Hardware Software Interface

TP3 : Code machine et assembleur

Alexis DALEY

Question 1. Générer le code assembleur d'un fichier implémentant la fonction swap vue en cours (gcc –O1 –S file.c) et étudier le. Puis générer un code objet et étudier le fichier avec objdump –d file.o ou gdb.

Question 2. Expliquer les erreurs :

1) movb $0xF, (%ebx)
2) movl %rax, (%rsp)
3) movw (%rax), 4(%rsp)
4) movb %al, %sl
5) movl %eax, 0x123
6) movl %eax, %dx
7) movb %si, 8(%rbp)

Reverse engineering club ☺

Question 3. Écrire le code C pour la fonction :

```
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
```

Question 4. Écrire le code C pour la fonction :

```
long decode2(long x, long y, long z)
/* x in %rdi, y in %rsi, z in %rdx , and the return value is in %rax*/
decode2:
        subq    %rsi, %rdx
        movq    %rdx, %rax
        salq    $63, %rax
        sarq    $63, %rax
        imulq   %rdi, %rdx
        xorq    %rdx, %rax
        ret
```

Question 5. Boucle

Pour le code C :

```
long dx_loop(long x) {
        long y = x*x;
        long *p = &x;
        long n = 2*x;
        do {
                x +=y;
                (*p)++;
                n--;
                } while (n>0);
                return x;
        }
```

gcc génère le code assembleur suivant :

```
        //x initially in %rdi
dx_loop:
        movq    %rdi, %rax
        movq    %rdi, %rcx
        imulq   %rdi, %rcx
        leaq    (%rdi, %rdi), %rdx
.L2:
        leaq    1(%rcx,%rax), %rax
        subq    $1, %rdx
        testq   %rdx, %rdx
        jg      .L2
        rep ret
```

1. Quels registres sont utilisés pour les variables x, y et n ?
2. Comment le compilateur a traité le pointer p ?

Question 6. Switch

Voici un code C non complet :

```
        void switch_omitted (long x, long *dest) {
                long val = 0;
                switch (x) {
                        // body with cases
                }
                *dest = val;
        }
```

Et gcc génère le code assembleur suivant :

```
        //x in %rdi
        switch_omitted :
                addq    $1, %rdi
                cmpq    $8, %rdi
                ja      .L2
```

```
        jmp     *.L4, (,%rdi,8)
```

avec le jump table :

```
.L4:
        .quad   .L9
        .quad   .L5
        .quad   .L6
        .quad   .L7
        .quad   .L2
        .quad   .L7
        .quad   .L8
        .quad   .L2
        .quad   .L5
```

En utilisant ces informations, répondre aux questions suivantes :

1. Quelles sont les valeurs de cas dans le switch ?
2. Quels cas ont des multiples labels dans le code C ?

Question 7. Transfert de données conditionnel (conditionnal move)

Considérer le code suivant :

```
long cread(long *xp) {
return (xp ? *xp : 0);
}
```

Et une implémentation non valide qui essaie d'utiliser un transfert de données conditionnel (cmove) :

```
//invalid implementation of function
// xp in %rdi
cread:
movq    (%rdi), %rax
        testq   %rdi, %rdi
movl    $0, %edx
cmove   %rdx, %rax
        ret
```

Écrire une fonction C cread_alt qui a le même comportement que cread mais qui peut être compilé en utilisant cmove (transfert conditionnel de données).