TD n°0, I45 – Projet
L2 Sciences et Technologies
Université du Sud Toulon-Var
Année universitaire 2005 – 2006

Emmanuel Bruno

Un compilateur pour le langage Algo : présentation du projet

Introduction :

Les programmes informatiques sont écrits en utilisant des langages de programmation mais les ordinateurs interprètent des séquences d'instructions et non de tels programmes. Les programmes doivent donc être « traduits » en une suite d'instructions. Cette traduction peut être automatisée et donc formulée par un programme. C'est ce programme que l'on appelle un *compilateur*. La réalisation d'un tel programme est intéressante à deux niveaux : elle permet de mieux comprendre le fonctionnement « interne » d'un ordinateur et elle demande l'utilisation d'une véritable méthode de conception qui peut (et doit) être utilisée lors de la réalisation de n'importe quel projet informatique.

Objectif :

L'objectif de ce projet est de comprendre comment réaliser un compilateur pour le langage Algo[1] et de programmer au moins la traduction d'expressions arithmétiques simples. Le but est de construire un programme qui, à partir d'une expression voire d'un algorithme (voir ci-dessous) vérifie que celui-ci est conforme à la syntaxe du langage, gère convenablement les variables déclarées puis produit le code machine[2] correspondant.

```
Algorithme somme
Déclaration
      A,B des entiers
Début
      demander-une-valeur-pour A
      demander-une-valeur-pour B
      affiche-la-valeur-de (A+B)
Fin
```

Organisation du projet :

Bien que la réalisation d'un compilateur soit un problème complexe, nous nous attacherons à en étudier les principes fondamentaux : *analyse lexicale, analyse syntaxique* (descendante), simulation d'une *machine RISC*, gestion des *variables*, *génération du code* machine… Ces aspects seront abordés sous la forme de modules dont la structure générale sera décrite en TD et dont certains aspects particuliers seront traités en détails. Ces modules seront ensuite implémentés lors des séances de TP.

Le détail du projet :

L'objectif est d'arriver dans un premier temps à traduire une expression arithmétique ne comportant que des constantes en code machine exécutable (par extension un algorithme complet pourra être compilé). Cette traduction est guidée par la structure du texte que l'on analyse.

Ce projet sera décomposé de la façon suivante :

– Partie standard –

1 – Programmer une **Machine RISC** opérationnelle
2 – Réaliser un **analyseur lexical**
3 – Réaliser un **Analyse syntaxique** pour des expressions simples
Si vous êtes en avance faire le 5
4 – **Générer le code** pour les expressions simples constantes
5 – Construire de l'**arbre syntaxique** des expressions

– Partie avancée –

6 – Gérer les variables, construire la **table des symboles**
7 – Générer **code pour les expressions avec variables**
Partie facultative du TP n°4

– Partie très avancée –

8 – Génération du **code pour les instruction**

Remerciement:
Ce projet est inspiré du livre « Compiler Construction », Niklaus Wirth, Addison-Wesley 1996.

1 Langage algorithmique présenté en 1ère année
2 Ce code sera celui d'une machine RISC (Reduced Instruction Set Computer) qui sera étudiée en TD et programmée en TP.

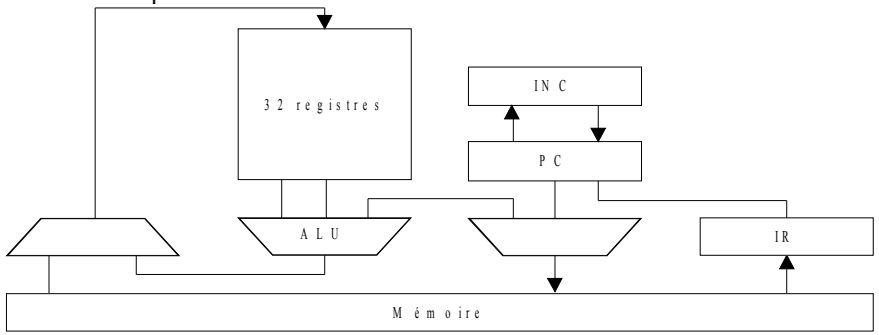
TD n°1, I45 – Projet
L2 Sciences et Technologies
Université du Sud Toulon-Var
Année universitaire 2005 – 2006

Emmanuel Bruno

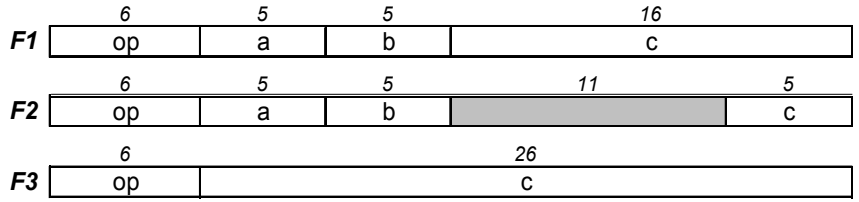
Présentation et étude d'une machine RISC

Description de la machine RISC

Nous définissons une machine « virtuelle » sur laquelle le code correspondant au programme sera exécuté. La machine est composée d'une unité arithmétique (ALU), d'une unité de contrôle et d'un accumulateur. L'unité arithmétique et logique contient 32 registres (notés R0 à R31) de 32 bits chacun. Le registre R0 contient toujours la valeur 0. L'unité de contrôle est composée du registre d'instructions (noté IR), qui contient l'instruction en cours d'exécution, et le compteur ordinal (noté PC) qui contient l'adresse de la prochaine instruction à exécuter. Les instructions de branchement utiliseront implicitement le registre R31 pour stocker l'adresse de retour. La mémoire est composée de mots de 32 bits et elle est adressable en octets (*bytes*), c'est-à-dire que les mots (*words*) ont des adresses multiples de 4.



Il existe trois formats d'instructions « compréhensibles » et exécutables par la machine RISC.



Les instructions de la machine RISC sont données dans le tableau ci-dessous. Les précisions suivantes sont à prendre en compte :

- « R.a » indique « le contenu du registre a »,
- « a » indique « la valeur a »,
- le champ « c » de 16 bits (pour les instructions de format F1 donc) est signé,
- l'instruction SUB effectue une différence et indique « arithmetic overflow » en cas de dépassement de capacité, c'est-à-dire si $-2^{-31} \leq x-y \leq 2^{31}$,
- l'instruction CMP effectue également une différence mais, en cas de dépassement de capacité, indique un résultat faux mais avec un signe correct,
- les instructions LSH et LSHI effectuent des décalages logiques, c'est-à-dire sans prendre en compte le bit de signe,
- les instructions ASH et ASHI effectuent des décalages arithmétiques, c'est-à-dire en tenant compte du bit de signe,
- dans le cas de LSH, LSHI (décalage logique) et de ASH, ASHI (décalage arithmétique), une valeur positive (pour R.c ou c) indique un décalage vers la gauche, une valeur négative indique un décalage vers la droite.

Code	*Instruction*	*Paramètres*	*Fonction*
00	ADD	a, b, c (F2)	R.a := R.b + R.c
01	SUB	a, b, c (F2)	R.a := R.b - R.c
02	MUL	a, b, c (F2)	R.a := R.b * R.c
03	DIV	a, b, c (F2)	R.a := R.b DIV R.c
04	MOD	a, b, c (F2)	R.a := R.b MOD R.c
05	CMP	a, b, c (F2)	R.a := R.b - R.c
06	CHK	a, c (F2)	$0 \leq$ R.a $\leq$ R.c
07	AND	a, b, c (F2)	R.a :=R.b & R.c
08	BIC	a, b, c (F2)	R.a := R.b & ~R.c
09	OR	a, b, c (F2)	R.a := R.b OR R.c

10	XOR	a, b, c (F2)	R.a := R.b XOR R.c
11	LSH	a, b, c (F2)	R.a := LSH(R.b, R.c)
12	ASH	a, b, c (F2)	R.a := ASH(R.b, R.c)
14	ADDI	a, b, c (F1)	R.a := R.b + c
15	SUBI	a, b, c (F1)	R.a := R.b - c
16	MULI	a, b, c (F1)	R.a := R.b * c
17	DIVI	a, b, c (F1)	R.a := R.b DIV c
18	MODI	a, b, c (F1)	R.a := R.b MOD c
19	CMPI	a, b, c (F1)	R.a := R.b – c
20	CHKI	a, c (F1)	0 ≤ R.a ≤ c
21	ANDI	a, b, c (F1)	R.a :=R.b & c
22	BICI	a, b, c (F1)	R.a := R.b & ~c
23	ORI	a, b, c (F1)	R.a := R.b OR c
24	XORI	a, b, c (F1)	R.a := R.b XOR c
25	LSHI	a, b, c (F1)	R.a := LSH(R.b, c)
26	ASHI	a, b, c (F1)	R.a := ASH(R.b, c)
27	LDW	a, b, c (F1)	R.a := Mem[R.b + c] (cword)
28	LDB	a, b, c (F1)	R.a := Mem[R.b + c] (c byte)
29	POP	a, b, c (F1)	R.a := Mem[R.b] ; R.b := R.b + c (pop stack)
30	STW	a, b, c (F1)	Mem[R.b + c] := R.a (c word)
31	STB	a, b, c (F1)	Mem[R.b + c] := R.a (c byte)
32	PSH	a, b, c (F1)	R.b := R.b – c ; Mem[R.b] := R.a (push stack)
33	BEQ	a, c (F1)	Branch to c if R.a = 0
34	BNE	a, c (F1)	Branch to c if R.a ≠ 0
35	BLT	a, c (F1)	Branch to c if R.a < 0
36	BGE	a, c (F1)	Branch to c if R.a ≥ 0
37	BGT	a, c (F1)	Branch to c if R.a > 0
38	BLE	a, c (F1)	Branch to c if R.a ≤ 0
39	BSR	a, c (F1)	Save PC in R31, then branch to c (PC-relative)
43	JSR	c (F3)	Save PC in R31, then jump to c (absolute)
13	RET	c (F2)	Jump to adress in R.c (absolute)
41	RD	a (F1)	Read into R.a
42	WRD	a (F1)	Write from R.a
40	HRS	c (F1)	End program (return value)

Exercice 1

Ecrire le programme qui évalue les expressions suivantes et qui stocke le résultat dans R3 : (3+4) et (3+4*2).

Exercice 2

Ecrire le programme qui écrit dans R3 la valeur de min(R1,R2).

Exercice 3

Exécuter le programme suivant à partir de l'adresse *100*.

Adresse Mémoire	*Valeur stockée*
100	00111000001000000000000000000011
104	00111000001000000000000000001010
108	01111000001000000000001111110100
112	01111000001000000000001111110000
116	01101100001000000000001111110100
120	01101100001000000000001111110000
124	00000100001000010000000000000010
128	00111000011000110000000000000001
132	10010100001000001000000000000010
136	01111000001000000000001111101000
140	01111000011000000000001111101100
144	10100000000000000000000000000001
...	
1000	00000000000000000000000000000000
1004	00000000000000000000000000000000
1008	00000000000000000000000000000011
1012	00000000000000000000000000001010

Que fait ce programme ?

Exercice 4

Proposer un algorithme général pour le décodage des instructions qui permette de récupérer dans les variables OP, A, B et C le code opération et les trois paramètres a, b et c des instructions lues.

Exercice 5

Ecrire avec le langage *Algo* le programme qui calcule le PGCD de deux nombres.

Ecrire ensuite le même programme en code machine.

TD n°1, I45 – Projet
L2 Sciences et Technologies
Université du Sud Toulon-Var
Année universitaire 2005 – 2006

Emmanuel Bruno

Présentation et étude d'une machine RISC : correction

Exercice 1

```
ADDI (1, 0, 3)
ADDI (2, 0, 4)
ADD (3, 1, 2)
HRS (0, 0, 1)
```

```
ADDI (1, 0, 3)
ADDI (2, 0, 4)
ADDI (4, 0, 2)
MUL (3, 2, 4)
ADD (3, 3, 1)
HRS (0, 0, 1)
```

Exercice 2

```
CMP (3, 1, 2)
BLE (3, 0, 2)
ADD (3, 0, 2)
HRS (0, 0, 1)
ADD (3, 0, 1)
HRS (0, 0, 1)
```

Exercice 3

Adresse Mémoire	*Valeur stockée*
100	ADDI (1, 0, 3)
104	ADDI (2, 0, 10)
108	STW (2, 0, 1012)
112	STW (1, 0, 1008)
116	LDW (1, 0, 1012)
120	LDW (1, 0, 1008)
124	SUB (1, 1, 2)
128	ADDI (3, 3, 1)
132	BGT (1, 0, -2)
136	STW (1, 0, 1000)
140	STW (3, 0, 1004)
144	HRS (0, 0, 1)
...	
1000	Valeur finale du compteur
1004	Nombre d'itérations
1008	Pas (ici, 3)
1012	Valeur initiale du compteur (ici, 10)

Exercice 4

```
Algorithme decode
Déclaration
     inst, opc, a, b, c des entiers
Début
     opc = inst >> 26
     a = (inst << 6) >> 27
     b = (inst <<11) >> 27

     si (opc ≥ JSR)
           c = (inst << 6) >> 6
     sinon
           c = (inst << 16) >> 16
           si (opc < ADDI)
                 c = c mod 32
           sinon
                 si (c ≥ 32768)
                       c = -(c - 32768)
                 finsi
           finsi
     finsi
Fin
```

Ou bien :

```
Algorithme decode
Déclaration
```

```
     inst, opc, a, b, c des entiers
Début
     opc = inst div 2^26 mod 2^6
     a = inst div 2^21 mod 2^5
     b = inst div 2^16 mod 2^5

     si (opc ≥ JSR)
           c = inst mod 2^6
     sinon
           c = inst mod 2^16
           si (opc < ADDI)
                 c = c mod 2^5
           sinon
                 si (c ≥ 2^15)
                       c = -(c - 2^15)
                 finsi
           finsi
     finsi
Fin
```

Exercice 5

```
Algorithme PGCD
Déclaration
     a, b, r, pgcd des entiers
Début
     si (a < b)
           r = a
           a = b
           b = r
     finsi
     tant que (b > 0)
           r = a mod b
           a = b
           b = r
     fin tq
     pgcd = a
Fin
```

Ecrire ensuite le même programme en code machine :

```
// chargement des données
ADDI (1, 0, 4568)
ADDI (2, 0, 768)

// inversion des données si nécessaire
CMP (3, 1, 2)
BGT (3, 0, 4)
ADDI (4, 1, 0)
ADD (1, 2, 0)
ADD (2, 4, 0)

// test de la boucle : inverse du test du while !!!
BLE (2, 0, 5)

// corps de la boucle
MOD (3, 1, 2)
ADD (1, 0, 2)
ADD (2, 0, 3)

// retour au test
BSR (0, 0, 4)

// fin
HRS (0, 0, 1)
```