Projet 2: ajout des fonctions

cours: Daniel Hirschkoff
TPs: Paul Renaud-Goud

Organisation – rendus

- votre compilateur engendre du code pour des programmes IMP (Pascal sans fonctions)
  si tout marche à merveille, vous avez fait l'essentiel du travail
- restent deux rendus:
  1. 3/5 ajout des fonctions
  2. 23/5 tout++ (à préciser, suivant les groupes)
- en deux déclinaisons (pour ceux qui compilent Pascal)
  1. groupe 1 *tout dans la pile*

     R. Crubille R. Nolllet G. Marti R. Tetley O. Manoussakis M. Schmitt X. Dumont
  2. groupe 2 *vivacité / registres*

     A. Mottet L. Parlant B. Simon A. Durier G. Gilbert M. Savaro F. Dross M. Rosenfeld

     H. Derycke T. Pecatte

Fonctions: code intermédiaire

- dans le code intermédiaire, on ajoute l'appel aux fonctions
  on va engendrer typiquement des `Move(tmp`$_i$`, Call(f,arglist))`
  voire des `Move(tmp`$_i$`, Call(f,tmplist))`

  (car on a calculé les arguments avant l'appel)

Fonctions: code intermédiaire

- dans le code intermédiaire, on ajoute l'appel aux fonctions
  on va engendrer typiquement des `Move(tmp`$_i$`, Call(f,arglist))`
  voire des `Move(tmp`$_i$`, Call(f,tmplist))`
  (car on a calculé les arguments avant l'appel)
- deux volets dans la compilation des fonctions
  - compilation de la définition d'une fonction
  - compilation des appels à une fonction
- à l'interface: *opacité*, via des conventions d'utilisation des registres

Compilation de la définition d'une fonction

- une adresse où sauter (label) pour la fonction
- les arguments et les variables locales: de nouveaux temporaires

  la table des symboles peut servir à stocker la correspondance paramètre/temporaire
- corps de la fonction: comme pour le programme principal

  allocation de registres incluse (mais: cf. plus tard)

Compilation de la définition d'une fonction

- une adresse où sauter (label) pour la fonction
- les arguments et les variables locales: de nouveaux temporaires

  la table des symboles peut servir à stocker la correspondance paramètre/temporaire
- corps de la fonction: comme pour le programme principal

  allocation de registres incluse (mais: cf. plus tard)
- convention: les fonctions renvoient leur résultat dans `v0`

  `function f(..):..  begin ..  f:= ..  end;`

  sera compilé en `..  ; Move(tmp`$_{v0}$`, tmp`$_i$`)`
  - des conventions d'utilisation des registres en lien avec les appels de fonctions
  - des temporaires (registres virtuels) correspondant aux registres réels ("pré-allocation")

Compilation de la définition d'une fonction

- une adresse où sauter (label) pour la fonction
- les arguments et les variables locales: de nouveaux temporaires

  la table des symboles peut servir à stocker la correspondance paramètre/temporaire
- corps de la fonction: comme pour le programme principal

  allocation de registres incluse (mais: cf. plus tard)
- convention: les fonctions renvoient leur résultat dans `v0`

  `function f(..):..  begin ..  f:= ..  end;`

  sera compilé en `..  ; Move(tmp`$_{\texttt{v0}}$`, tmp`$_i$`)`
  - des conventions d'utilisation des registres en lien avec les appels de fonctions
  - des temporaires (registres virtuels) correspondant aux registres réels ("pré-allocation")
- arguments passés par référence: à vous de voir s'il faut faire quelque chose de particulier

SPIM: conventions pour les registres

- 0,1, 26-31: pas touche (usages spéciaux)
- 2,3, v0,v1: évaluation, résultat de fonction
- 4-7, a0-a3: arguments des fonctions
- 8-15, 24, 25, t0-t9: à sauvegarder par l'appelant

  caller-save
- 16-23, s0-s7: à sauvegarder et restaurer par l'appelé

  callee-save

Compilation d'un appel de fonction

- On commence par calculer la valeur des arguments
- Le *prologue* doit:
  1. Allouer k mots en pile.
  2. Sauvegarder les registres *callee-saved* dans des temporaires frais
  3. Placer les arguments dans les temporaires réservés à cet effet
- Symétriquement, l'*épilogue* doit:
  1. Placer le résultat éventuel dans le registre conventionnel `v0`
  2. Ressusciter les registres *callee-saved*
  3. Libérer les k mots en pile
  4. Sauter à l'adresse de retour
- l'allocation de registres a lieu vers la fin

  on retarde donc l'écriture des prologues et épilogues, ou alors on les écrit avec des constantes à définir (`taille_frame`)

Temporaires et appels de fonctions

- au moment de l'appel d'une fonction, les arguments sont dans des temporaires de l'appelant; après l'appel, ils sont dans des temporaires de l'appelé
- ainsi, les arguments voyagent: lorsqu'une fonction f appelle une fonction g (avec un seul argument pour simplifier):
  - la valeur de l'argument est préparée dans un temporaire tf (vue de l'appelant) et transféré dans a0 avant l'appel à g.
  - à l'entrée de g la valeur de a0 est placée dans un temporaire tg (vue de l'appelé).
  - on a donc trois emplacements pour l'argument: tf (avant), a0 (pendant) et tg (après).
- ces mouvements sont nécessaires dans le cas général pour libérer les registres conventionnels pour d'autres appels de fonctions.
- mais ils seront éliminés, si possible, par l'allocateur de registres en choisissant le même registre a0 pour tf et tg *(coalescing)*

Fonctions et allocations de registres

- un appel `Move(tmp`$_i$`, Call(f,arglist))` a dans son **def**:
  - `$ra, $v0`
  - les registres *caller-save*
  - les registres servant à passer les arguments
  - **pas** les registres callee-save (responsabilité de f)

Fonctions et allocations de registres

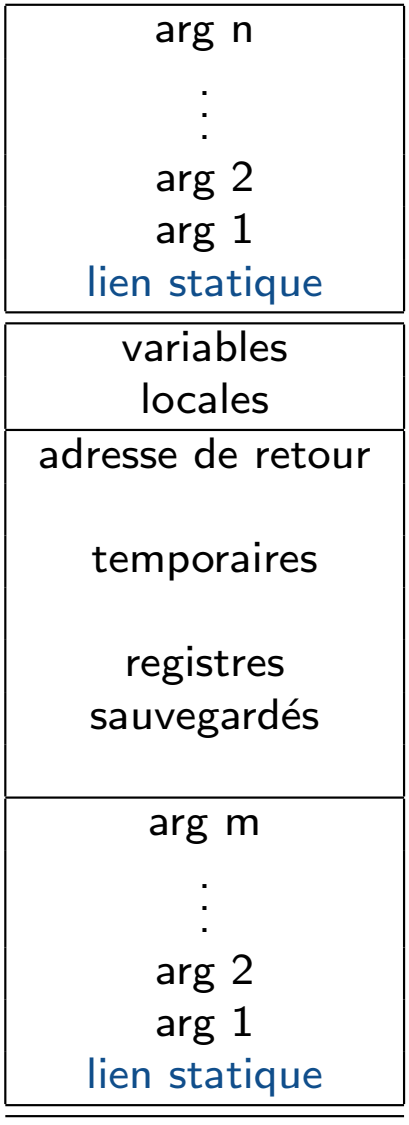
- un appel `Move(tmp`$_i$`, Call(f,arglist))` a dans son **def**:
  - `$ra, $v0`
  - les registres *caller-save*
  - les registres servant à passer les arguments
  - **pas** les registres callee-save (responsabilité de f)

- un saut de sortie de fonction a dans son **use**:
  - `$ra`: nécessaire pour sauter
  - `$v0`: sera utilisé dans la suite
  - les registres *callee-save*: contiennent ce qu'on pense qu'ils contiennent

Fonctions et allocations de registres

- un appel `Move(tmp`$_i$`, Call(f,arglist))` a dans son **def**:
  - `$ra, $v0`
  - les registres *caller-save*
  - les registres servant à passer les arguments
  - **pas** les registres callee-save (responsabilité de f)
- un saut de sortie de fonction a dans son **use**:
  - `$ra`: nécessaire pour sauter
  - `$v0`: sera utilisé dans la suite
  - les registres *callee-save*: contiennent ce qu'on pense qu'ils contiennent
- allocation de registres pour (prologue+corps+épilogue)

Organisation d'un enregistrement d'activation



arguments entrants

enregistrement précédent

← `$fp`

enregistrement courant

arguments sortants

← `$sp`

(cf. chap. 6 livre d'Appel)

lien statique: pointeur vers l'enregistrement de la fonction qui *domine statiquement*
(= dans le code)
la fonction courante

. . . mais **on n'a pas de fonctions imbriquées** (λ-lifting)

$\rightsquigarrow$ **simplification** (pour certains d'entre vous)

À faire

- étendre le code intermédiaire avec les appels de fonctions
- vivacité et registres pour les fonctions
- définir le plan de vos enregistrements d'activation
- au passage, enrichissements de la table des symboles

- penser aux cas particuliers:
  - appels terminaux:
    on sort de la fonction en en appelant une autre

    récursivité terminale notamment
  - procédures n'appelant pas de fonctions: `v0` est disponible