9

Arquitectura del Computador - LCC

Examen Parcial - 15 de noviembre de 2024

Nombre y apellido: Santiago Bussanich

Legajo: B-6488/2

IMPORTANTE: Justifica todas las respuestas e indica de manera clara y detallada los procedimientos utilizados para alcanzar las soluciones. No se permite el uso de calculadoras ni de ningún otro dispositivo electrónico.
DURACIÓN MÁXIMA: 2 hs.

1. (50 puntos) Dado el siguiente código en Assembler x86-64:

```
.data
str1: .asciz "Estoy en foo!\n"
str2: .asciz "La cantidad de argumentos es %d\n"
str3: .asciz "El argumento %d es %s\n"

.text
.global main
main:
        pushq %rbp
        movq %rsp, %rbp
        subq $16, %rsp
        movq %rdi, -8(%rbp)
        movq %rsi, -16(%rbp)
        call foo
        movq -8(%rbp), %rsi
        movq $str2, %rdi
        xorl %eax, %eax
        call printf
        xorl %r12d, %r12d
        movq -16(%rbp), %rbx
bucle:
        incq %r12
        movq $str3, %rdi
        movq %r12, %rdx
        movq %r12, %rsi
        xorq %rax, %rax
        call printf
        cmpq -8(%rbp), %r12
        jbe bucle
        movq %rbp, %rsp
        popq %rbp
        xorl %eax, %eax
        ret

foo:
        pushq %rbp
        movq %rsp, %rbp
        movq $str1, %rdi
        xorq %rax, %rax
        call printf
        movq %rbp, %rsp
        popq %rbp
        xorl %eax, %eax
        ret
```

completo acá

Guardamos rdi que tenía la cant. de argumentos, en %rsi que será el entero a imprimir en printf

aquí %r12 vale 0.
aquí %rbx contiene el valor del puntero argc.

(%rbx, %r12, 8)

(a) Completar las líneas de puntos en las instrucciones para que el programa imprima lo siguiente cuando se compile y ejecute de la forma indicada: `./a.out Hola mundo`

```
Estoy en foo!
La cantidad de argumentos es 3
El argumento 0 es ./a.out
El argumento 1 es Hola
El argumento 2 es mundo
```

(b) Elaborar un diagrama de la pila (*stack*) que muestre su contenido y la evolución de los registros `rsp` y `rbp`. Considerar como valores iniciales `rsp=0x7fffffffe078` y `rbp=0x401190`.

(c) Explicar por qué no es recomendable utilizar la instrucción `loop` para realizar las iteraciones en este código.

(d) Explicar por qué se utilizó el registro `r12` en lugar de `rcx`, aunque este último no estaba en uso. ¿Qué cambios adicionales serían necesarios en el código si, de todos modos, se quisiera utilizar el registro `rcx`?

(e) ¿Cómo podría modificarse el código para imprimir el mensaje `Cantidad de argumentos insuficientes` cuando solo hay un argumento de entrada, es decir, al ejecutarse como `./a.out`?

(f) ¿Por qué el código guarda el registro `rbp` al inicio y al final de las funciones `main` y `foo`? ¿Cuál es el propósito general de este registro?

(g) Explicar para qué se utiliza la instrucción `subq $16, %rsp` en la tercera línea de la función `main`.

(h) Explicar qué hace la instrucción `xorl %r12d, %r12d` y proporciona una instrucción equivalente.

(i) Si quisieras que el mensaje impreso por la función `foo` incluyera el número de argumentos pasados al programa, ¿cómo modificarías el código?

2. (20 puntos) Dado el siguiente código en Assembler x86-64:

```
.data
a: .byte 1, 2, 3, 4, 5
b: .quad 0x1122334455667788
str: .asciz "Hola mundo\n"

.text
.global main
main:
        movq b, %rax            # rax = 0x1122334455667788
        movq $a, %rbx           # rbx = 0x0000000000404028
        movl $4, %ecx           # rcx = 0x4
        movb (%rbx,%rcx,1), %ah # rax = 0x1122334455660588
        movl 0x5(%rbx), %eax    # rax = 0x00000000 55667788
        lea b, %rbx             # rbx = 0x40402d
        movl b(,%rcx,1), %eax   # rax = 0x0000000044332211
        mulb %cl                # rax = 0x0000000044330044
        ret
```

(a) Dibujar el esquema de memoria mostrando el contenido de cada byte, las direcciones de memoria y las etiquetas. Asumir que la etiqueta a corresponde a la dirección `0x404028`.

(b) Indicar el contenido de los registros mencionados en los comentarios después de ejecutar la instrucción correspondiente.

(c) Indicar el valor de retorno de la función después de su ejecución.

3. (30 puntos) Escribir una función en ensamblador x86-64 que reciba un puntero a un arreglo de datos de tipo `double` y un dato tipo `long int` que indique la cantidad de elementos en dicho arreglo. La función debe calcular la suma de los valores en el arreglo y el promedio de los mismos. Luego, debe imprimir estos resultados y retornar el entero 0.

El prototipo de la función en C es la siguiente:

```
int calcular_suma_y_promedio(double *arreglo, long int cantidad);
```

Ejercicio	1	2	3	Total
Puntos	50	20	30	100
Puntos obtenidos	46	16	28	90