))  #if =0 or >0 jump!`
	`code.add(InstructionAND(dr, dr, 0))`
	`code.add(InstructionADD(dr, dr, 1))  #dr <- true`
	`code.add(InstructionBR(lend))`
	`code.addLabel(lfalse)`
	`code.add(InstructionAND(dr, dr, 0))  #dr <- false`
	`code.addLabel(lend)`
	`return dr`

FIGURE 1 – Génération de code