TD n°4, I45 – Projet
L2 Sciences et Technologies
Université du Sud Toulon-Var
Année universitaire 2005 – 2006

Emmanuel Bruno

Génération de code

Objectif

L'objectif de ce TD est générer le code assembleur correspondant aux facteurs, termes et aux expressions simples du langage Algo.

Exercice 1 : génération de code pour des expressions sans variables

Ecrire « à la main » (i.e. de façon intuitive) le code assembleur correspondant aux expressions suivantes :

- `3 + 2`
- `3 + 2 * 4`
- `2 * 4 + 3`

Remarquer que les registres sont utilisés comme une pile.

La table des symboles

Bien que la description de notre langage soit basée sur une grammaire *hors contexte*, notre langage n'est pas complètement indépendant du contexte. Ainsi, chaque identificateur utilisé (les variables par exemple) doit avoir été déclaré. Bien que cette gestion ne rentre pas dans le cadre de la grammaire, elle peut être aisément réalisée en étendant l'analyseur syntaxique.

On représentera chaque identificateur déclaré dans une liste chaînée. Chaque entrée dans la liste aura plusieurs attributs :

- **classe :** variable, constante, procédure, …
- **type :** entier, réel, booléen, …
- **adresse :** adresse en mémoire à laquelle est stockée la valeur associée à la variable.

Pour plus de simplicité, on supposera dans le cadre de ce projet que le table des symboles est cconstruction au fur et à mesure de l'analyse syntaxique de la partie déclaration et que les adresses des variables sont fixées au fur et à mesure en partant de la fin de la mémoire. On supposera aussi que toutes les variables sont stockées sur quatre octets.

Exercice 2 : génération de code avec variables

On note SP le numéro du registre du sommet de pile. Indiquer le code généré lorsqu'on lit un facteur, un terme ou une expression simple ainsi que les variations de SP. Tester avec les expressions `a + b + c + d` et `( a + b ) * ( c – d )`.

Exercice 4 : génération de code pour les instructions

Quel est le code généré par les instructions suivantes :

- `demander_une_valeur_pour` *`identificateur`*
- `affecter_la_valeur_de` *`expression`* `à` *`identificateur`*

Exercice 5 : génération de code pour une expression

Ecrire le code assembleur généré par une expression.

Remarque : les structures conditionnelles sont implantées à l'aide de sauts.

Exercice 6 : exemple simple de structure conditionnelle

Ecrire le code assembleur correspondant au programme suivant :

```
si (x mod 2 = 0) alors x=x-2 sinon x=x-1
finsi
```

Exercice 7 : forme générale du si…alors…sinon…

Donner la forme générale du code machine correspondant à une instruction conditionnelle de la forme :

- `si expression alors {instructions}`
  `finsi`
- `si expression alors {instructions}`
  `sinon {instructions} finsi`

Exercice 8 : génération de code pour le tant_que

Sur le même modèle que pour le *si…alors…sinon…*, donner le code associé à *tant_que {expression} faire {instructions}*.

TD n°5, I41 – Projet
DEUG MIAS/MASS 2nde année
Université de Toulon et du Var
Année universitaire 2002 – 2003

Emmanuel Bruno

Génération de code : correction

Exercice 1 : génération de code pour des expressions sans variables

`3 + 2`	`3 + 2 * 4`	`2 * 4 + 3`

addi 1,0,3
addi 2,0,2
add 1,1,2
	addi 1,0,3
	addi 2,0,2
	addi 3,0,4
	mul 2,2,3
	add 1,1,2
		addi 1,0,2
		addi 2,0,4
		mul 1,1,2
		addi 2,0,3
		add 1,1,2

Exercice 2 : génération de code avec variables

Pour un facteur :

- **entier :** addi SP,0,val SP++
- **identificateur :** ldw SP,0,addr(*ident*) SP++
- **(expression) :** code(*expression*)
- **non facteur :** code(*facteur*)
  addi SP,0,1 SP--
  bic SP,SP+1,SP
- **vrai :** addi SP,0,1 SP++
- **faux :** addi SP,0,0 SP++

Pour un terme :

- Code($facteur_1$)
- Pour chaque couple *opérateur* $facteur_n$ suivant faire
  code($facteur_n$)
  opérateur SP-1,SP-1,SP SP--

Pour une expression-simple :

- Si on lit un -
  code($terme_1$)
  sub(SP,0,SP)

- Sinon
  code(*terme*$_1$)
- Pour chaque couple *opérateur terme*$_n$ suivant faire
  code(*terme*$_n$)
  opérateur SP-1,SP-1,SP SP--

Exercice 3 : limite de la génération de code

Pour `x + 1 :`

Solution normale	Solution optimisée

ldw 1,0,addr(x)
addi 2,0,3
add 1,1,2
	ldw 1,0,addr(x)
	addi 1,1,3

➔ attention aux problèmes posés par les **expressions constantes**

➔ attention également au **nombre de registres utilisés** (30 au maximum en même temps)

Exercice 4 : génération de code pour les instructions

Pour `Demander_une_valeur pour identificateur:`

- rd SP,0,0
- stw SP,0,addr(*identificateur*)

Pour `Affecter_la_valeur_de expression à identificateur:`

- code(*expression*)
- stw SP,0,addr(*identificateur*)

Exercice 5 : génération de code pour une expression

code(*expression-simple*)
S'il y a un couple *opérateur expression_simple*$_2$
code(*expression-simple*$_2$)
cmp SP-1,SP-1,SP SP--
inverse opérateur SP, 0,?
Si on n'est pas dans une condition
addi SP-1,0,1 /* vrai */
bsr 0,0,2
addi SP-1,0,0 /* faux */

Exercice 6 : exemple simple de structure conditionnelle

```
ldw   1,0,addr(x)
addi  2,0,2
mod   1,1,2
addi  2,0,0
cmp   1,1,2
bne   1,0,? → 6

ldw   2,0,addr(x)
addi  3,0,2
sub   2,2,3
stw   2,0,addr(x)
bsr   0,0,? → 5
ldw   2,0,addr(x)
addi  3,0,1
sub   2,2,3
stw   2,0,addr(x)
hrs   0,0,0
```

Exercice 7 : forme générale du si…alors…sinon…

Pour le `si…alors…` *:*

code(*expression*) *Sauvegarder la position du saut*
code(*instruction*) *Mettre à jour le saut conditionnel*

Pour le `si…alors…sinon…` *:*

code(expression) *Sauvegarder la position du saut*
code(instr. SI) *Mettre à jour le saut conditionnel*
bsr 0,0,? *Sauvegarder la position du saut*
code(instr. SINON) *Mettre à jour le saut inconditionnel*

Exercice 8 : génération de code pour le tant_que

code(*expression*) Sauvegarder la position courante
Sauvegarder la position du saut
code(instructions)
bsr 0, 0, position courante
Mettre à jour la position du saut