

TD 6 - Génération de code MVàP

1 La boucle «tant que»

Soit le code MVàP suivant

```

PUSHI 0
PUSHI 0
JUMP Main
LABEL Main
PUSHI 7
STOREG 0
PUSHI 0
STOREG 1
LABEL Label1
PUSHG 0
PUSHI 1
SUP
JUMPF Label2
PUSHG 0
PUSHI 2
DIV
STOREG 0
PUSHG 1
PUSHI 1
ADD
STOREG 1
JUMP Label1
LABEL Label2
PUSHG 1
WRITE
POP
HALT

```

et le résultat de son assemblage

	Adr	Instruction	
	0	PUSHI	0
	2	PUSHI	0
	4	JUMP	6
	6	PUSHI	7
	8	STOREG	0
	10	PUSHI	0
	12	STOREG	1
	14	PUSHG	0
	16	PUSHI	1
	18	SUP	
	19	JUMPF	37
	21	PUSHG	0
	23	PUSHI	2
	25	DIV	
	26	STOREG	0
	28	PUSHG	1
	30	PUSHI	1
	32	ADD	
	33	STOREG	1
	35	JUMP	14
	37	PUSHG	1
	39	WRITE	
	40	POP	
	41	HALT	

Le début de l'exécution donne la trace suivante :

pc		fp	pile
0	PUSHI	0	0 [] 0
2	PUSHI	0	0 [0] 1
4	JUMP	6	0 [0 0] 2
6	PUSHI	7	0 [0 0] 2
8	STOREG	0	0 [0 0 7] 3
10	PUSHI	0	0 [7 0] 2
12	STOREG	1	0 [7 0 0] 3
14	PUSHG	0	0 [7 0] 2
16	PUSHI	1	0 [7 0 7] 3
18	SUP		0 [7 0 7 1] 4
19	JUMPF	37	0 [7 0 1] 3

Qu 1. Commenter ce début de trace. Compléter.

Les règles de la grammaire et les actions engendrant le code MVàP des expressions arithmétiques étant définies, on peut introduire les premières instructions de notre calculette ainsi :

```
instruction
  : expression finInstruction
  | affectation finInstruction
  | NEWLINE
  ;

affectation : ID '=' expression
  ;

finInstruction : NEWLINE | ';'
  ;
```

On se propose ensuite d'ajouter à notre langage une structure de contrôle de boucle. On pourra par exemple écrire le programme suivant :

```
int i
i = 6
while (i < 10) i = i + 1
println(i)
```

Qu 2. Quel code MVàP doit être produit par le compilateur pour effectuer ce calcul ?

Dans un premier temps, on se contentera de conditions simples avec des opérateurs relationnels et définies par la règle suivante.

```
condition
  : expression ('==' | '<>' | '>' | '>=' | '<' | '<=') expression
  | 'True'
  | 'False'
  ;
```

Qu 3. Ajouter les actions nécessaires pour la génération de code correspondante.

On enrichit alors la définition d'instruction avec la structure de contrôle de boucle :

```
instruction
  : ...
  | 'while' '(' condition ')' instruction
  ;
```

Qu 4. Ajouter les actions nécessaires pour la génération de code correspondante (On dispose d'une fonction qui génère les numéros de Label).

2 Expressions logiques

Qu 5. Donner les règles de la grammaire pour la prise en compte des expressions logiques (négation, conjonction, disjonction). Ajouter les actions nécessaires pour la génération de code.