Estructura de computadores
Practica 2
Documentación

Alumno: Antonio David López Machado **Grupo:** C-2

Índice

1. Planificación

Semana 3/10 - 9/10	-Realización del tutorial de la práctica -Pruebas realizadas con las diferentes herramientas que usaremos.
Semana 10/10 - 16/10	-Comprensión del primer ejercicio a realizar -Realización del ejercicio 1
Semana 17/10 - 23/10	-Comprensión del segundo ejercicio a realizar -Realización del ejercicio 2 -Comprensión del tercer ejercicio a realizar -Realización del ejercicio 3 -Mejora en los comentarios de todos los ejercicios de la práctica -Realización de preguntas opcionales -Desarrollo de la planificación
Semana 24/10 - 28/10	-Finalización de la práctica -Entrega de la práctica

2. Código de los ejercicios

2.1 Ejercicio 1

```
##################################################
#
#       Antonio David López Machado - Curso 2016/2017
#       Practica-2 Ejercicio 1
#
##################################################

.section .data
        .macro linea
                #.int 1,2,10, 1,2,0b10, 1,2,0x10
                #.int 1,1,1,1,1,1,1,1,1
                #.int 2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2
                #.int 1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4
                #.int
0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xf
fffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff,0xffffffff
                #.int 0x08000000,0x08000000,0x08000000,0x08000000
                .int
0x10000000,0x20000000,0x40000000,0x80000000,0x10000000,0x20000000,0x40000000,0x80000000,0x1000
0000,0x20000000,0x40000000,0x80000000,0x10000000,0x20000000,0x40000000,0x80000000,0x10000000,0
x20000000,0x40000000,0x80000000,0x10000000,0x20000000,0x40000000,0x80000000,0x10000000,0x20000
000,0x40000000,0x80000000,0x10000000,0x20000000,0x40000000,0x80000000
        .endm
lista: .irpc i,1
                linea
    .endr
```

```
longlista:
        .int (.-lista)/4
resultado:
        .quad 0x01234567

formato:
        .ascii "suma = %llu = %llx hex\n\0"

.section .text
main: .global main

        mov $lista,    %ebx              #Introducimos en el registro ebx la lista de elementos
        mov longlista, %ecx              #Introducimos en el registro ecx el tamaño de la lista de
elementos
        call suma                        #Llamada al metodo suma
        mov %edi, resultado+4            #Introducimos en el resultado los byte mas significativos
        mov %eax, resultado              #Introducimos en el resultado los byte menos significativos



        pushl resultado+4        #Realizamos los push necesarias para realizar un printf con
        pushl resultado          #Valores de 64bits
        pushl resultado+4
        pushl resultado
        push $formato
        call printf              #Realizamos la impresión del resultados
        add $20,%esp

        mov $1, %eax             #Ponemos los valores necesarias para realizar la salida del programa
        mov $0, %ebx

        int $0x80                #Realizamos la salida del programa

#######################################################
# Método que realiza la suma sin signo de un conjunto de elementos
# pasados como parametro
# Devuelve la suma total de todos los elementos en los registros:
# edi  -> bytes más significativos
# eax -> bytes menos significativos
#######################################################

suma:
        push %edx
        mov $0, %eax    #Inicializamos a 0 los registros
        mov $0, %edx    #que utilizaremos en nuestro método
        mov $0, %edi

bucle:
        add (%ebx,%edx,4),%eax          #Realizamos la suma del elemento de la lista
        jnc sinC                        #Saltamos a la etiquet sinC cuando no se produzca acarreo
        add $1,%edi                     #Caso -> Con carrero
                                        #Añadimos el carrero al registro con los valores mas
                                        significativos(%edi)
```

```
sinC:   inc %edx                    #Incrementamos el contador
        cmp %edx,%ecx               #Comparamos nuestro contador
        jne bucle                   #Si no ha llegado al final vuelve a empezar en la
                                    etiqueta bucle:
        pop %edx
        ret
```

2.2 Ejercicio 2

```
#####################################################
#
#       Antonio David López Machado - Curso 2016/2017
#       Practica-2 Ejercicio 2
#
#####################################################

.section .data
        .macro linea
                #.int  1,2,10, 1,2,0b10, 1,2,0x10
                .int -1,-1,-1,-1,-1,-1,-1,-1,-1
                #.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,-1
                #.int 1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2
                #.int 1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4

        .endm
lista: .irpc i,1
                linea
    .endr

longlista:
        .int (.-lista)/4
resultado:
        .quad 0x01234567

formato:
        .ascii "suma = %lld = %llx hex\n\0"

.section .text
main: .global main

        mov $lista,    %ebx          #Introducimos en el registro ebx la lista de elementos
        mov longlista, %ecx          #Introducimos en el registro ecx el tamaño de la lista de
elementos
        call suma                    #Llamada al metodo suma
        mov %edi, resultado+4        #Obtenemos la suma acumulada
        mov %esi, resultado          #Y la introducimos en la variable resultado

        pushl resultado+4       #Realizamos los push necesarias para realizar un printf con
        pushl resultado         #Valores de 64bits
        pushl resultado+4
        pushl resultado
        push $formato
        call printf             #Realizamos la impresión del resultados
        add $20,%esp
```

```
        mov $1, %eax            #Ponemos los valores necesarias para realizar la salida del programa
        mov $0, %ebx

        int $0x80               #Realizamos la salida del programa

#####################################################
# Método que realiza la suma con signo de un conjunto de elementos
# pasados como parametro
# Devuelve la suma total de todos los elementos en los registros:
# edi -> bytes mas significativos
# esi -> bytes menos significativos
#####################################################
suma:
        push %edx
        mov $0, %edx    #mas   significativo sin acumular
        mov $0, %eax    #menos significativo sin acumular
        mov $0, %ebp    #contador del bucle
        mov $0, %edi    #mas   significativo acumulado
        mov $0, %esi    #menos significativo acumulado

bucle:
        mov (%ebx,%ebp,4),%eax      #Movemos el elemento actual de la lista
        cdq                         #Realizamos la instruccion cdq y nos introduce los valores
        add %eax,%esi               #resltantes en eax y edx
        adc %edx,%edi               #Acumulamos ambos valores

        inc %ebp                    #Incrementamos el contador
        cmp %ebp,%ecx               #Comparamos nuestro contador
        jne bucle                   #Si no ha llegado al final vuelve a empezar en la etiqueta bucle:

        pop %edx
        ret
```

2.3 Ejercicio 3

```
#####################################################
#
#       Antonio David López Machado - Curso 2016/2017
#       Practica-2 Ejercicio 3
#
#####################################################


.section .data
        .macro linea
                #.int  1,2,10, 1,2,0b10, 1,2,0x10
                #.int -1,-1,-1,-1,-1,-1,-1,-1,-1,-1
                #.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,-1,-1
                .int 1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2,1,-2
                #.int 1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4,1,2,-3,-4

        .endm
lista: .irpc i,1
```

```
                    linea
    .endr
longlista:
        .int (.-lista)/4
cociente:
        .int 0x01234567
resto:
        .int 0x01234220

formato:
        .ascii "Cociente : %8d -> 0x%08x hex \nResto : %8d -> 0x%08x hex \n\0"

.section .text
main: .global main

        mov $lista,    %ebx      #Introducimos en el registro ebx la lista de elementos
        mov longlista, %ecx      #Introducimos en el registro ecx el tamaño de la lista de elementos
        call suma                #Llamada al metodo suma
        mov %edi, resto          #Introducimos el resto en la variable
        mov %eax, cociente       #Introducimos el cociente en la variable

        pushl resto              #Realizamos los push necesarias para realizar un printf
        pushl resto              #Tanto del cociente como del resto
        pushl cociente
        pushl cociente
        push $formato
        call printf              #Realizamos la impresión del resultados
        add $20,%esp

        mov $1, %eax             #Ponemos los valores necesarias para realizar la salida del programa
        mov $0, %ebx

        int $0x80                #Realizamos la salida del programa


#####################################################
# Método que realiza la suma con signo de un conjunto de elementos
# y realiza su media
# Devuelve el cociente y el resto de la media en los registros:
# edi -> el resto
# eax -> cociente
#####################################################
suma:
        push %edx
        mov $0, %edx    #mas   significativo sin acumular
        mov $0, %eax    #menos significativo sin acumular
        mov $0, %ebp    #contador del bucle
        mov $0, %edi    #mas   significativo acumulado
        mov $0, %esi    #menos significativo acumulado

bucle:
        mov (%ebx,%ebp,4),%eax          #Movemos el elemento actual de la lista
        cdq                      #Realizamos la instruccion cdq y nos introduce los valores
        add %eax,%esi            #resultantes en eax y edx
```

```
adc %edx,%edi           #Acumulamos ambos valores

inc %ebp                #Incrementamos el contador
cmp %ebp,%ecx           #Comparamos nuestro contador
jne bucle               #Si no ha llegado al final vuelve a empezar en la etiqueta bucle:

mov %esi,%eax  # Realizamos estos mov ya que idiv necesita el dividendo en %edx:%eax
mov %edi,%edx
idiv %ecx          # %edx:%eax/%ecx  = Resto-> %edx Cociente-> %eax

mov %edx,%edi  #guardamos nuestro resto ya que debido al pop perderiamos el valor

pop %edx
ret
```

3. Ejercicios Opcionales

3.1 Sesión de depuración saludo.s

1.- ¿Qué contiene EDX tras ejecutar mov longsaludo, %edx?¿Para qué necesitamos esa instrucción, o ese valor?Responder no sólo el valor concreto(en decimal y hex) sino también el significado del mismo.

-Contiene el tamaño de la cadena saludo el cual es 28(0x1c en hexadecimal)
-Lo necesitamos para realizar la impresión por pantalla ya que un parámetro de lo que requiere es el tamaño de la cadena que queremos imprimir por pantalla

7.- ¿Qué sucede si se elimina del programa la primera instrucción int 0x80?¿Y si se elimina la segunda?Razona las respuestas.

-Es la llamada para la impresión por lo que al quitarla no se realiza la impresión por pantalla de la cadena.
-No realiza la llamada al sistema para salir del programa por lo que si hubiese más código seguiría la ejecución. Al no poner esa llamada al sistema para salir del programa se produce un fallo de segmentación

8.- ¿Cuál es el número de la llamada al sistema READ(en kernel Linux 32bits)?¿De dónde se ha obtenido esa información?

-El número de la llamada al sistema es el 0x03
-Información obtenida a traves de la plataforma Swad, en la sección FAQ de esta asignatura -> url de la información : http://syscalls.kernelgrok.com/

3.2 Sesión de depuración suma.s

1.- ¿Cuál es el contenido de EAX justo antes de ejecutar la instrucción RET, para esos componentes de lista concretos?Razonar la respuesta, incluyendo cuánto valen 0b10,0x10, y(.-lista)/4

-Su contenido es 37. Ese es el resultado de realizar la suma de los elementos de la lista. La suma es 1+2+10+1+2+0xb10+1+2+0x10 donde 0xb10 [FIX]

3.- ¿Qué dirección se le ha asignado a la etiqueta suma?¿Y a bucle?¿Cómo se ha obtenido esa información?

La etiqueta suma tiene el valor 0x8048457 y la etiqueta bucle tiene el valor 0x8048462

Se ha obtenido mediante ddd utilizando el comando print etiqueta.

4.-¿Para qué usa el procesador los registros EIP y ESP?

-EIP -> es utilizado para saber la instrucción a ejecutar(contador de programa)

-ESP-> es un puntero al inicio de la pila

5.-¿Cuál es el valor de ESP antes de ejecutar CALL, y cuál antes de ejecutar RET?¿En cuánto se diferencian ambos valores?¿Por qué?¿Cuál de los valores de ESP apunta a algún dato de interés para nosotros?¿Cuál es ese dato'

-El valor de %esp es 0xffffcffc antes de ejecutar CALL y 0xffffcff8 antes de ejecutar RET.

-Se diferencia por un total de 4, porque en call se realiza un push en la pila lo que hace que aumente en 4 (4 es el tamaño del elemento en la pila) y antes de ejecutar el RET aun tenemos ese push en la pila.

-0xffffcff8 apunta a la instrucción de retorno, es decir, la instrucción que se realiza desde de usar el método suma.

6.-¿Qué registros modifica la instrucción CALL?Explicar por qué necesita CALL modificar esos registros

-El puntero a la pila(%esp) lo modifica ya que introduce un nuevo elemento en la pila. Este elemento contiene la instrucción a la que tendrá que volver al acabar el método

-También modifica el registro %eip ya que introduce la dirección de la primera instrucción del método llamado por call

7.-¿Qué registros modifica la instrucción RET?Explicar por qué necesita RET modificar esos registros

-Modifica el puntero a la pila(%esp) el cual apunta ahora a la dirección siguiente, es decir, mantenemos en la pila la dirección de retorno pero el puntero apunta a la siguiente dirección por lo que se queda más allá del tope, por lo que no se considera que esté en la pila.

-También modifica el registro%eip ya que introduce la dirección de retorno(instrucción siguiente a la instrucción CALL).

10.-¿Qué ocurriría si se eliminara la instrucción RET?Razonar la respuesta.Comprobarlo usando ddd

Lo que ocurriría es que no se realiza un retorno de valor por lo que nuestro registro %eax se queda en 0 en vez contener una parte del resultado(la menos significativa).

3.3 Cuestiones sobre suma64unsigned.s

1.-Para N=32, ¿Cuántos bits adicionales pueden llegar a necesitarse para almacenar el resultado? Dicho resultado se alcanzaría cuando todos los elementos tomarán el valor máximo sin signo. ¿Cómo se escribe ese valor en hexadecimal?¿Cuántos acarreos se producen?¿Cuánto vale la suma (indicarla en hexadecimal)?Comprobarlo usando ddd.

-Pueden necesitar un total de 5 bits más ya que como máximo se producirán 31 acarreos.

-El valor maximo sin signo es 4294967295 y en hexadecimal sería 0xFFFFFFFF

-Por cada suma se produciría un acarreo menos en el primer elemento ya que no produce acarreo por sumarse con 0, es decir, que se producirían un total de 31 acarreos.

-La suma vale 0x1FFFFFFFe0 en hexadecimal.

3- Por probar valores intermedios: si la lista se inicializará con los valores 0x10000000, 0x20000000, 0x40000000, 0x80000000, repetidos cíclicamente, ¿qué valor tomaría la suma de los 32 elementos?¿Cuándo se producirían los acarreos?Comprobarlo con ddd

-El valor sería 32212254720(0x780000000 en hexadecimal)

-Al final de cada ciclo de elementos y sumando un elemento más cada vez. Es decir, la primera vez para conseguir un acarreo sería con la suma del elemento 0x100.. del siguiente ciclo, la segunda vez para conseguir acarreo sería con la suma de los elementos 0x100.. y 0x200 del siguiente ciclo, así sucesivamente. Dando así un total de 7 carreos.

3.4 Cuestiones sobre suma64signed.s

5. Respecto a negativos, -2^{31} sí cabe en 32 bits como número negativo en complemento a dos.Calcular qué valor de elemento se requiere para obtener como suma -2^{31}, y para obtener -2^{32}. Comprobarlo usando ddd.

Como tendremos solo 32 elementos en la lista cada elemento deberá valer -67.108.864 para obtener un total de -2^{31}

Para obtener un total de -2^{32} cada elemento de la lista deberá valer 134.217.728

6. Por probar valores intermedios: si la lista se inicializará con los valores 0xF0000000,0xE0000000,0xE0000000,0xD0000000, repetidos cíclicamente, ¿Qué valor tomaría la suma de los 32 elementos(en hex)?Comprobarlo con ddd.

La suma tomará un valor total de 0xFFFFFFFC00000000 en hexadecimal (-17.179.869.184 decimal)