

## Preguntas de Autocomprobación

-----Sesión de depuración saludo.s (tabla 10)-----

**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 o del mismo (¿de dónde sale?) Comprobar que se corresponden los valores hexadecimal y decimal mostrados en la ventana Status->Registers**

En el registro `edx` almacenamos la cantidad o la longitud de saludo.

Utilizamos esta instrucción para que la variable `longsaludo` tome automáticamente el valor correcto (longitud) para la posterior llamada a `WRITE`, ajustándose automáticamente también cuando cambiemos el tamaño de saludo.

**2 ¿Qué contiene ECX tras ejecutar `mov $saludo, %ecx`? Indicar el valor en hexadecimal, y el significado del mismo. Realizar un dibujo a escala de la memoria del programa, indicando dónde empieza el programa (`_start`, `.text`), dónde empieza `saludo` (`.data`), y dónde está el tope de pila (`%esp`)**

`%ECX` contiene la dirección de memoria donde se encuentra el texto de saludo

**3 ¿Qué sucede si se elimina el símbolo de dato inmediato (\$) de la instrucción anterior? (`mov saludo, %ecx`) Realizar la modificación, indicar el contenido de `ECX` en hexadecimal, explicar por qué no es lo mismo en ambos casos. Concretar de dónde viene el nuevo valor (obtenido sin usar \$)**

Como queremos almacenar la dirección de memoria y no lo que contiene la variable `saludo`, utilizamos \$, y si lo omitiéramos almacenaríamos en el registro el contenido de la variable.

**4 ¿Cuántas posiciones de memoria ocupa la variable `longsaludo`? ¿Y la variable `saludo`? ¿Cuántos bytes ocupa por tanto la sección de datos? Comprobar con un volcado `Data->Memory` mayor que la zona de datos antes de hacer `Run`.**

`Longsaludo` contiene el conteo del tamaño de saludo, por lo que ocupa un bloque de la pila de 4 bytes.

`Saludo` contiene el texto, por lo que es una cadena de caracteres, por lo tanto ocupará según los caracteres que contenga la cadena.

**5 Añadir dos volcados `Data->Memory` de la variable `longsaludo`, uno como entero hexadecimal, y otro como 4 bytes hex. Teniendo en cuenta lo mostrado en esos volcados... ¿Qué direcciones de memoria ocupa `longsaludo`? ¿Cuál byte está en la primera posición, el más o el menos significativo? ¿Los procesadores de la línea x86 usan el criterio del extremo mayor (big-endian) o menor (little-endian)? Razonar la respuesta**

**6 ¿Cuántas posiciones de memoria ocupa la instrucción `mov $1, %ebx`? ¿Cómo se ha obtenido esa información? Indicar las posiciones concretas en hexadecimal.**

La instrucción hace poner todos los bits del registro a uno, por lo que como trabajamos con registros de 32 bits, la instrucción ocupará 4 bytes.

**7 ¿Qué sucede si se elimina del programa la primera instrucción `int 0x80`? ¿Y si se elimina la segunda? Razonar las respuestas**

La primera instrucción `int 0x80` realiza la llamada al sistema `WRITE` y la segunda `EXIT`, por lo que si omitimos la primera el programa no escribirá por pantalla, y si omitimos la segunda daría un fallo de compilación ya que no terminaría el programa.

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

La instrucción `int 0x80`, es la interrupción software disponible en cualquier procesador de la arquitectura x86, por lo que la podemos utilizar para `READ`, teniendo cuidado con los argumentos de cada instrucción.

Los números de servicios (llamada) pueden encontrarse en `/usr/include/asm/unistd_32.h`, y los argumentos de cada llamada pueden conocerse leyendo la correspondiente página del manual de la sección 2 (Llamada al Sistema).

-----Sesión de depuración `suma.s` (tabla 11)-----

**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`**

El contenido de `eax` antes de ejecutar `RET`, podemos decir que se refiere justo después de terminar el bucle por lo que contendrá el resultado de la suma final.

**2 ¿Qué valor en hexadecimal se obtiene en resultado si se usa la lista de 3 elementos: `.int 0xffffffff, 0xffffffff, 0xffffffff`? ¿Por qué es diferente del que se obtiene haciendo la suma a mano? NOTA: Indicar qué valores va tomando `EAX` en cada iteración del bucle, como los muestra la ventana `Status->Registers`, en hexadecimal y decimal (con signo). Fijarse también en si se van activando los flags `CF` y `OF` o no tras cada suma. Indicar también qué valor muestra resultado si se vuelca con `Data->Memory` como decimal (con signo) o `unsigned` (sin signo).**

4294967264 = ffffffff0 hex

Esto se debe a que el registro de 32 bits contendrá todos los bits del registro en -1 por lo que es diferente trabajar con la suma de registros `0xffffffff`, a sumar a mano -1.

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

La dirección de la etiqueta suma es 0x08048095 y la de bucle es 0x080480a0.  
Con el registro EIP podemos averiguar la dirección de las etiquetas

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

Usualmente se usa ESP para obtener la dirección de memoria donde se encuentra el último valor almacenado en la pila por el procesador y el registro EIP se usa para apuntar a la dirección de la próxima instrucción.

**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 dos valores de ESP apunta a algún dato de interés para nosotros? ¿Cuál es ese dato?**

Antes de ejecutar la llamada call no sabemos el valor de ESP, ya que en tiempo de ejecución reservamos 12 bytes para la pila.

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

La instrucción CALL modifica los registros esp y eip.  
Es necesario para poder saber cual es la siguiente instrucción para ejecutar y poder volver a seguir ejecutando el programa después del retorno.

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

A igual que CALL, RET modifica los registros esp y eip.  
Pero en este caso, recupera la instrucción por la que se quedó antes de ejecutar call y sigue la ejecución del programa.

**8 Indicar qué valores se introducen en la pila durante la ejecución del programa, y en qué direcciones de memoria queda a cada uno. Realizar un dibujo o de la pila con dicha información. NOTA: en los volcados Data- > Memory se puede usar \$esp para referirse a donde apunta el registro ESP**

La pila comienza en (0xbffff360), al llamar a suma se decrementa en 4 (0xbffff35c) y se mete el valor de retorno.  
Cuando hace push %edx, la pila se vuelve a decrementar en 4 (0xbffff358), y metemos en la pila el valor 0.  
Al hacer pop %edx, se incrementa el valor de la pila en 4(0xbffff35c) y se saca el valor 0 de %edx.  
Y al retorna se vuelve a incrementar el valor en 4(0xbffff360) y extrae el valor del punto de retorno.

**9 ¿Cuántas posiciones de memoria ocupa la instrucción mov \$0, %edx? ¿Y la instrucción inc %edx? ¿Cuáles son sus respectivos códigos máquina? Indicar cómo se han obtenido.**

**NOTA: en los volcados Data-> Memory se puede usar una dirección hexadecimal 0x... para indicar la dirección del volcado. Recordar la ventana View-> >Machine Code Window.**

**Recordar también la herramienta objdump.**

mov \$0, %edx utiliza 8 posiciones de memoria, 4 bytes por argumento.

inc %edx ocupa 1 posición de memoria.

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

Darí error de compilación, aun así si no se ejecutara la instrucción ret, no podría retornar la instrucción por la que se quedó antes de la llamada, y no imprimirá el resultado.

-----Cuestiones de suma64unsigned.s (tabla 12)-----

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

El máximo valor que puede tomar un numero sin signo es 0xffffffff que en decimal seria 4294967295, si lo sumamos 32 veces, necesitaríamos 7 bits mas para representar el numero. El numero obtenido por la suma de 32 veces el numero 0xffffffff seria en hexadecimal 1ffffffe0. En cada suma se producirá un acarreo, así que tendremos 31 acarreo.

**2 Si nos proponemos obtener sólo 1 acarreo con una lista de 32 elementos iguales, el o objetivo es que la suma alcance 2 elevado a 32 (que ya no cabe en 32bits). Cada elemento debe valer por tanto 2 elevado (2 por 3) /32 = 2 elevado a 32 /2 elevado ? = ?. ¿Cómo se escribe ese valor en hexadecimal? Inicializar los 32 elementos de la lista con ese valor comprobar cuándo se produce el acarreo.**

Para llegar a esta suma tenemos que obtener la suma de un numero que sumado 32 veces nos de el numero  $2^{32}$ , por lo que el numero sería  $2^{27}=134217728$  en hexadecimal=0x08000000. Al sumar este numero 32 veces el acarreo se produce en el penúltimo elemento de la lista.

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

El valor de la suma es 32212254720, en hexadecimal 0x0780000000.

El acarreo se produce en la quinta interacción o en el momento de la suma del sexto elemento de la lista.

-----Cuestiones de suma64signed.s (tabla 13)-----

**1 ¿Cuál es el máximo entero positivo que puede representarse (escribirlo en hexadecimal)? Si se sumaran los  $N \approx 32$  elementos de la lista inicializados a ese valor ¿qué resultado se obtendría (en hexadecimal)? ¿Qué valor aproximado tienen el elemento y la suma (indicarlo en múltiplos de potencias binarias  $K_i$ ,  $M_i$ ,  $G_i$ )? Comprobarlo usando ddd.**

Ante el ejercicio anterior, ahora el número máximo que podemos representar con 32 bits es la mitad -1, para representar los números positivos y negativos, es decir, el máximo número entero positivo que se puede representar es 2147483647 en hexadecimal 7fffffff.  
La suma de 32 veces este número es 68719476704 en decimal ffffffff0, no podría ser representado en 32 bits.

**2 Misma pregunta respecto a negativos: menor valor negativo en hexadecimal, suma, valores decimales aprox., usar ddd**

El menor negativo es -2147483648 en hexadecimal -80000000  
Su suma -68719476736 en hexadecimal -1000000000

**3 Si nos proponemos obtener sólo 1 acarreo con una lista de 32 elementos positivos iguales, se podría pensar que el objetivo es que la suma alcance 2 elevado a 31 (que ya no cabe en 32bits como número positivo en complemento a dos). Aparentemente, cada elemento debe valer por tanto 2 elevado a 31 / 32 = 2 elevado a 31 / 2 elevado a ? = ?. ¿Cómo se escribe ese valor en hexadecimal? Inicializarlos 32 elementos de la lista con ese valor y comprobar si se produce el acarreo.**

134217728 en hexadecimal 8000000.  
Su suma 4294967296 en hexadecimal 100000000, no se produce acarreo

**4 Repetir el ejercicio anterior de forma que sí se produzca acarreo desde los 32bits inferiores a los superiores. ¿Cuál es el valor del elemento requerido? ¿Por qué es incorrecto el razonamiento anterior? Indicar los valores decimales aproximados (múltiplos de potencias de 10) del elemento y de la suma. Comprobarlo usando ddd.**

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

**6 Por probar valores intermedios: si la lista a se inicializara 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 sería 18446744056529682000 en hexadecimal 0xffffffffc00000000

-----Cuestiones media.s (tabla 14)-----

**1 Rellenando la lista al valor -1, la media es -1. Cambiando un elemento a 0, la media pasa a valer 0. ¿ Porqué? Consultar el manual de Intel sobre la instrucción de división. ¿Cuánto vale el resto de la división en ambos casos? Probarlo con ddd.**

Porque en el caso de ser siempre números negativos, su suma va ser negativa y dividido entre el tamaño de la lista (un numero entero positivo) la media es negativa.

En el caso de que haya un 0 ya es distinto, porque por ejemplo en este caso seria la suma -3 y el tamaño de la lista 4, es decir, que  $-3/4=0$ ,.... por lo que pasa a valer 0

**2 También se obtiene cociente 0 si se cambia lista[0]=1, o si lista[0]=2, ó si lista[0]=3... Comprobarlo con ddd. La siguiente pregunta lógica es hasta cuánto se puede incrementar lista[0] sin que cambie e cociente=0.**

**Para facilitar el cálculo mental, podemos ajustar lista[1]=-2, , y así la suma de todo el array vale lista[0]-32, resultando más fácil calcular el resto. ¿Para qué rango de valores de lista[0 ] se obtiene cociente 0? ¿Cuánto vale el resto a lo largo de ese rango? Comprobar que coinciden los signos del dividendo (suma) y del resto.**

**NOTA: Para evitar el ciclo editar-ensamblar-e enlazar-depurar, se pueden poner un par de breakpoints antes y después de llamar la subrutina que calcula la media. Tras encontrar el primer breakpoint se puede modificar lista[0] con el comando set var lista =<valor>. Pulsar Cont para llegar al segundo o breakpoint y ver en EAX y EDX los resultados retornados por la subrutina (que no debe hacer PUSH/POP EDX ya que no se pretende conservar el valor de EDX sino retornar el resto). Para hacer muchas ejecuciones seguidas, puede merecer la pena (re)utilizar la línea de comandos (run/set var.../cont) en lugar del ratón.**

**3 ¿Para qué rango de valores de lista[0] se obtiene media 1? ¿Cuánto vale el resto en ese rango?**

**Comprobarlo con ddd, y notar que tanto los dividendos como los restos son positivos (el cociente se redondea hacia cero).**

**4 ¿Para qué rango de valores de lista[0] se obtiene media -1? ¿Cuánto vale el resto en ese rango? Comprobarlo con ddd, y notar que tanto los dividendos como los restos son negativos (el cociente se redondea hacia cero).**

### Ejercicio 5.1

Sumar N enteros sin signo de 32bits en una plataforma de 32bits sin perder precisión ( $N \approx 32$ ).

Código Fuente:

```
.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 0x08000000,0x08000000,0x08000000,0x08000000
        #      .int 0x10000000,0x20000000,0x40000000,0x80000000
    .endm
lista: .irpc i,12345678
        linea
    .endr
longlista: .int (.-lista)/4
resultado: .int -1
formato: .ascii "suma = %8u = %08x hex\n\0"

.section .text
main: .global main

    mov $lista, %ebx # El primer elemento de la lista
    mov longlista, %ecx #Guardamos el tamaño de la lista
    call suma
    mov %eax, resultado # En eax tenemos el resultado

    push resultado
    push resultado
    push $formato
    call printf
    add $12, %esp

    mov $1, %eax
    mov $0, %ebx
    int $0x80

suma:
    push %esi
    mov $0, %eax # Ponemos a 0 los registros
    mov $0, %edx
    mov $0, %esi
bucle:
    add (%ebx,%esi,4), %eax #Sumamos el elemento iésimo elemento de la lista y lo
almacenamos en eax
    jnc incrementar # Salto si no hay acarreo
    inc %edx # Incrementamos el registro edx si hay acarreo
```

Práctica 2  
Marcos Avilés Luque 2°C (C2)

incrementar:

```
inc    %esi        # incrementamos el registro esi que es el contador i del bucle for
cmp    %esi,%ecx    # Comparamos el contador con el tamaño de la lista
jne    bucle        # Salto si no son iguales

pop    %esi        # recuperamos memoria de la pila
ret
```

### Ejercicio 5.2

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

Código Fuente:

```
.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 0xF0000000,0xE0000000,0xE0000000,0xF0000000
    .endm
lista: .irpc i,12345678
        linea
    .endr

longlista: .int (.-lista)/4
resultado: .quad -1
formato:
    .ascii "suma = %lld = %llx hex\n\0"

.section .text

main: .global main

    mov    $lista, %ebx
    mov    longlista, %ecx
    call    suma
    mov    %eax, resultado
    mov    %edx, resultado+4

    push    resultado
    push    resultado
    push    $formato
```



## Práctica 2

### Marcos Avilés Luque 2°C (C2)

```
call printf
add $12, %esp
```

```
mov $1, %eax
mov $0, %ebx
int $0x80
```

suma:

```
push %edx
mov $0, %eax
mov $0, %edx
mov $0, %esi
mov $0, %edi
```

bucle:

```
mov (%ebx, %edx, 4), %edi #Movemos la instruccion en el registro edi para ahorrarnos
trabajo de realizar la operacion (%ebx, %edx, 4).
cmp $0, %edi             #Comparamos si el contenido del registro es positivo o negativo
js salto                 # saltamos si signo
add %edi, %eax           # si el numero es positivo sumamos el elemento iésimo de la
lista y lo guardamos en eax
adc $0, %esi             # Sumamos los dos operandos y sumamos 1 si hay acarreo
jmp salto2
```

salto:

```
add %edi, %eax          # Si el numero es negativo sumamos el elemento iésimo de la lista y lo
guardamos en eax
adc $-1, %esi           # Sumamos los dos operandos y sumamos 1 si hay acarreo
```

salto2:

```
inc %edx                #Incrementamos el contador i del bucle
cmp %edx, %ecx
jne bucle
mov %esi, %edx          # despues del bucle movemos lo que hay en esi y lo pasamos a edx
pop %edx
ret
```

### Ejercicio 5.3

Media de N enteros con signo de 32bits en una plataforma de 32bits.

Código Fuente:

```
.section .data
.macro linea
#   .int -1,-2,1,-2
#   .int 1,2,-3,-4,1
#   .int 0x7fffffff,0x7fffffff,0x7fffffff,0x7fffffff
#   .int 0x80000000,0x80000000,0x80000000,0x80000000
#   .int 0xf0000000,0xe0000000,0xe0000000,0xd0000000
#   .int -1
```

Práctica 2  
Marcos Avilés Luque 2°C (C2)

```
# .int 0,-1,-1,-1
# .int 0,-2,-1,-1
# .int 1,-2,-1,-1
# .int 0x08000000,0x08000000,0x08000000,0x08000000
# .int 0xFC000000,0xFC000000,0xfc000000,0xfc000000
# .int 0xF8000000,0xF8000000,0xf8000000,0xf8000000
# .int 0xF0000000,0xE0000000,0xE0000000,0xF0000000
.endm
lista: .irpc i,12345678
        linea
        .endr

longlista: .int (.-lista)/4
resultado: .int -1
formato:
        .ascii "suma = %lld = %llx hex\n\0"

.section .text

main: .global main

        mov $lista, %ebx
        mov longlista, %ecx
        call suma
        mov %eax, resultado
        mov %edx, resultado+4

        push resultado
        push resultado
        push $formato
        call printf
        add $12, %esp

        mov $1, %eax
        mov $0, %ebx
        int $0x80

suma:
        push %edx
        mov $0, %eax
        mov $0, %edx
        mov $0, %esi
        mov $0, %edi

bucle:
        mov (%ebx, %edx, 4), %edi #Movemos la instruccion en el registro edi para ahorrarnos
trabajo de realizar la operacion (%ebx, %edx, 4).
        cmp $0, %edi             #Comparamos si el contenido del registro es positivo o negativo
        js salto                 # saltamos si signo
```

Práctica 2  
Marcos Avilés Luque 2°C (C2)

```
        add %edi, %eax          # si el numero es positivo sumamos el elemento iésimo de la
lista y lo guardamos en eax
        adc $0, %esi           # Sumamos los dos operandos y sumamos 1 si hay acarreo
        jmp salto2
```

salto:

```
        add %edi, %eax          # Si el numero es negativo sumamos el elemento iésimo de la lista y lo
guardamos en eax
        adc $-1, %esi          # Sumamos los dos operandos y sumamos 1 si hay acarreo
```

salto2:

```
        inc %edx                #Incrementamos el contador i del bucle
        cmp %edx, %ecx
        jne bucle
        mov %esi, %edx          # despues del bucle movemos lo que hay en esi y lo pasamos a edx
        idiv %ecx               # Idiv realiza la division entera con signo
        pop %edx
        ret
```