

# Memoria: Estructura de Computadores

David Martínez Díaz GII-ADE

Titulación: Doble Grado en Ingeniería Informática y Administración y Dirección de Empresas.

**- Enunciado ejercicio 5.1: Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits usando uno de ellos como acumulador de acarreo (N≈16).**

**- Solución del ejercicio:**

```
.section .data
lista: .int 0x10000000,0x10000000,0x10000000,0x10000000
       .int 0x10000000,0x10000000,0x10000000,0x10000000
       .int 0x10000000,0x10000000,0x10000000,0x10000000
       .int 0x10000000,0x10000000,0x10000000,0x10000000
longlista: .int (.-lista)/4
resultado: .quad 0 # cambiamos de int(x32) a quad(x64)
formato: .asciz "suma = %lu = 0x%x hex\n"

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

    call trabajar # subrutina de usuario
    call imprim_C # printf() de libc
    call acabar_C # exit() de libc
    ret
trabajar:
    mov $lista, %rbx
    mov longlista, %ecx
    call suma # == suma(&lista, longlista);
    mov %eax, resultado
    mov %edx, resultado + 4 # 4 posiciones detras de resultado
    ret
suma:

    mov $0, %rsi # contador de la suma
    mov $0, %eax # suma
    mov $0, %edx # acumulador de acarreo

bucle:
    add (%rbx,%rsi,4), %eax # %eax += 4 * %rsi + %rbx
    jnc no_acarreo
    inc %edx

no_acarreo:
    inc %rsi
    cmp %rsi,%rcx
    jne bucle

    ret

imprim_C: # requiere libc
    mov $formato, %rdi
    mov resultado,%rsi
    mov resultado,%rdx
    mov $0,%eax # varargin sin xmm
    call printf # == printf(formato, res, res);
    ret

acabar_C: # requiere libc
    mov resultado, %rdi
    call _exit # == exit(resultado)
    ret

acabar_C: # requiere libc
    mov resultado, %edi
    call _exit # == exit(resultado)
    ret
```

**- Pruebas del ejercicio:**

\* En el primer caso utilizaremos los valores (16 veces) 0x10000000, donde el resultado es 4294967296 como podemos ver en la siguiente captura:

```
Starting program: /home/dmartinez01/Escritorio/2 curso/Practicas EC/Practicas/Practica 2/Ejercicio5.1/Ejercicio5-1
suma = 4294967296 = 0x100000000 hex
[Inferior 1 (process 4036) exited normally]
(gdb)
```

\* En el segundo caso utilizaremos los valores 3,2,1,10,12, donde el resultado es 28 como podemos ver en la siguiente captura:

```
Starting program: /home/dmartinez01/Escritorio/2 curso/Practicas EC/Practicas/Practica 2/Ejercicio5.1/Ejercicio5-1
suma = 28 = 0x1c hex
[Inferior 1 (process 4130) exited with code 034]
(gdb)
```

\* En el tercer caso utilizaremos los valores 0x10000000,10,12 donde el resultado es 22 como podemos ver en la siguiente captura:

```
Starting program: /home/dmartinez01/Escritorio/2 curso/Practicas EC/Practicas/Practica 2/Ejercicio5.1/Ejercicio5-1
suma = 268435478 = 0x10000016 hex
[Inferior 1 (process 4143) exited with code 026]
(gdb)
```

\* En el tercer caso utilizaremos los valores 3,4,7,9 donde el resultado es 22 como podemos ver en la siguiente captura:

```
Starting program: /home/dmartinez01/Escritorio/2 curso/Practicas EC/Practicas/Practica 2/Ejercicio5.1/Ejercicio5-1
suma = 23 = 0x17 hex
[Inferior 1 (process 2840) exited with code 027]
(gdb)
```

**Enunciado ejercicio 5.2: Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits mediante extensión con ceros ( $N \approx 16$ ).**

**- Solución del ejercicio:**

```
.section .data

#ifndef TEST
#define TEST 9
#endif

.macro linea
    #if TEST==1
        .int 1, 1, 1, 1
    #elif TEST==2
        .int 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff
    #elif TEST==3
        .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
    #elif TEST==4
        .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
    #elif TEST==5
        .int -1, -1, -1, -1
    #elif TEST==6
        .int 200000000, 200000000, 200000000, 200000000
    #elif TEST==7
        .int 300000000, 300000000, 300000000, 300000000
    #elif TEST==8
        .int 500000000, 500000000, 500000000, 500000000
    #else
        .error "Definir TEST entre 1..8"
    #endif
.endm

lista: .irpc i, 1234
        linea
    .endr
longlista: .int (.-lista)/4
resultado: .quad 0 # cambiamos de int(x32) a quad(x64)
formato: .ascii "Suma \t = %18lu (uns) \n"
        .ascii "\t\t = 0x%18lx (hex)\n"
        .asciz "\t\t = 0x %08x %08x\n"

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

    call trabajar # subrutina de usuario
    call imprim_C # printf() de libc
    call acabar_C # exit() de libc
    ret

trabajar:
    mov $lista, %rbx
    mov longlista, %ecx
    call suma # == suma(&lista, longlista);
    mov %eax, resultado
    mov %edx, resultado + 4 # 4 posiciones detras de resultado
    ret

suma:
    mov $0, %rsi # contador de la suma
    mov $0, %eax # suma
    mov $0, %edx # acumulador de acarreo

bucle:
    add (%rbx,%rsi,4), %eax # %eax += 4 * %rsi + %rbx
    adc $0, %edx
    inc %rsi
    cmp %rsi,%rcx
    jne bucle
    ret

imprim_C: # requiere libc
    mov $formato, %rdi
    mov resultado,%rsi
    mov resultado,%rdx
    mov resultado,%rcx
    mov $0,%eax # varargin sin xmm
    call printf # == printf(formato, res, res);
    ret

acabar_C: # requiere libc
    mov resultado, %rdi
    call _exit # == exit(resultado)
    ret
```

**- Pruebas del ejercicio:**

\*Para los casos utilizaremos los siguientes:

- 1.- .int 1, 1, 1, 1
- 2.- .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
- 3.- .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
- 4.- .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
5. .int -1, -1, -1, -1
6. .int 200000000, 200000000, 200000000, 200000000
7. .int 300000000, 300000000, 300000000, 300000000
8. .int 500000000, 500000000, 500000000, 500000000

```

T#1 Suma      =      16 (uns)
              = 0x      10 (hex)
              = 0x 00000010 ccd70d80
T#2 Suma      =      4294967280 (uns)
              = 0x      ffffffff0 (hex)
              = 0x ffffffff0 8adb4d80
T#3 Suma      =      4294967296 (uns)
              = 0x      100000000 (hex)
              = 0x 00000000 4b3f8d80
T#4 Suma      =      68719476720 (uns)
              = 0x      ffffffff0 (hex)
              = 0x ffffffff0 6d294d80
T#5 Suma      =      68719476720 (uns)
              = 0x      ffffffff0 (hex)
              = 0x ffffffff0 f8522d80
T#6 Suma      =      3200000000 (uns)
              = 0x      bebc2000 (hex)
              = 0x bebc2000 2c954d80
T#7 Suma      =      4800000000 (uns)
              = 0x      11e1a3000 (hex)
              = 0x 1e1a3000 a161dd80
T#8 Suma      =      8000000000 (uns)
              = 0x      1dcd65000 (hex)
              = 0x dcd65000 cb237d80
```

**Enunciado ejercicio 5.3: Sumar N enteros con signo de 32 bits sobre dos registros de 32 bits (mediante extensión de signo, naturalmente) (N≈16)**

**- Solución del ejercicio:**

```
.section .data

longlista: .int    (.-lista)/4
resultado: .quad    0      # cambiamos de int(x32) a quad(x64)
formato:   .ascii  "resultado \t = %18lld (sgn)\n"
           .ascii  "\t\t = 0x%18llx (hex)\n"
           .asciz   "\t\t = 0x  %08x %08x\n"

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

    call trabajar    # subrutina de usuario
    call imprim_C    # printf() de libc
    call acabar_C    # exit() de libc
    ret

trabajar:
    mov     $lista, %rbx
    mov     longlista, %ecx
    call    suma      # == suma(&lista, longlista);
    mov     %eax, resultado
    mov     %edx, resultado + 4    # 4 posiciones detras de resultado
    ret

suma:
    mov     $0, %rsi # contador de la suma
    mov     $0, %eax # suma
    mov     $0, %edx # acumulador de acarreo
    mov     $0, %r8d
    mov     $0, %r9d

bucle:

    mov     (%rbx,%rsi,4), %eax
    cld

    add     %eax, %r8d
    adc     %edx, %r9d

    inc     %rsi
    cmp     %rsi,%rcx
    jne     bucle

    mov     %r8d, %eax
    mov     %r9d, %edx

    ret
imprim_C: # requiere libc
    mov     $formato, %rdi
    mov     resultado,%rsi
    mov     resultado,%rdx
    mov     resultado + 4,%rcx
    mov     $0,%eax    # varargin sin xmm
    call    printf      # == printf(formato, res, res);
    ret

acabar_C: # requiere libc
    mov     resultado, %rdi
    call    _exit        # == exit(resultado)
    ret
```

### Pruebas del ejercicio:

\*Para los casos utilizaremos los siguientes:

- 1.- .int 1, 1, 1, 1
- 2.- .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
- 3.- .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
- 4.- .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
5. .int -1, -1, -1, -1
6. .int 200000000, 200000000, 200000000, 200000000
7. .int 300000000, 300000000, 300000000, 300000000
8. .int 500000000, 500000000, 500000000, 500000000
9. .int 0xf7ffffff, 0xf7ffffff, 0xf7ffffff, 0xf7ffffff
10. .int 100000000, 100000000, 100000000, 100000000

```
dmartinez01@dmartinez01-VirtualBox:~/Escritorio/2 curso/P
T#1 resultado      =          -16 (sgn)
                  = 0x  ffffffff00000000 (hex)
                  = 0x  ffffffff 00000000
T#2 resultado      =          1073741824 (sgn)
                  = 0x      40000000 (hex)
                  = 0x  00000000 40000000
T#3 resultado      =          2147483648 (sgn)
                  = 0x      80000000 (hex)
                  = 0x  00000000 80000000
T#4 resultado      =          4294967296 (sgn)
                  = 0x     100000000 (hex)
                  = 0x  00000001 00000000
T#5 resultado      =          34359738352 (sgn)
                  = 0x      7fffffff0 (hex)
                  = 0x  00000007 ffffffff0
T#6 resultado      =         -34359738368 (sgn)
                  = 0x  ffffffff800000000 (hex)
                  = 0x  ffffffff8 00000000
T#7 resultado      =         -4294967296 (sgn)
                  = 0x  ffffffff000000000 (hex)
                  = 0x  ffffffff 00000000
T#8 resultado      =         -2147483648 (sgn)
                  = 0x  ffffffff800000000 (hex)
                  = 0x  ffffffff 80000000
T#9 resultado      =         -2147483664 (sgn)
                  = 0x  ffffffff7fffffff0 (hex)
                  = 0x  ffffffff 7fffffff0
T#10 resultado     =          1600000000 (sgn)
                  = 0x      5f5e1000 (hex)
                  = 0x  00000000 5f5e1000
```

**Enunciado ejercicio 5.4: Media y resto de N enteros con signo de 32 bits calculada usando registros de 32 bits (N≈16)**

**- Solución del ejercicio:**

```
.section .data
lista: .int 2, 4, 6, 8
longlista: .int (.lista)/4

resultado: .quad 0
formato: .ascii "Media (decimal) = %lld \t Resto (decimal) = %lld\n"
        .ascii "Media (Hexadecimal) = [0x %08x] \t Resto (Hexadecimal)= [0x %08x]\n"
cociente: .int 0
resto: .int 0
.section .text
#_start: global _start
main: global main

        call trabajar      # subrutina de usuario
        call imprim_C      # printf() de libC
        call acabar_C      # exit() de libC
        ret

trabajar:
        mov $lista, %edi #Mueve la primera posición de memoria lista al registro %edi
        mov $longlista, %esi #Mueve la longitud de la lista al registro %ecx
        call suma          # == suma(&lista, longlista);

        call media
        mov %eax, cociente
        mov %edx, resto
imprim_C:

        mov $formato, %edi
        mov cociente, %esi
        mov resto, %edx
        mov %eax, %ecx
        mov %edx, %ebx
        mov $0, %eax
        call printf        # == printf(formato, res, res);

acabar_C:          # requiere libC

        #mov resultado, %edi
        movl $1, %eax
        xor %ebx, %ebx
        int $0x80 # int $0x80, en caso de que %eax valga 1, termina ejecución y retorna %ebx

suma:
        mov $0, %ecx # inicializo el índice
        mov $0, %r8d # acumulador (no significativo)
        mov $0, %r9d # acumulador (significativo)
        mov $0, %edx
        mov $0, %rax

bucle:
        mov (%edi,%ecx,4), %eax #eax=Lista[i]
        clt # cll // Coge eax y lo amplía a edx con el signo.
        add %eax, %r8d
        adc %edx, %r9d

        inc %ecx # incrementar el índice
        cmp %esi, %ecx # comparar índice con longitud
        jge bucle

        mov %r8d, %eax # Una vez acabado el bucle retornar los acumuladores %eax y %edx
        mov %r9d, %edx
        mov $0, %r8d
        mov $0, %r9d
        ret # retornar

media:
        idiv %esi # Siempre actúa en edx:eax
        ret
```

**- Pruebas del ejercicio:**

\*Para los casos utilizaremos los siguientes:

```
Resto (Hexadecimal)= [0x 00000000] Resto (Hexadecimal)= [0x 00000000]
dmartinez01@dmartinez01-VirtualBox:~/Escritorio/2 curso/Practicas EC/Practicas/Practica 2/Ejercicio5.4$ ./ejecucion.sh
T#1 Media (decimal) = 1 Resto (decimal) = 8
Media (Hexadecimal) = [0x 00000001] Resto (Hexadecimal)= [0x 00000008]
T#2 Media (decimal) = -1 Resto (decimal) = -8
Media (Hexadecimal) = [0x ffffffff] Resto (Hexadecimal)= [0x ffffffff8]
T#3 Media (decimal) = 2147483647 Resto (decimal) = 0
Media (Hexadecimal) = [0x 7fffffff] Resto (Hexadecimal)= [0x 00000000]
T#4 Media (decimal) = -2147483648 Resto (decimal) = 0
Media (Hexadecimal) = [0x 80000000] Resto (Hexadecimal)= [0x 00000000]
T#5 Media (decimal) = -1 Resto (decimal) = 0
Media (Hexadecimal) = [0x ffffffff] Resto (Hexadecimal)= [0x 00000000]
T#6 Media (decimal) = 2000000000 Resto (decimal) = 0
Media (Hexadecimal) = [0x 77359400] Resto (Hexadecimal)= [0x 00000000]
T#7 Media (decimal) = -1294967296 Resto (decimal) = 0
Media (Hexadecimal) = [0x b2d05e00] Resto (Hexadecimal)= [0x 00000000]
T#8 Media (decimal) = -2000000000 Resto (decimal) = 0
Media (Hexadecimal) = [0x 88ca6c00] Resto (Hexadecimal)= [0x 00000000]

T#9 Media (decimal) = 1294967296 Resto (decimal) = 0
Media (Hexadecimal) = [0x 4d2fa200] Resto (Hexadecimal)= [0x 00000000]
T#10 Media (decimal) = 1 Resto (decimal) = 0
Media (Hexadecimal) = [0x 00000001] Resto (Hexadecimal)= [0x 00000000]
T#11 Media (decimal) = 1 Resto (decimal) = 4
Media (Hexadecimal) = [0x 00000001] Resto (Hexadecimal)= [0x 00000004]
T#12 Media (decimal) = 3 Resto (decimal) = 0
Media (Hexadecimal) = [0x 00000003] Resto (Hexadecimal)= [0x 00000000]
T#13 Media (decimal) = 4 Resto (decimal) = 12
Media (Hexadecimal) = [0x 00000004] Resto (Hexadecimal)= [0x 0000000c]
T#14 Media (decimal) = 5 Resto (decimal) = 0
Media (Hexadecimal) = [0x 00000005] Resto (Hexadecimal)= [0x 00000000]
T#15 Media (decimal) = 0 Resto (decimal) = 0
Media (Hexadecimal) = [0x 00000000] Resto (Hexadecimal)= [0x 00000000]
T#16 Media (decimal) = -1 Resto (decimal) = -4
Media (Hexadecimal) = [0x ffffffff] Resto (Hexadecimal)= [0x ffffffff4]
T#17 Media (decimal) = -3 Resto (decimal) = 0
Media (Hexadecimal) = [0x ffffffff] Resto (Hexadecimal)= [0x 00000000]
T#18 Media (decimal) = -4 Resto (decimal) = -12
Media (Hexadecimal) = [0x ffffffff] Resto (Hexadecimal)= [0x ffffffff4]
T#19 Media (decimal) = -5 Resto (decimal) = 0
Media (Hexadecimal) = [0x ffffffff] Resto (Hexadecimal)= [0x 00000000]
```