Génération de code

La génération du code a pour données le programme compilé et produit comme résultat du texte objet équivalent. La qualité du code objet produit est déterminée par sa vitesse d'exécution et sa taille. Une machine cible possédant un jeu d'instructions riche peut fournir plusieurs façons d'implanter Une bonne connaissance de la machine cible et de son jeu d'instructions est un préalable à la conception d'un générateur de code.

Un générateur de code doit être :

- ✓ Correct.
- ✓ De bonne qualité.
- ✓ Plus rapide.

La machine utilisée dans ce cours est à 8 registres, les opérations arithmétiques ou logiques se font dans l'accumulateur.

a) Chargement

LOAD M	(M) → ACC
LREG i, M	(M) → Reg(i)
LACCR i	Reg(i) → ACC
LRACCR i	(ACC) → Reg(i)

b) **Rangement**

STORE M	(ACC) → (M)
STORE i, M	Reg(i) → (M)

c) Opérations

ADD M	(ACC) + (M) → (ACC)
SUB M	(ACC) - (M) → (ACC)
MULT M	(ACC) *(M) → (ACC)
DIV M	(ACC) / (M) → (ACC)
ABS	|(ACC)| → ACC
CHS	- (ACC) → ACC
B M	Se brancher à M

La gestion des temporaires vise à minimiser le nombre de temporaires utilisés par le programme. La gestion des registres vise à assurer la pleine utilisation des registres et minimise le nombre de chargement de et de rangement de ces registres.

Le contenu de l'accumulateur est initialement nul. On définit dans la suite la procédure ***GetInAcc(x,y)*** qui permet le chargement du premier opérande du quadruplet dans l'accumulateur.

```
Procedure GetInAcc(x,y)
    Begin
     IF ACC = ‘    ‘
       Then     Begin
                        Gen(‘load’,x);  /* load x
                         Acc:=x;
                End;
        Else   IF   ACC = ‘y’     /* 2 ème opérande
             Then   Begin    T := x; x := y; y:= T; /* permutation dans le quadruplé
                        End;
              Else    IF ACC < > ‘x’
              Then  Begin
                                Gen(‘store’, z)
                                Gen(‘Load’, x)
                                Acc:= x;
                         End;
      END.
```

Génération du code pour ‘+’

GetInAcc(quad(Qc,2), quad(Qc,3))

Gen(‘ADD’, quad(Qc,3))

ACC :=quad(Qc,4)

Génération du code pour ‘∗’

GetInAcc(quad(Qc,2), quad(Qc,3))

Gen(‘MULT, quad(Qc,3))

ACC :=quad(Qc,4)

L’appel de GetInAcc(quad(Qc,2), quad(Qc,3)) va réaliser :

Si ACC =’ ‘ Alors Gen(‘load’, quad(Qc,2))

Si ACC = quad(Qc,3) Permutation (+ ou ∗ commutative)

Si ACC < > quad(Qc,2) et quad(Qc,3) alors

Gen(‘Store’,x)

Gen(‘Load’, quad(Qc,2))

ACC := quad(Qc,2)

Génération du code pour ‘-’ binaire

GetInAcc(quad(Qc,2), ‘ ‘)

Gen('SUB', quad(Qc,3)

ACC :=quad(Qc,4) er

Remarque : le '-' binaire n'étant pas commutatif, le chargement du 1 opérande doit se faire si ACC est différent du premier opérande (pas de permutation).

Génération du code pour '-' unaire

GetInAcc(quad(Qc,2), ' ')

Gen('CHS',)

ACC :=quad(Qc,4)

Exemple : Soit l'expression suivante :

a∗((a∗b+c)-(c∗(-b)))

Les quadruplés :

(∗,a,b,t1)

(+,t1,c,t2)

(-,b, ,t3)

(+,c,t3,t4)

(-,t2,t4,t5)

(∗,a,t5,t6)

GetInAcc	**ACC**	**Code**
	Vide	
GetInAcc(a,b)	a	Load a
	t1	Mult b
GetInAcc(t1,c)	t2	Add c
GetInAcc(b, '')		Store t2
		Load b
	t3	CHS
GetInAcc(c,t3) /* permutation des opérandes dans les quadruplés*/	t4	Mult c
GetInAcc(t2, '')	t4	Store t4
	t2	Load t2
	t5	Sub t4
GetInAcc(a,t5) /* permutation des opérandes dans les quadruplés*/	t6	Mult a