Incrément 3

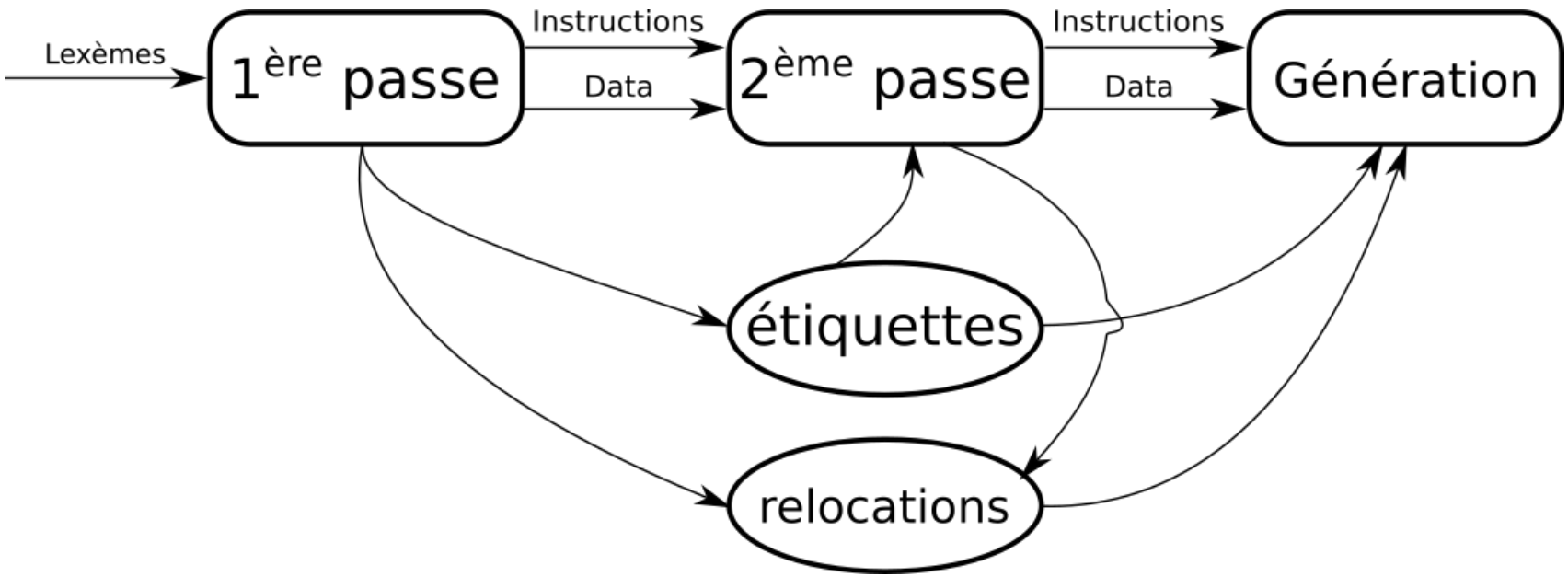
13 novembre 2018

Introduction

- Objectif du livrable
  - Décodage des opérandes dans les instructions
  - Calcul des adresses d'instructions et des étiquettes **(livrable 2)**
  - Table des relocations

Analyse grammaticale

Deux passes



Première passe

Dictionnaire d'instructions

- registre : **Reg** $\longrightarrow$ \$10
- immediat : **Imm** $\longrightarrow$ ADDI \$t1, \$t2, 0x300
- shift amount : **sa** $\longrightarrow$ ROTR \$t1, \$t2, 5
- offset(base) : **Bas** $\longrightarrow$ Lw \$t1 -0x200(\$t2)
- relatif : **Rel** $\longrightarrow$ BNE \$t1, \$t2, 40
- absolu aligné : **Abs** $\longrightarrow$ J 0x4000

Ajouter dans le dictionnaire d'instructions ces types d'informations.
Ex :

```
ADDI I 3 Reg Reg Imm
Lw   I 2 Reg Bas
```

Première passe

Exemple

Exemple de programme :

-	*.text*
0x0	*ADDI $1, $1, 5* # $1 = 0 $2 = 20
0x4	*BEQ $1, $2, 16*
0x8	*NOP*
0xC	*J 0x0*
0x10	*NOP*
-	*.data*
0x0	*.space 0x4000*
0x4000	*etiquette3 : .word 0xFF*
-	*.text*
0x14	*Lw $t1, 10($3)* # Charge la valeur à l'indice 10 du tableau pointé

Première passe

Détection des types

Automate à états finis pour détecter les types suivants :

- nombre entier
- registre
- offset(base)
- symbole (**seconde passe**)

But du décodage :

- Vérifier que chaque opérande dans les instructions est conforme avec ce qui est attendu (instruction ou directive)

Seconde passe

Association entre définition et assembleur

DEF	Vérification	Assembleur
Registre (5 bits non signés entier [0,31])	Erreur si $\notin$ [0,31]	Registre
Immediat (16 bits signé)	Erreur si $\notin$ [SHRT_MIN, SHRT_MAX]	Entier signé
Sa (shift amount) (entier 5 bits non signé)	Erreur si $\notin$ [0,31]	Entier non signé
Offset (Base) (Offset : 16 bits signés, Base : registre)	Base = registre ; Offset dans [SHRT_MIN, SHRT_MAX]	Entier signé et registre
Relatif (16 bits représentant 18bits signés)	Erreur si pas dans 18bits et si pas divisible par 4	Entier signé
Absolu (26 bits non signé représentant 28bits signé)	Erreur si pas dans 28bits ou si pas divisible par 4	Entier signé

Implantation

```
union operande{
    unsigned int reg_sa_base_abs;
    int offset_rel_imm;
    char* symbole;
};
```

ajouter un tableau d'opérandes dans la structure des instructions.

Gestion des étiquettes → nécessité de la relocation

Exemple de programme avec des étiquettes :

-	*.text*
-	*etiquette1 :*
0x0	*ADDI $1, $1, etiquette3*
0x4	*BNE $1, $2, etiquette1*
0x8	*NOP*
0xC	*J etiquette2*
0x10	*NOP*
-	*.data*
0x0	*.space 0x4000*
0x4000	*etiquette3 : .word etiquette2*
-	*.text*
0x14	*etiquette2 : Lw $t1, etiquette3*

Remarque : Symboles non définis ? Ok pour relatif mais pour absolu ? Relocation ?

Table de relocation

- On stocke le décalage du symbole
- Dès que l'on connaît l'adresse effective des sections, on recalcule l'opérande
- 2 tables de relocations (.text et .data)
- Structure de données d'une relocation :
  - ***char**** *nom de la section*
  - ***unsigned int*** *adresse relative du code (instruction ou data) à reloger*
  - ***enum*** *type{R_MIPS_32=2, R_MIPS_26=4, R_MIPS_HI16=5, R_MIPS_LO16=6}*
  - ***Symbol**** *pointeur vers le symbole de l'opérande dans la table des symobles*

Relocation

Les symboles sont traités dans le cas :

- **Immediat**, on tronque l'adresse du symbole à 16 bits (poids faibles) et on génère une entrée de type R_MIPS_LO16 qui pointe vers le symbole
- **Absolu**, on tronque l'adresse du symbole à 28 bits (poids faibles) et on génère une entrée de type R_MIPS_26 qui pointe vers le symbole
- **.word**, on stocke l'adresse du symbole sur 32 bits, on génère une entrée de type R_MIPS_32 qui pointe vers le symbole
- **offset(base)**, impossible, car pseudo-instruction et doit être remplacée

Implantation

Implantation ?

Cas du offset(base)

Relocation

Première passe on modifie le programme original :

Lw \$t1, etiqu	$\longrightarrow$	LUI \$at, etiqu$_{poidsforts>>16}$	R_MIPS_HI_16
		Lw \$t1, etiqu$_{poidsfaibles}$(\$at)	R_MIPS_LO_16

Deuxième passe on calcule la valeur des opérandes :

valeur du symbole (32bits)	0xABCD	FFFF
association	\$at	offset
relocation	R_MIPS_HI_16	R_MIPS_LO_16

Attention si l'offset < 0 (comme c'est le cas ici) il faut ajouter 1 à \$at pour que la somme soit juste lors de l'exécution.

Cas des symboles non définis

Relocation

Que ce passe-t'il si un symbole n'est défini nul part dans le fichier ?
$\rightarrow$ Il peut être défini dans un autre fichier et être résolu lors de l'édition de lien !

Dans ce cas :

1. Le symbole doit être ajouté dans la table des symboles.
2. un champ `undefined` dans la structure du symbole sera mis à vrai.
3. la relocation se comportera comme d'habitude (mais avec un zéro comme valeur de addend).
4. lors de la génération, ce symbole aura une portée globale.

Variable globale

Relocation

Les variables globales c'est le mal absolu. Mais c'est quand même pratique dans quelques cas. Par exemple, la table des symboles pourrait être une variable globale. Elle est unique est accédée par beaucoup de fonctions.

symbol.h	*main.c*	*symbol.h*

`Table sym;`	`#include <symbol.h>`
	`extern sym = NULL;`
	`void main(){`
	`...`
	`}`
		`#include <symbol.h>`
		`void fonction(){`
		`extern sym;`
		`...`
		`}`

déclaration de sym	définition de sym (espace mémoire du main)	utilisation de sym