Compilation et Analyse Lexicale
TD 5 - 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

On considère la fonction suivante

```
long fact_do(long n)
{
    long result = 1;
    do {
        result *= n;
        n = n-1;
    } while (n > 1);
    return result;
}
```

1. Obtenir le code assembleur de la fonction `fact_do`.

```
fact_do:
    movl    $1, %eax        # resultat = 1
.L2:                        # boucle:
    imulq   %rdi, %rax      # result *= n
    subq    $1, %rdi        # r duire n
    cmpq    $1, %rdi        # test n=1
    jg      .L2             # sauter si >
    rep; ret                # retour
```

2. Identifier les lignes de code C avec le fonctionnement sur assembleur.

   Vous trouverez de commentaires qui vous permettront d'identifier les lignes du code C. Vous pouvez consulter la table d'instructions pour celles que vous ne connaissez pas (e.g. `imulq`), par exemple ici (https://cs.brown.edu/courses/cs033/docs/guides/x64_cheatsheet.pdf).

3. Créez un programme pour tester la fonction.

4. Exécutez le code dans `gbd` pour examiner les opérations réalisées sur les registres.

Exercice 2

On considère la fonction suivante

```
10 short dw_loop(short x) {
11     short y = x/9;
12     short *p = &x;
13     short n = 4*x;
14     do {
15         x += y;
16         (*p) += 5;
17         n -= 2;
18     } while (n > 0);
19     return x;
20 }
```

1. Quels registres sont utilisés pour contenir les valeurs `x`, `y` et `n` du programme ?

   Voir le code commencté ci-dessous. Bien que le paramètre `x` soit passé à la fonction dans le registre `%rdi`, nous pouvons voir que le registre n'est jamais référencé une fois que la boucle est entrée. Au lieu de cela, nous pouvons voir que les registres `%rbx`, `%rcx`, et `%rdx` sont initialisés dans les lignes 11-13 (cf code ci-dessus) à `x`, `x/9`, et `4*x`. Nous pouvons donc conclure que ces registres contiennent les variables du programme.

2. Comment le compilateur a-t-il éliminé le besoin d'une variable pointeur `p` et le déréférencement de pointeur impliqué par l'expression `(*p)+=5` ?

   Le compilateur détermine que le pointeur `p` pointe toujours sur `x`, et donc l'expression `(*p)+=5` incrémente simplement `x`. Il combine cet incrément de 5 avec l'incrément de y, via l'instruction `leaq` de la ligne 9 du code ci-dessous.

3. Ajoutez des annotations au code d'assemblage décrivant le fonctionnement du programme.

```
1     short dw_loop(short x)
2      x initi  dans %rdi
3 dw_loop:
4     movq    %rdi, %rbx                  Copier x to %rbx
5     movq    %rdi, %rcx
6     idivq   $9, %rcx                    Calculer y = x/9
7     leaq    (,%rdi,4), %rdx             Calculer n = 4*x
8 .L2:                                 boucle
```

```
9     leaq    5(%rbx,%rcx), %rcx          Calculer y += x + 5
10    subq    $2, %rdx                    R'eduire n de 2
11    testq   %rdx, %rdx                  Test n
12    jg      .L2                         Si > 0, aller it rer
13    rep; ret                            Retourner
```