

Estructura de Computadores (2016-2017)
GRADO EN INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE GRANADA

Memoria Práctica 2

Álvaro Fernández-Alonso Araluce

30 de octubre de 2016

Índice

1. Sumar N enteros <u>sin signode</u> 32 bits en una plataforma de 32 bits sin perder precisión (N=32)	3
1.1. Suma	3
1.2. Bucle	4
1.3. Código completo	4
2. Sumar N enteros <u>con signode</u> 32 bits en una plataforma de 32 bits	7
3. Media de N enteros <u>con signode</u> 32 bits, en plataforma de 32 bits	9
4. Depuración saludo.s	12
4.1. ¿Qué contiene el registro EDX tras ejecutar mov longsaludo,%edx ? . .	12
4.2. ¿Qué contiene el registro ECX tras ejecutar mov saludo,%ecx ?	12

Índice de figuras

3.1. Ejecutando el código media.s	11
4.1. Valor del registro EDX en decimal y hexadecimal tras ejecutar mov longsaludo,%edx	12
4.2. Valor del registro ECX en decimal y hexadecimal tras ejecutar mov saludo,%ecx	13

1 Sumar N enteros sin signo de 32 bits en una plataforma de 32 bits sin perder precisión (N=32)

En este ejercicio debemos sumar una lista de enteros sin signo. La dificultad de este ejercicio consiste en salvar las sumatorias en registros acumuladores ya que, en ciertos casos, debemos acumular el acarreo de la operación.

En este ejercicio la solución la reproducimos en la sección **suma** y **bucle** al que entra inevitablemente después de ejecutar todas las instrucciones que se realizan en la sección **suma**.

Suma

suma:

```
push %ebp          # preservar %ebp
xor %eax, %eax     # poner a 0 el acumulador
xor %edx, %edx     # poner a 0 el acumulador 2
xor %ebx, %ebx     # poner a 0 el registro de acarreo
xor %esi, %esi     # poner a 0 el indice

mov %esp, %ebp
```

Esta sección se encargará de inicializar dos acumuladores, uno para acumular los elementos y el segundo para acumular la parte del acarreo; poner a cero los registros de acarreo e índice.

Como vamos a romper la linealidad del programa, almacenaremos el registro `%ebp` para poder después recuperarlo antes de hacer la llamada **ret**. De esta forma, retornaremos al punto del código después de la llamada a la función **suma**.

Bucle

```
bucle:
    mov (%edi,%esi,4), %ecx # ecx = lista[esi]

    add %ecx, %eax          # acumular i-esimo elemento
    adc %ebx, %edx          # acumular i-esimo elemento si acarreo
    inc %esi                # incrementar indice
    cmp 0x8(%esp), %esi     # comparar con longitud
    jne bucle              # si no iguales, seguir acumulando

    pop %ebp                # recuperar %ebp antiguo
    ret
```

Cuando llegamos a la etiqueta **bucle**, accedemos a la posición **%esi** en la lista y almacenamos su contenido en el registro **%ecx**. El resultado, por convenio, se almacena en el registro **%eax**, así que usamos este registro para ir almacenando la sumatoria de los elementos de la lista.

Adicionalmente hacemos uso de la instrucción **adc**, que realiza una suma si el flag de acarreo está activado. Una vez hemos realizado y acumulado la suma del elemento realizamos un incremento del índice **%esi** y, después, comparamos la longitud de la lista con el índice actual. La operación **cmp** compara dos elementos y activa los flags correspondientes, en este caso vamos a comparar la longitud de la lista con el índice y usaremos la instrucción **jne** para ver si el flag zero no está activado. Si el resultado de la instrucción **jne** arroja un true haremos un salto a la etiqueta bucle y en caso false seguiremos ejecutando instrucciones hasta llegar a ret.

Código completo

```
.section .data
    .macro linea
    #      .int 1,1,1,1
    #      .int 2,2,2,2
    #      .int 1,2,3,4
    #      .int -1,-1,-1,-1
    #      .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
    #      .int 0x00000000,0x00000000,0x00000000,0x00000000
    #      .int 0x10000000,0x20000000,0x40000000,0x80000000
    .endm
lista:.irpc i,12345678
    linea
```

```

        .endr
longlista:
        .int (.-lista)/4 #lista.lenght() se divide entre 4. int = 4bytes
resultado:
        .quad -1          #Aqui guardaremos el resultado
formato:
        # llu (long long unsigned, decimal)
        # llx (hexadecimal)
        .ascii "suma_=_%llu_=_0x%llx_hex\n\0"

# SECCION DE CODIGO (. text, instrucciones maquina)

.section .text                                # PROGRAMA PRINCIPAL
#_start          .global _start              # se puede abreviar de esta forma
main: .global main                            # Programa principal si se usa gcc

        mov $lista, %edi                     # direccion del array lista
        push longlista                       # numero de elementos a sumar
        call suma                            # llamar suma(&lista, longlista);
        add $4, %esp

        mov %eax, resultado                  # salvar resultado
        mov %edx, resultado + 4              # salvar resultado

        push resultado + 4
        push resultado                      # version libC de syscall__NR_write
        push resultado + 4
        push resultado                      # ventaja: printf() con formato "%d" / "%x"
        push $formato                      # traduce resultado a ASCII decimal/hex
        call printf                         # ==printf(formato, resultado, resultado)

        add $20, %esp

# void _exit(int status);
mov $1, %eax                                # exit: servicio 1 kernel Linux
mov $0, %ebx                                # status: codigo a retornar (0=OK)
int $0x80                                   # llamar _exit(0);

# SUBROUTINA: suma(int* lista, int longlista);
# entrada:
#
#                                     1) %ebx = direccion inicio array
#                                     2) %ecx = numero de elementos a sumar
# salida:                                %eax = resultado de la suma

```

suma:

```
push %ebp          # preservar %ebp
xor %eax, %eax     # poner a 0 el acumulador
xor %edx, %edx     # poner a 0 el acumulador 2
xor %ebx, %ebx     # poner a 0 el registro de acarreo
xor %esi, %esi     # poner a 0 el indice

mov %esp, %ebp
```

bucle:

```
mov (%edi,%esi,4), %ecx # ecx = lista[esi]

add %ecx, %eax       # acumular i-esimo elemento
adc %ebx, %edx       # acumular i-esimo elemento si acarreo
inc %esi             # incrementar indice
cmp 0x8(%esp), %esi  # comparar con longitud
jne bucle            # si no iguales, seguir acumulando

pop %ebp             # recuperar %ebp antiguo
ret
```

2 Sumar N enteros con signo de 32 bits en una plataforma de 32 bits

```
.section .data
    .macro linea
        # .int -1,-1,-1,-1
        # .int 1,-2,1,-2
        # .int 1,2,-3,-4
        # .int 0x7fffffff,0x7fffffff,0x7fffffff,0x7fffffff
        # .int 0x80000000,0x80000000,0x80000000,0x80000000
        # .int 0x04000000,0x04000000,0x04000000,0x04000000
        # .int 0x08000000,0x08000000,0x08000000,0x08000000
        # .int 0xFC000000,0xFC000000,0xFC000000,0xFC000000
        # .int 0xF8000000,0xF8000000,0xF8000000,0xF8000000
        .int 0xF0000000,0xE0000000,0xE0000000,0xD0000000
    .endm
lista: .irpc i,12345678
        linea
    .endr
longlista:
    .int (.-lista)/4 #lista.lenght() se divide entre 4. int = 4bytes
resultado:
    .quad -1          #Aqui guardaremos el resultado
formato:
    .ascii "suma_=_%dli_=_%lx_hex\n\0"

# SECCION DE CODIGO (. text, instrucciones maquina)

.section .text
#_start .global _start
main: .global main

    mov $lista, %edi      # direccion del array lista
    push longlista        # numero de elementos a sumar
    call suma             # llamar suma(&lista, longlista);
    add $4, %esp

    mov %eax, resultado   # salvar resultado
    mov %edx, resultado + 4 # salvar resultado

    push resultado + 4
```

```

push resultado                # version libC de syscall__NR_write
push resultado + 4
push resultado                # ventaja: printf() con formato "%d" / "%x"
push $formato                 # traduce resultado a ASCII decimal/hex
call printf                   # ==printf(formato, resultado, resultado)

add $20, %esp

# void _exit(int status);
mov $1, %eax # exit: servicio 1 kernel Linux
mov $0, %ebx # status: codigo a retornar (0=OK)
int $0x80    # llamar _exit(0);

# SUBROUTINA: suma(int* lista, int longlista);
# entrada:
#           1) %ebx = direccion inicio array
#           2) %ecx = numero de elementos a sumar
# salida:
#           %eax = resultado de la suma

suma:
    push %ebp                # preservar %ebp
    mov $0, %eax             # poner a 0 acumulador
    mov $0, %edx             # poner a 0 acumulador 2
    mov $0, %esi             # poner a 0 indice

    mov %esp, %ebp

bucle:
    mov (%edi,%esi,4), %ecx # ecx = lista[esi]
    mov %ecx, %ebx
    sar $0x1f, %ebx
    add %ecx, %eax           # acumular i-esimo elemento
    adc %ebx, %edx           # acumular i-esimo elemento si acarreo
    inc %esi                 # incrementar indice
    cmp 0x8(%esp), %esi      # comparar con longitud
    jne bucle                # si no iguales, seguir acumulando

    pop %ebp                 # recuperar %ebp antiguo
    ret

```


3 Media de N enteros con signo de 32 bits, en plataforma de 32 bits

Este ejercicio es el mismo que el anterior. La diferencia radica en que, al realizar la sumatoria de los elementos de la lista, debemos dividir el resultado por el número de elementos de la lista, es decir, *longlista*.

Para dividir los registros usaremos el comando **idivl** [1], **Integer Division** en sus siglas en inglés. Esta instrucción realiza una división **con signo** de los registros *%edx::%eax* entre *longlista* para así poder dividir la sumatoria por el número de elementos de la lista.

```
.section .data
    .macro linea
        .int 1,2,3,4,5
    #      .int 5,5,5,5
    .endm
lista: .irpc i,12345678
        linea
    .endr
longlista:
    .int (.-lista)/4 #lista.lenght() se divide entre 4. int = 4bytes
resultado:
    .quad -1          #Aqui guardaremos el resultado
formato:
    .ascii "media_entera_=_%i\n\0"

# SECCION DE CODIGO (. text, instrucciones maquina)

.section .text                                # PROGRAMA PRINCIPAL
#_start .global _start                        # se puede abreviar de esta forma
main: .global main                            # Programa principal si se usa gcc

    mov $lista, %edi                          # direccion del array lista
    push longlista                            # numero de elementos a sumar
    call suma                                # llamar suma(&lista, longlista);
    add $4, %esp

    mov %eax, resultado                       # salvar resultado
    mov %edx, resultado + 4                   # salvar resultado

    idivl longlista                           # dividir entre longlista para hacer la media
```

```

push %eax                # ponemos el resultado de la division en la pi

push $formato            # traduce resultado a ASCII decimal/hex
call printf              # ==printf(formato, resultado, resultado)

add $12, %esp

# void _exit(int status);
mov $1, %eax             # exit: servicio 1 kernel Linux
mov $0, %ebx             # status: codigo a retornar (0=OK)
int $0x80                # llamar _exit(0);

# SUBROUTINA: suma(int* lista, int longlista);
# entrada:
#           1) %ebx = direccion inicio array
#           2) %ecx = numero de elementos a sumar
# salida:
#           %eax = resultado de la suma

suma:
    push %ebp            # preservar %ebp
    mov $0, %eax         # poner a 0 acumulador
    mov $0, %edx         # poner a 0 acumulador 2
    mov $0, %esi         # poner a 0 indice

    mov %esp, %ebp

bucle:
    mov (%edi,%esi,4), %ecx # ecx = lista[esi]
    mov %ecx, %ebx
    sar $0x1f, %ebx
    add %ecx, %eax        # acumular i-esimo elemento
    adc %ebx, %edx        # acumular i-esimo elemento si acarreo
    inc %esi              # incrementar indice
    cmp 0x8(%esp), %esi   # comparar con longitud
    jne bucle             # si no iguales, seguir acumulando

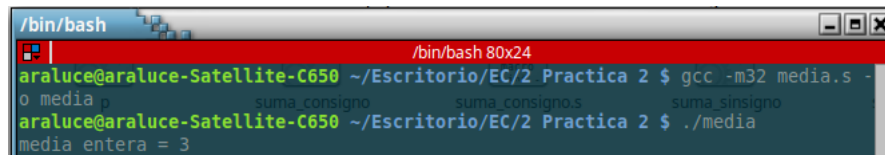
    pop %ebp              # recuperar %ebp antiguo
    ret

```

Vamos a realizar la media de los elementos de la lista 1,2,3,4,5 a mano:

$$\frac{1 + 2 + 3 + 4 + 5}{5} = \frac{15}{5} = 3$$

Si compilamos y ejecutamos el código con esos mismos datos nos da el mismo resultado:



```

/bin/bash
araluce@araluce-Satellite-C650 ~/Escritorio/EC/2 Practica 2 $ gcc -m32 media.s -o media
araluce@araluce-Satellite-C650 ~/Escritorio/EC/2 Practica 2 $ ./media
media entera = 3

```

Figura 3.1: Ejecutando el código media.s

4 Depuración saludo.s

¿Qué contiene el registro EDX tras ejecutar `mov longsaludo, %edx`?

Contiene el valor 28 en decimal y hexadecimal (1C). Este número corresponde a la longitud de la cadena **saludo** y se le ha dado el valor a través de la línea:

```
longsaludo:                .int     .-saludo
```

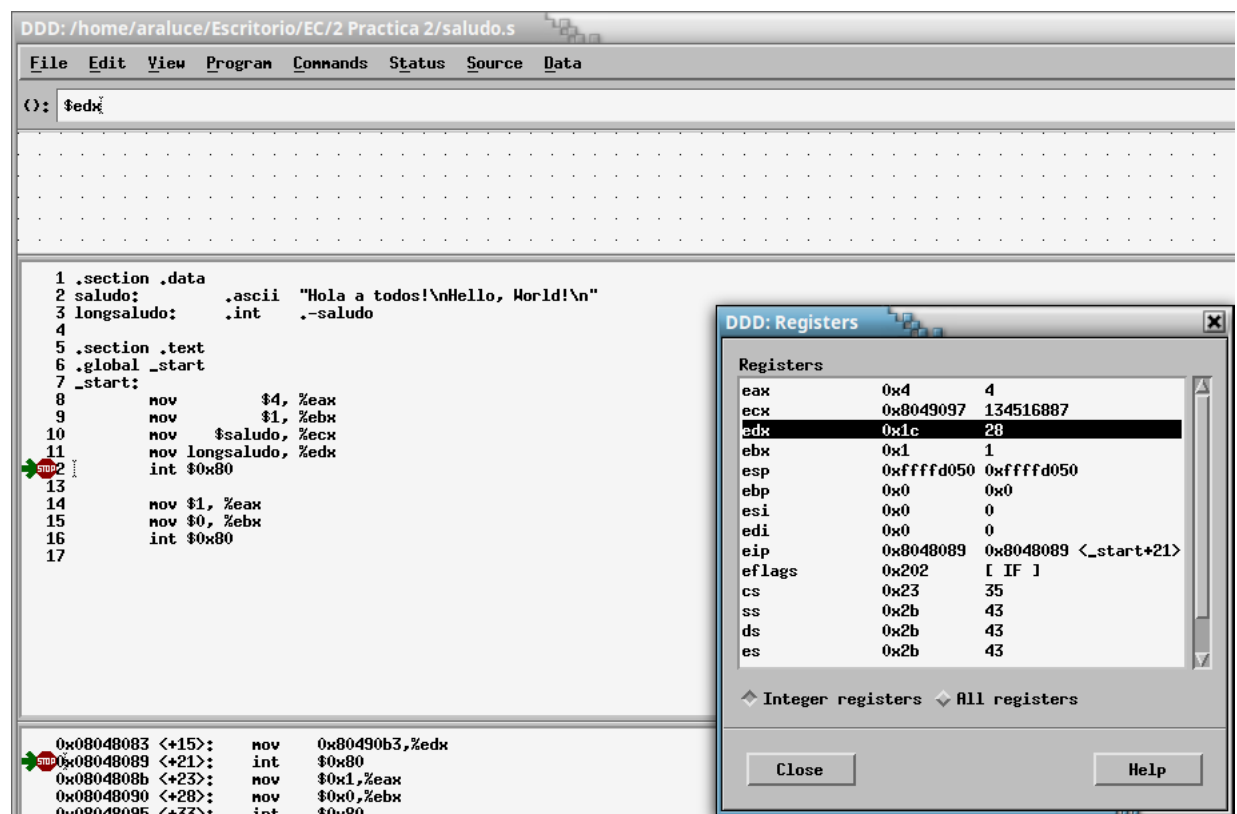


Figura 4.1: Valor del registro EDX en decimal y hexadecimal tras ejecutar `mov longsaludo, %edx`

¿Qué contiene el registro ECX tras ejecutar `mov saludo, %ecx`?

Contiene la posición de memoria donde está almacenada la cadena "Hola a todos! Hello, World!"

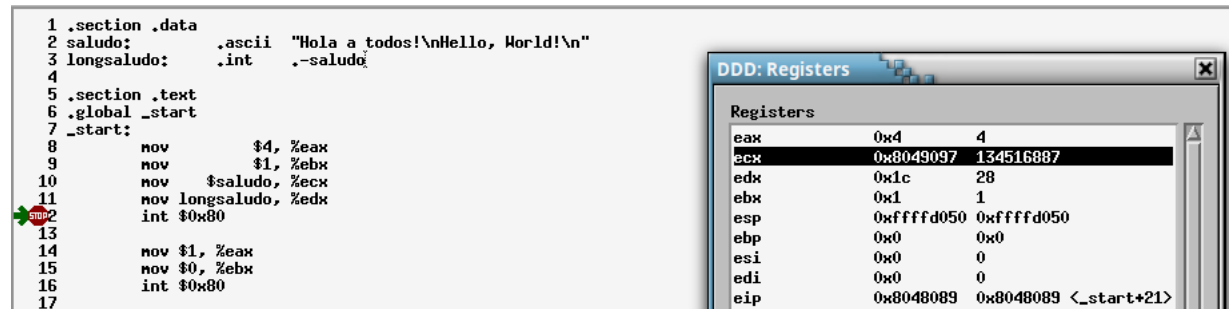


Figura 4.2: Valor del registro ECX en decimal y hexadecimal tras ejecutar `mov saludo, %ecx`

Referencias

- [1] <https://www.lri.fr/~filliatr/ens/compil/x86-64.pdf>, 20 de Octubre 2016.