CC4301 Arquitectura de Computadores
Solución Auxiliar 6

Prof. Aux.: Gaspar Pizarro V.

10 de octubre de 2012

1. Decompilación

Considere el siguiente código en ensamblador para x86, 32 bits, sintaxis AT&T:

```
1  .globl f
2  f:
3      pushl %ebp
4      movl %esp, %ebp
5      pushl %edi
6      pushl %esi
7      pushl %ebx
8      movl 8(%ebp), %edi
9      cmpl $1, %edi
10     je L6
11     testl %edi, %edi
12     jz L6
13     movl $1, %eax
14 .L5:
15     mull    %edi
16     subl    $1, %edi
17     testl   %edi, %edi
18     jg  .L5
19     jmp L7
20 L6:
21     movl $1, %eax
22 L7:
23     popl %ebx
24     popl %esi
25     popl %edi
26     leave
27     ret
```

Transcriba este código a código C, que realicé la misma función.

Solución

```
int f(int a) {
    if (a==1) return 1;
    if (a==0) return 1;
    int b = 1;
    while (a>0) {
        b = a*b;
        b--;
    }
    return b;
}
```

2. P1 Control 2 año 2006

Transcriba el siguente programa en Assembler a C:

```
    .text
.globl p
p:
    pushl %ebp
    movl %esp, %ebp
    pushl %edi
    pushl %esi
    pushl %ebx
    movl 8(%ebp), %esi
    movl 12(%ebp), %edi
    movl 16(%ebp), %ecx
    decl %ecx
    movl $0, %ebx
.L2:
    leal (%ebx, %ecx), %eax #eax = ebx+ecx
    shrl %eax
    movl (%edi, %eax, 4), %edx
    cmpl %esi, %edx
    je .L5
    cmpl %esi, %edx
    jge .L6
    leal 1(%eax), %ebx
    jmp .L8
.L6:
    leal -1(%eax), %ecx
.L8:
    cmpl %ecx, %ebx
    jbe .L2
    movl $-1, %eax
.L5:
    popl %ebx
    popl %esi
    popl %edi
    popl %ebp
    ret
```

Solución

```
int p(int x, int *y, int z){
    z--;
    int res;
    int b = 0;
    do {
        res = b+z;
        res = res/2;
        if (y[res]==x) return res;
        else if (y[res]<x) z--;
        else b = res+1;
    } while (z-b>=0)
    return -1;
}
```

3. P1 Control 2 año 2009

```
1       .globl p
2   p:
3       pushl %ebp
4       movl %esp, %ebp
5       pushl %edi
6       pushl %esi
7       pushl %ebx
8       subl $12, %esp
9       movl 8(%ebp), %esi
10      movl 12(%ebp), %ebx
11      movl -4(%esi,%ebx,4), %edi
12      leal -2(%ebx), %ecx
13      testl %ecx, %ecx
14      jle .L2
15      leal -8(%esi, %ebx, 4), %edx
16  L4:
17      movl (%edx), %eax
18      movl %eax, 4(%edx)
19      subl $4, %edx
20      decl %ecx
21      jne .L4
22  L2:
23      movl %edi, 4(%esi)
24      movl %ebx, 4(%esp)
25      movl %esi, (%esp)
26      call q
27      addl $12, %esp
28      popl %ebx
29      popl %esi
30      popl %edi
31      popl %ebp
32      ret
```

Solución

```
void p(int *a, int b) {
    int c = a[b-1];
    int d = b-2;
    int e = b-2;
    while (d>=0) {
        a[e+1] = a[e];
        e--;
        d--;
    }
    a[1] = c;
    q(a,b);
}
```

En la linea 15 se carga el registro %edx (mapeada a e en el código C) con una posición en memoria, dada por -8(%esi, %ebx, 4). Esto es *(a+(b*4-8)/4[1]) = *(a+b-2) = a[b-2]. Se ve que se usa así en las lineas 17 y 18, donde se derreferencia %edx directamente (no con el metodo base+index*scale+disp tradicional). Entonces se ve que la variable b no es un puntero, sinó que es simplemente un índice.

En las lineas 24 y 25 se ponen números en posiciones arriba de %esp. Como en la instrucción 8 el puntero %esp decrece, entonces se tiene en la pila un espacio vacio entre el último valor relevante de la pila y el lugar apuntado por %esp. Como en 24 y 25 se ponen cosas en la posición apuntada por %esp y 4 bytes mas arriba, lo que se hace en realidad es ocupar los espacios vacíos de la pila, de forma que es lo mismo que hacer operaciones push.

¿Ciclo while o ciclo do-while? Se ve que la etiqueta .L4 es el comienzo de un ciclo. En este ciclo se hace la verificación de condición al final (linea 21), de forma que aparentemente debería ser un ciclo do-while el correspondiente al código C. Sin embargo, en la linea 13 se pregunta por el valor de %ecx (mapeado a d), y la pregunta es la misma que la que se hace en la linea 21, de forma que es imposible que se entre al ciclo sin satisfacer su condición, lo que implica que el ciclo en el código C es un ciclo while.

[1] Ya que se está en arquitectura de 32 bits, entonces moverse 4 bytes es lo mismo que moverse un int en un arreglo. Lo mismo se ve en la linea 19, donde %edx decrece de a 4.