Compilation et Analyse Lexicale
TD 3 - Correction

Licence 3 Informatique (2022-2023)

Jairo Cugliari, Guillaume Metzler
Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2

jairo.cugliari@univ-lyon2.fr guillaume.metzler@univ-lyon2.fr

Exercice 1

Voici une table pour vous aider à repérer les valeurs :

Type	Form	Operand value	Name
Immediate	\$*Imm*	*Imm*	Immediate
Register	r_a	$R[r_a]$	Register
Memory	*Imm*	$M[Imm]$	Absolute
Memory	(r_a)	$M[R[r_a]]$	Indirect
Memory	$Imm(r_b)$	$M[Imm + R[r_b]]$	Base + displacement
Memory	(r_b, r_i)	$M[R[r_b] + R[r_i]]$	Indexed
Memory	$Imm(r_b, r_i)$	$M[Imm + R[r_b] + R[r_i]]$	Indexed
Memory	$(, r_i, s)$	$M[R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(, r_i, s)$	$M[Imm + R[r_i] \cdot s]$	Scaled indexed
Memory	(r_b, r_i, s)	$M[R[r_b] + R[r_i] \cdot s]$	Scaled indexed
Memory	$Imm(r_b, r_i, s)$	$M[Imm + R[r_b] + R[r_i] \cdot s]$	Scaled indexed

La notation [ ] fait référence à un indice sur la table R (ensemble de registres) ou M (tableau de mémoire). Un registre générique est noté par r_a

Compléter la table ci-dessous avec les valeurs de chaque opérande.

Opérandes	Valeur	Commentaire
`%rax`	`0x100`	registre
`0x104`	`0xAB`	valeur absolue
`$0x108`	`0x108`	valeur immédiate
`(%rax)`	`0xFF`	adresse `0x100`
`4(%rax)`	`0xAB`	adresse `0x104`
`9(%rax,%rdx)`	`0x11`	adresse `0x10C`
`260(%rcx,%rdx)`	`0x13`	adresse `0x108`
`0xFC(,%rcx,4)`	`0xFF`	adresse `0x100`
`(%rax,%rdx,4)`	`0x11`	adresse `0x10C`

Table 1: Valeurs

Exercice 2

Nous devons identifier la taille la plus petite entre la source et la destination. Puis utiliser le bon identifiant de taille. Rappelez-vous que

b	1 octet	8 bits
w	2 octets	16 bits
l	4 octets	32 bits
q	8 octets	64 bits

1. `movl %eax, (%rsp)`
2. `movw (%rax), %dx`
3. `movb $0xFF, %bl`
4. `movb (%rsp,%rdx,4), %dl`
5. `movq (%rdx), \rax`
6. `movw %dx, (%rax)`

Exercice 3

Une fois le décodage repéré, vous pouvez écrire la fonction `decode1.c`. Il suffit alors de faire une compilation partielle, en s'arrêtant avant le point d'assemblage avec le drapeau `-S`. Nous rajoutons le drapeau `-Og` pour définir l'optimisation au niveau "debug", ce qui rend plus lisible le code assembleur.

```
gcc -S -Og decode1.c
```

Vous trouverez ci-dessous un code un peu plus complet, avec un exemple d'utilisation, qui nous permettra de vérifier que les opérations sont bien celles qu'on pense faire lors du décodage.

```c
#include <stdio.h>

void decode1(long *xp, long *yp, long *zp)
{
  long x = *xp;
  long y = *yp;
  long z = *zp;

  *yp = x;
  *zp = y;
  *xp = z;
}

void main() {
    long x = 2;
    long y = 3;
    long z = 10;

    printf("--- AVANT ---\n");
    printf("\%ld \%ld \%ld\n", x, y, z);

    decode1(&x, &y, &z);

    printf("--- APRES ---\n");
    printf("\%ld \%ld \%ld\n", x, y, z);
}
```

Exercice 4