Estructura de Computadores: Práctica 1 (preguntas)

Patricia Córdoba Hidalgo

Sesión de depuración saludo.s

1.

En el registro %edx se guarda longsaludo, cuyo valor tras ejecutar `mov longsaludo, %edx` es 28 (en decimal) o 0x1c (en hexadecimal). La variable longsaludo representa la longitud de la cadena saludo. Dicha cadena contiene el valor `"Hola a todos!\nHello, World!\n"`.

Para comprobarlo, usamos la orden `list`, y localizamos la línea donde se encuentra la orden `mov longsaludo, %edx`. Colocamos un `breakpoint` en dicha línea y lanzamos la ejecución con `run`. Una vez que se para la depuración, ejecutamos la instrucción `stepi` y con `print %edx` consultamos el valor del registro. También podemos ver su valor con la orden `info registers`.

2.

Con el procedimiento anterior, podemos obtener el valor del registro `%edx` tras ejecutar `mov $saludo, %ecx`, que es 134516888 (en decimal) o 0x8049098 (en hexadecimal). Este valor es a la dirección de memoria de la variable *saludo*.

3.

Eliminamos el símbolo de dato inmediato ($) de la instrucción `mov saludo, %ecx`, ensamblamos y enlazarmos el nuevo programa. Al depurar observamos que `%ecx` tras ejecutar `mov saludo, $ecx` guarda el valor 1634496328 (en decimal) o 0x616c6f48 (en hexadecimal).

La variable $saludo representa el valor de la variable saludo, sin embargo, usando sólo saludo los caracteres del string se transforman en a valor en ASCII en hexadecimal hasta completar una posición de memoria, que son 32 bits.

4.

Cada caracter de la variable *saludo* ocupa un byte, luego, cada uno ocupa una posición de memoria, lo que podemos ver usando `x/32cb &saludo`. Por lo tanto, en total *saludo* ocupa 28 bytes, que corresponde a 28 direcciones de memoria, ya tiene 28 *saludo* tiene caracteres. A continuación, se guarda la variable *longsaludo*, que solo ocupa 1 byte. Lo podemos comprobar con `x/4cb &longsaludo`.

5.

Con las instrucciones `x /1xb &longsaludo` y `x /4xb &longsaludo`, comprobamos que la variable *longsaludo* ocupa una posición de memoria, 0x80490b3. Con la instrucción `x /4xb &longsaludo` vemos que el byte de la primera posición es el menos significativo, luego usa el criterio little-endrian.

6.

La instrucción `mov $1, %ebx` ocupa 5 posiciones de memoria, lo cual se obtiene mediante la opción View-> Machine Code Window. Ahí vemos que dicha instrucción comienza en 0x08048079 y ocupa 5 posiciones más, hasta 0x0804807d.

7.

Si eliminamos la primera instrucción `int 0x80`, el mensaje `"Hola a todos!\nHello, World!\n"` ya no aparece por pantalla, a pesar de que el programa `exited normally`. Si eliminamos la segunda instrucción `int 0x80`, el mensaje `"Hola a todos!\nHello, World!\n"` aparece por pantalla, pero el programa termina con: `Program received signal SIGSEGV, Segmentation fault.`

8.

En el archivo `"/usr/include/asm/unistd_32.h"`, que contiene los números de llamadas al sistema, vemos que el número de llamada al sistema READ es el 3 en la siguiente linea: `#define __NR_read 3`

Sesión de depuración suma.s

1.

El registro `%eax` contiene el valor 37 (0x25 en hexadecimal) justo antes de ejecutar la instrucción `RET`. La comprobación se hizo con un `break point` en la línea 29, donde se encuentra dicha orden, y con la orden `info reg` para comprobar el valor de los registros. Dicho registro contiene el valor 37, ya que éste es la suma de todos los elementos de la lista: 1, 2, 10, 1, 2, 0b10 (2 en decimal) 1, 2, 0x10 (16 en decimal).

`(.-lista)` es número de bytes que ocupa lista, ya que cada entero ocupa 32b, es decir, 4B, y ésta tiene 9 enteros. Por lo tanto, el valor de `(.-lista)` es 36. La variable `(.-lista)/4` representa la longitud de la lista, que es 9. Podemos ver el valor de dicha variable con un `print longlista`.

2.

Al sumar los elementos: 0xffffffff, 0xffffffff, 0xffffffff, se obtiene el valor -3, que en hexadecimal es 0xfffffffd, debido a que, en complemento a 2, 0xffffffff vale -1, y lo estamos sumando 3 veces.

3.

La dirección de memoria asignada a la etiqueta suma y al bucle se obtiene pinchando sobre la etiqueta en el código, y manteniendo el cursor sobre ésta. Obtenemos así que la dirección de suma es 0x8048095 y la del bucle es 0x80480a0.

4.

El registro `%esp` es el puntero de pila, y el registro `%eip` contine la dirección de memoria de la instrucción que va a ejecutarse a continuación.

5.

El registro `%esp` contiene la dirección 0xbffff870 antes de ejecutar `call`, y la dirección 0xbffff86c antes de ejecutar `ret`.

Se diferencian en 4 bits, porque al llamar a la función `call`, la pila usa 4 bits para guardar la dirección de retorno a la función, la necesitamos cuando se ejecute `ret`, por lo que este valor es de gran interés para nosotros..

6.

La orden `call` modifica los siguentes registros: `%eax`, que va guardando el resultado de las sucesivas sumas. `%esp`, que almacena la dirección de retorno en la pila. `%edx`, que actúa como contador. `%eip`, que guarda la dirección ed la próxima instrucción que va a ser ejecutada.

7.

La instrucción `ret` modifica los registros `%esp` y `%eip`. El primero, el puntero de pila, ha de apuntar a la dirección en la que estaba el programa antes de ejecutar la subrutina. El segundo, contiene la próxima instrucción que va a ejecutarse, luego cambia al cambiar de instrucción.

8.

Cuando comienza el programa, el puntero de pila, `%esp`, almacena 0xffffcfa0. Una vez dentro de la subrutina suma, el puntero de pila tiene el valor 0xffffcf98, y conserva dicho valor dentro del bucle. Después de la instrucción `pop %edx`, el `%esp` vale 0xffffcf9c.

9.

La instrucción `mov $0, %edx` ocupa 5 posiciones de memoria, lo cual se obtiene mediante la opción View-> Machine Code Window. La instrucción `inc %edx` ocupa 1 posición de memoria.

10.

Si se elimina la instrucción `ret`, el programa finaliza con el mensaje `Program received signal SIGSEGV, Segmentation fault. 0x080480a9 in ?? ()`. Al no ejecutar la instrucción `ret`, el programa no podría volver al programa principal que llamó a esta subrutina.

Cuestiones sobre suma64unsigned.s

1.

El valor máximo que puede llegar a sumarse es 1fffffffe, luego se necesitan 5 bits adicionales para almacenar dicho resultado, un total de 37 bits. Dicho valor es el resultado de sumar 32 veces 0xffffffff, donde se producen 31 acarreos.

2.

Para que el resultado obtenido sea 2^32, cada elemento debe valer 2^32 / 32 = 2^32 / 2^5 = 2^27, que expresado en hexadecimal es 0x08000000. Tras ejecutar sumasinsigno.s con todos los elementos inicializados con 0x080000000, el resultado al hacer el volcado de memoria da, ciertamente 0x00000000 0x00000001 (en little-endrian), lo que equivale a 2^32. El acarreo se produce al llamar por última vez a la función suma, al sumar el último elemento de la lista 4 (L4) al acumulador `%eax`, que almacenaba el valor 0xf8000000 tras sumar los elementos anteriores.

3.

La suma de los valores 0x10000000, 0x20000000, 0x40000000, 0x80000000 repetidos cíclicamente tiene como resultado 0x80000000 0x00000007 (little-endrian), o lo que es lo mismo, 780000000 en hexadecimal. Cada vez que el programa llama a la función suma se produce un acarreo, ya que en cada lista estamos sumando 1e0000000.

Código de sumasinsigno.s

```
1  .section .data             #Listas de números que sumará el programa (y
       otros ejemplos)
2  #L1:         .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0
       xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
3  #L2:         .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0
       xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
4  #L3:         .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0
       xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
5  #L4:         .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0
       xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
6
7  #L1:         .int 0x08000000, 0x08000000, 0x08000000, 0x08000000, 0
       x08000000, 0x08000000, 0x08000000, 0x08000000
8  #L2:         .int 0x08000000, 0x08000000, 0x08000000, 0x08000000, 0
       x08000000, 0x08000000, 0x08000000, 0x08000000
9  #L3:         .int 0x08000000, 0x08000000, 0x08000000, 0x08000000, 0
       x08000000, 0x08000000, 0x08000000, 0x08000000
```

```
10 #L4:        .int 0x08000000, 0x08000000, 0x08000000, 0x08000000, 0
       x08000000, 0x08000000, 0x08000000, 0x08000000
11
12 #L1:        .int 1, 2, 3, 4, 1, 2, 3, 4
13 #L2:        .int 1, 2, 3, 4, 1, 2, 3, 4
14 #L3:        .int 1, 2, 3, 4, 1, 2, 3, 4
15 #L4:        .int 1, 2, 3, 4, 1, 2, 3, 4
16
17 L1:     .int 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000
       , 0x20000000, 0x40000000, 0x80000000
18 L2:     .int 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000
       , 0x20000000, 0x40000000, 0x80000000
19 L3:     .int 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000
       , 0x20000000, 0x40000000, 0x80000000
20 L4:     .int 0x10000000, 0x20000000, 0x40000000, 0x80000000, 0x10000000
       , 0x20000000, 0x40000000, 0x8000000
21
22 longlista:  .int 8                  #Longitud de la lista
23 resultado:  .quad -1                #Resultado de la suma (variable de
       64 bits)
24
25 .section .text
26 _start: .global _start
27
28     mov $L1, %ebx                   #En el registro %ebx se guarda la
           dirección de memoria donde comienza la lista L1
29     mov longlista, %ecx             #En el registro %ecx se guarda la
           longitud de la lista
30     mov $0, %eax                    #Inicializa el registro %eax con el
            valor 0
31     mov $0, %edx                    #Inicializa el registro %edx con el
            valor 0
32     call suma                       #Llama a la subrutina suma
33     mov $L2, %ebx                   #En el registro %ebx se guarda la
           dirección de memoria donde comienza la lista L2
34     call suma                       #Llama a la subrutina suma
35     mov $L3, %ebx                   #En el registro %ebx se guarda la
           dirección de memoria donde comienza la lista L3
36     call suma                       #Llama a la subrutina suma
37     mov $L4, %ebx                   #En el registro %ebx se guarda la
           dirección de memoria donde comienza la lista L1
38     call suma                       #Llama a la subrutina suma
```

```
39      mov %eax, resultado             #Guarda en resultado el valor
            guardado en %eax
40      mov %edx, resultado+4           #Guarda en resultado+4 el valor
            guardado en %edx
41
42      mov $1, %eax
43      mov $0, %ebx
44      int $0x80                       #Traduce lo que hay en %eax a una
            instrucción para finalizar el programa correctamente
45
46  suma:
47      push %esi
48      mov $0, %esi                    #Inicializa %esi a 0
49
50  bucle:
51      add (%ebx,%esi,4), %eax         #Suma lo que hay en la posición con
             el valor de %esi de la lista a %eax y guarda el resultado en %
            eax
52      adc $0, %edx                    #Suma el acarreo a %edx y lo guarda
             en %edx
53      inc       %esi                  #Incrementa %esi (registra la
            posición de la lista que voy a sumar)
54      cmp  %esi,%ecx                  #Compara el valor de %esi con la
            longitud de la lista
55      jne bucle                       #Si %esi es menor que la longitud
            de la lista, sigue sumando
56
57      pop %esi
58      ret                             #Vuelve a la dirección de retorno
            tras la subrutina
```

Cuestiones sobre suma64signed.s

1.

El máximo valor positivo que puede represertarse es el 0x7fffffff, cuyo valor en decimal es 2147483647, y la suma de 32 elementos inicializados a este valor es 0xfffffffe0 0x0000000f (little-endrian), o lo que es lo mismo, fffffffe0 (68719476704 en decimal).

2.

El menor número negativo que puede representarse en decimal es -1 (0xffffffff en hexadecimal) y la suma de 32 elementos inicializados a este valor es -32, cuya representación en hexadecimal es 0xfffffffe0.

3.

Para que el resultado obtenido sea, aparentemente, 2^32, cada elemento debe valer 2^32 / 32 = 2^32 / 2^5 = 2^27, que expresado en hexadecimal es 0x08000000. Tras ejecutar sumaconsigno.s con todos los elementos inicializados con 0x080000000, el resultado al hacer el volcado de memoria da, 0x00000000 0xfffffff0 (en little-endrian), lo que equivale a -2^36.

6.

La suma de 32 elementos iniciallizados con los valores 0xf0000000, 0xe0000000, 0xe0000000, 0xd0000000 repetidos cíclicamente es 0x00000000 0xfffffffc (little-endrian).

Código de sumaconsigno.s

```
1  .section .data                #Listas de números que sumará mi programa
       (y otro ejemplo)
2
3  lista:      .int 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1,
       1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1
4
5  #lista:     .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff
6
7  longlista:  .int (.-lista)/4        #Longitud de la lista
8  resultado:  .quad -1                #Resultado de la suma (variable de
       64 bits)
9
10 .section .text
11 _start: .global _start
12
13     mov    $lista, %ebx             #En el registro %ebx se guarda la
           dirección de memoria donde comienza la lista
```

```
14      mov longlista, %ecx             #En el registro %ecx se guarda la
            longitud de la lista
15      call suma                       #Llama a la subrutina suma
16      mov %eax, resultado             #Guarda en resultado el valor
            guardado en %eax
17      mov %edx, resultado+4           #Guarda en resultado+4 el valor
            guardado en %edx
18
19      mov $1, %eax
20      mov $0, %ebx
21      int $0x80                       #Traduce lo que hay en %eax a una
            instrucción para finalizar el programa correctamente
22
23  suma:
24      push %esi                       #Inicializa los registros que voy a
             usar en el bucle a 0
25      push %ecx
26      mov $0, %eax
27      mov $0, %edx
28      mov $0, %esi
29      mov $0, %ecx
30      mov $0, %edi
31  bucle:
32      mov (%ebx,%esi,4), %eax         #Guarda lo que hay en la posición
            con el valor de %esi de la lista en %eax
33      cdq                             #Guarda en %edx la extensión del
            signo de %eax
34      add %eax, %edi                  #Suma el valor guardado en %eax y
            con el valor de %edi, y lo guarda en %edi(acumula suma de la
            lista)
35      adc %edx, %ecx                  #Suma el valor guardado en %edx y
            con el valor de %ecx, y lo guarda en %ecx(acumula suma de la
            lista)
36      inc       %esi                  #Incrementa %esi (registra la
            posición de la lista que voy a sumar
37      cmp %esi, (%esp)                #Compara el valor de %esi con la
            longitud de la lista
38      jne bucle                       #Si %esi es menor que la longitud
            de la lista, sigue sumando
39                                      #El valor de la suma se encuentra
                                            en %ecx:edi%
40      mov %edi, %eax                  #Guarda el valor de %edi en %eax
41      mov %ecx, %edx                  #Guarda el valor de %ecx en %edx
```

```
42      pop %esi
43      pop %ecx
44      ret                             #Vuelve a la dirección de retorno
           tras la subrutina
```

Cuestiones sobre media.s

1.

Según el manual de Intel sobre la función división, si divido con un dividendo de 8 bits, divido el contenido de `%ax` entre el dividendo, el cociente lo guarda en `%al` y el resto en `%ah`. Si el dividendo tiene 16 bits, divido `%dx:%ax`, el cociente se guarda en `%ax` y el resto en `%dx`. En nuestro caso, el dividendo tiene 32 bits, luego divide `%edx:%eax`, guardando el cociente en `%eax` y el resto en `%edx`. Al dividir -31 entre 32, el cociente sale 0, que se guarda en `%eax` y el resto vale -31, luego resultado vale 0x00000000 0xfffffffe1 (little-endrian).

Código de media.s

```
1  .section .data                #Lista de números que sumará mi programa (
       y otro ejemplo)
2
3  #lista:      .int 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1,
        1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1, -1, 1, -1
4
5  lista:       .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff, 0
       x7fffffff, 0x7fffffff, 0x7fffffff
6
7  longlista:   .int (.-lista)/4           #Longitud de la lista
8  resultado:   .quad -1                   #Resultado de la suma (variable
       de 64 bits)
9
10 .section .text
11 _start: .global _start
12
```

```
13      mov    $lista, %ebx                 #En el registro %ebx se guarda
           la dirección de memoria donde comienza la lista
14      mov longlista, %ecx                 #En el registro %ecx se guarda
           la longitud de la lista
15      call suma                           #Llama a la subrutina suma
16      mov longlista, %ecx                 #En el registro %ecx se guarda
           la longitud de la lista
17      idiv %ecx                           #Divido %edx:%eax entre %ecx
18      mov %eax, resultado                 #Guarda en resultado el valor
           guardado en %eax
19      mov %edx, resultado+4               #Guarda en resultado+4 el valor
           guardado en %edx
20
21      mov $1, %eax
22      mov $0, %ebx
23      int $0x80                           #Traduce lo que hay en %eax a
           una instrucción para finalizar el programa correctamente
24
25  suma:
26      push %esi                           #Inicializa los registros que
           voy a usar en el bucle a 0
27      push %ecx
28      mov $0, %eax
29      mov $0, %edx
30      mov $0, %esi
31      mov $0, %ecx
32      mov $0, %edi
33  bucle:
34      mov (%ebx,%esi,4), %eax              #Guarda lo que hay en la posici
           ón con el valor de %esi de la lista en %eax
35      cdq                                 #Guarda en %edx la extensión
           del signo de %eax
36      add %eax, %edi                      #Suma el valor guardado en %eax
            y con el valor de %edi, y lo guarda en %edi(acumula suma de la
           lista)
37      adc %edx, %ecx                      #Suma el valor guardado en %edx
            y con el valor de %ecx, y lo guarda en %ecx(acumula suma de la
           lista)
38      inc       %esi                      #Incrementa %esi (registra la
           posición de la lista que voy a sumar
39      cmp %esi, (%esp)                    #Compara el valor de %esi con
           la longitud de la lista
```

```
40      jne bucle                               #Si %esi es menor que la
            longitud de la lista, sigue sumando
41
42      mov %edi, %eax                          #Guarda el valor de %edi en %
            eax
43      mov %ecx, %edx                          #Guarda el valor de %ecx en %
            edx
44      pop %esi
45      pop %ecx
46      ret                                     #Vuelve a la dirección de
            retorno tras la subrutina
```