TD 3 Compilation – Premier contact avec la MV

L3 INFO, Univ Lumière Lyon 2

2022 – 2023

Les exercices de ce TD ainsi que la plupart des images sont prises du livre *Computer Systems - A Programmer's Perspective, 3er ed.* de Randal E. Bryant et David R. O'Hallaron.

Nous avons parlé de registres du CPU en CM. Voici une table qui sera d'utilité, montrant les accès et la notation habituelle.

Figure 3.2 **Integer registers.** The low-order portions of all 16 registers can be accessed as byte, word (16-bit), double word (32-bit), and quad word (64-bit) quantities.

Exercice 1 Supposons que les valeurs suivantes soient stockées aux adresses de mémoire et aux registres indiqués (table à gauche). Compléter la table à droite avec les valeurs de chaque opérande.

Adresse	Valeur
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x11

Registre	Valeur
%rax	0x100
%rcx	0x1
%rdx	0x3

Opérandes	Valeur
%rax	
0x104	
$0x108	
(%rax)	
4(%rax)	
9(%rax,%rdx)	
260(%rcx,%rdx)	
0xFC(,%rcx,4)	
(%rax,%rdx,4)	

TABLE 1 – État de la mémoire et registres (à gauche), valeurs à compléter (à droite).

Exercice 2 Pour chacune des lignes de langage assembleur suivantes, déterminez le suffixe d'instruction approprié en fonction des opérandes. Par exemple, `mov` peut être réécrit en tant que `movb` (byte), `movw` (word, 2b), `movl` (double word, 4b), ou `movq` (quad word, 8b).

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

Exercice 3 Vous disposez des informations partielles sur une fonction avec un prototype

```
void decode1(long *xp, long *yp, long *zp);
```

Elle produit le code assembleur qui suit

```
  void decode1(long *xp, long *yp, long *zp)
  xp in %rdi, yp in %rsi, zp in %rdx
decode1:
  movq (%rdi), %r8
  movq (%rsi), %rcx
  movq (%rdx), %rax
  movq %r8, (%rsi)
  movq %rcx, (%rdx)
  movq %rax, (%rdi)
  ret
```

Les paramètres `xp`, `yp` et `zp` sont stockés dans les registres `%rdi`, `%rsi` et `%rdx`, respectivement. Écrivez la fonction `decode1` en `C` qui aura un effet équivalent au code assembleur présenté.

Exercice 4

1. Créer un fichier source nommé `multstore.c` avec le contenu suivant

```
long mult2(long, long);

void multstore(long x, long y, long *dest) {
  long t = mult2(x, y);
  *dest = t;
}
```

2. Obtenez la version assembleur de multstore.c.
3. Identifiez les instructions assembleur qui correspondent au code C.