Arquitectura del Computador - LCC

Examen Parcial - 12 de noviembre de 2025

Nombre y apellido: _____

Legajo: _____

1. (25 puntos) Dado el siguiente código en Assembler x86-64:

```
.data
str1:   .asciz "Estoy en foo!\n"
str2:   .asciz "%d\n"

.text
foo:
    pushq   %rbp
    movq    $str1, %rdi
    xorl    %eax, %eax
    call    printf
    ret
suma:
    pushq   %rbp
    movq    %rsp, %rbp
    subq    $32, %rsp
    movl    %edi, -4(%rbp)
    movl    %esi, -8(%rbp)
    movl    %edx, -12(%rbp)
    movl    %ecx, -16(%rbp)
    movl    %r8d, -20(%rbp)
    movl    %r9d, -24(%rbp)
    call    foo
    movl    -4(%rbp), %edx
    movl    -8(%rbp), %eax
    addl    %eax, %edx
    movl    -12(%rbp), %eax
    addl    %eax, %edx
    movl    -16(%rbp), %eax
    addl    %edx, %eax
    movq    %rbp, %rsp
    popq    %rbp
    ret

.global main
main:
pushq   %rbp
movq    %rsp, %rbp
    pushq   $8
    pushq   $7
    movl    $6, %r9d
    movl    $5, %r8d
    movl    $4, %ecx
    movl    $3, %edx
    movl    $2, %esi
    movl    $1, %edi
    call    suma
    movq    $str2, %rdi
    movl    %eax, %esi
    xorl    %eax, %eax
```

edi esi r8d r9d
↑ ↑ ↑ ↑
1, 2, 7, 8

→ la suma tiene que dar 18.

7 + 8 + edi + esi = 15 + edi + esi

∴ edi = 2 ó edi = 1
esi = 1 esi = 2

```
call    printf
movl    $0, %eax
ret
```

(a) Completar las líneas de puntos para que el programa imprima el siguiente mensaje, en el cual la segunda línea muestra la suma de los argumentos 1, 2, 7 y 8:

```
Estoy en foo!
La suma es 18
```

(b) Elaborar un diagrama de la pila que muestre su contenido después de ejecutado el prólogo de la función `foo`, así como la evolución de los registros `rsp` y `rbp` hasta alcanzar ese estado. Considerar como valores iniciales `rsp=0x7fffffffeb38` y `rbp=0x7fffffffebd0`.

(c) Explicar por qué es necesaria la instrucción `subq $32, %rsp` en el código anterior y qué ocurre si se omite.

2. (25 puntos) Dado el siguiente código en Assembler x86-64:

```
.data
a: .word -1, -2, 0xff, 0xab
b: .quad 0x12345678
str: .asciz "%d\n"

.text
.global main
main:
    movl a, %eax              # rax = ..........
    movq $a, %rbx             # rbx = ..........
    movl $3, %ecx             # rcx = ..........
    movw (%rbx,%rcx,1), %ax   # rax = ..........
    movb 0x5(%rbx), %al       # rax = ..........
    leaq b, %rbx              # rbx = ..........
    movl (%rbx,%rcx,1), %eax  # rax = ..........
    movl $-1, %r8d            # r8 = ..........
    imull %r8d                # rax = ..........., rdx = ..........
    ret
```

(a) Dibujar el esquema de memoria, mostrando el contenido de cada byte, las direcciones y las etiquetas. Asumir que la etiqueta a corresponde a la dirección `0x404010`.

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

(c) Indicar el valor de retorno que se obtendría al imprimir la variable de entorno en el shell mediante el comando `echo $?`.

3. (30 puntos) Escribir una función en ensamblador x86-64 que recibe un puntero a un arreglo de elementos de tipo `float` y un valor de tipo `int` que indica la cantidad de elementos del arreglo. La función debe calcular la suma y el promedio de los valores del arreglo, imprimir ambos resultados y retornar el entero 0 si la cantidad de elementos es par o 1 si es impar.

El prototipo de la función en C es el siguiente:

```
int funcion(float *arreglo, int cantidad);
```

4. (20 puntos) Indique si las siguientes afirmaciones son verdaderas o falsas. Justifique cada respuesta.

(a) La dirección de memoria `0x7fffffffeb40` está alineada a 16 bytes.

(b) La instrucción `andq $-16, %rsp` permite alinear la pila a 16 bytes.

(c) La instrucción `pushq %rax` equivale a ejecutar la instrucción `movq %rax, -8(%rsp)`.

(d) La instrucción `movss %rax, %xmm0` es inválida.

(e) La instrucción `addps %xmm0, %xmm1` permite sumar tanto vectores de enteros como vectores de flotantes.

Ejercicio	1	2	3	4	Total
Puntos	25	25	30	20	100
Puntos obtenidos					