

Estructura de Computadores: Práctica 2

Programación Ensamblador x86-64 Linux

Pablo Olivares Martínez

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).

Primero, realicemos la elaboración del programa:

```
# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
lista:      .int 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
longlista:  .int  (-lista)/4  # . = contador posiciones. Aritm.etiq.
resultado:  .quad  0

# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text          # PROGRAMA PRINCIPAL
_start: .global _start  # Programa principal

    call trabajar        # subrutina de usuario
    call acabar_L        # finaliza el programa
    ret

trabajar:
    mov     $lista, %rbx  # dirección del array lista
    mov     longlista, %ecx # número de elementos a sumar
    call suma             # == suma(&lista, longlista);
    mov     %eax, resultado # salvar resultado
    mov     %edx, resultado+4 # incluye el acarreo
    ret

# SUBROUTINA: int suma(int* lista, int longlista);
# entrada:  1) %rbx = dirección inicio array
#           2) %ecx = número de elementos a sumar
# salida:   %eax = resultado de la suma
suma:
    mov     $0, %edx      # poner a 0 acarreo
    mov     $0, %eax      # poner a 0 acumulador
    mov     $0, %rsi      # poner a 0 índice
bucle:
    add     (%rbx,%rsi,4), %eax # acumular i-ésimo elemento
    jnc     incremento      #
    inc     %edx             #
incremento:
    inc     %rsi             # incrementar índice
    cmp     %rsi,%rcx        # comparar con longitud
    jne     bucle            # si no iguales, seguir acumulando
    ret

acabar_L:
    mov     $60, %rax
```

```

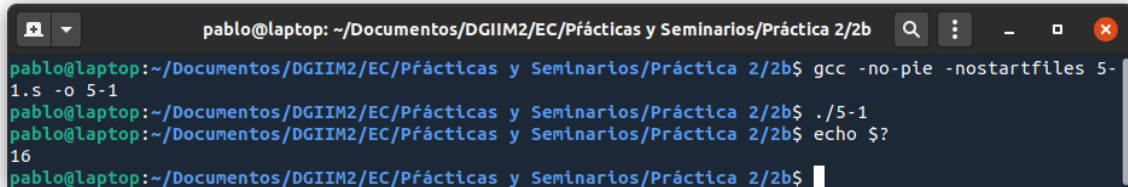
mov resultado, %edi
syscall      # == _exit(resultado)
ret

```

Para compilarlo, escribamos el siguiente comando en bash:

```
gcc -no-pie -nostartfiles 5-1.s -o 5-1
```

Tras la compilación y ejecución del programa obtendremos esto, donde 16 es la solución y es correcta:



```

pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ gcc -no-pie -nostartfiles 5-1.s -o 5-1
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ ./5-1
16
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ echo $?
16
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$

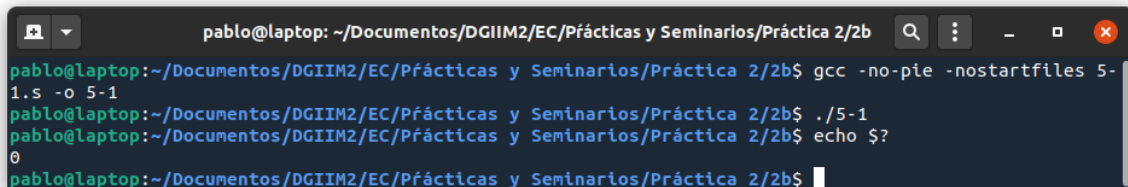
```

A continuación, modifiquemos los valores de la lista para probar si el algoritmo es correcto para distintas pruebas donde si que haya acarreo:

```

.section .data
lista:      .int 0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000,
0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000,
0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000
longlista:  .int (.-lista)/4

```



```

pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ gcc -no-pie -nostartfiles 5-1.s -o 5-1
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ ./5-1
0
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ echo $?
0
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$

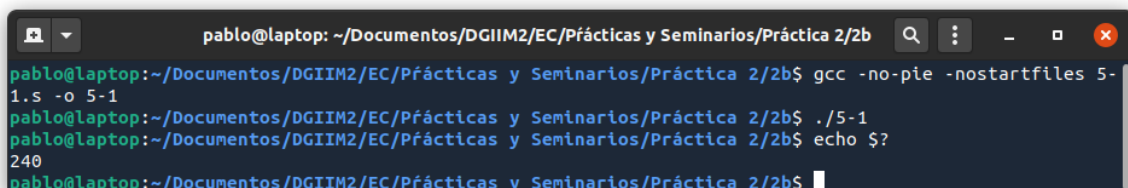
```

Como hemos podido ver, al probar con el valor 0x1000 0000 nos damos cuenta que la solución no es correcta. Este valor es el primero que hace uso del acarreo, por ello si probáramos el anterior (0x0fff ffff), debería de ser una operación correcta:

```

.section .data
lista:      .int 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff,
0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff,
0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff
longlista:  .int (.-lista)/4

```



```

pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ gcc -no-pie -nostartfiles 5-1.s -o 5-1
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ ./5-1
240
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ echo $?
240
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$

```

Sin embargo, al obtener el resultado vemos que no es correcto. Si depuramos el programa con gdb, nos damos cuenta rápidamente de que el valor que debería devolver lo calcula correctamente (0xffff fff0), estando el problema en la impresión del valor. Por ello, vamos a modificar el programa para poder usar la impresión de *libc* y así solucionar este problema. Además le añadiremos un formato de salida:

```
# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
lista:      .int  0xffffffff,0xffffffff,0xffffffff,0xffffffff,
0xffffffff,0xffffffff,0xffffffff,0xffffffff,
0xffffffff,0xffffffff,0xffffffff,0xffffffff,
0xffffffff,0xffffffff,0xffffffff,0xffffffff
longlista:  .int  (-lista)/4  # . = contador posiciones. Aritm.etiq.
resultado:  .quad  0

formato:    .asciz  "suma = %lu = 0x%x hex\n"  # fmt para printf() libc

# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text      # PROGRAMA PRINCIPAL
main: .global  main      # Programa principal si se usa C runtime

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

trabajar:
    mov     $lista, %rbx    # dirección del array lista
    mov     longlista, %ecx  # número de elementos a sumar
    call suma              # == suma(&lista, longlista);
    mov     %eax, resultado  # salvar resultado
    mov     %edx, resultado+4 # incluye el acarreo
    ret

# SUBROUTINA: int suma(int* lista, int longlista);
# entrada:  1) %rbx = dirección inicio array
#           2) %ecx = número de elementos a sumar
# salida:   %eax = resultado de la suma
suma:
    mov     $0, %edx        # poner a 0 acarreo
    mov     $0, %eax        # poner a 0 acumulador
    mov     $0, %rsi        # poner a 0 índice
bucle:
    add     (%rbx,%rsi,4), %eax  # acumular i-ésimo elemento
    jnc     incremento        # hace el incremento
    inc     %edx              # si no hay acarreo, se salta este inc
incremento:
    inc     %rsi              # incrementar índice
    cmp     %rsi,%rcx         # comparar la longitud
    jne     bucle             # si no iguales, seguir acumulando
    ret

imprim_C:
    # requiere libc, void
    mov     $formato, %rdi    # traduce resultado a decimal/hex
    mov     resultado,%rsi    # versión libc de syscall __NR_write
    mov     resultado,%rdx    # ventaja: printf() con fmt "%u" / "%x"
    mov     $0,%eax           # varargin sin xmm
```

```

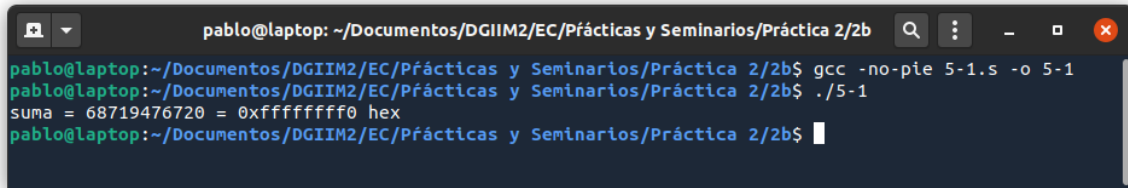
    call    printf                # == printf(formato, res, res);
    ret

acabar_C:                        # requiere libC, void exit(int status);
    mov     resultado, %edi       # status: código a retornar (la suma)
    call    _exit                # == exit(resultado)
    ret

```

Compilamos el programa y lo ejecutamos:

```
gcc -no-pie 5-1.s -o 5-1
```



```

pablo@laptop: ~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ gcc -no-pie 5-1.s -o 5-1
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ ./5-1
suma = 68719476720 = 0xffffffff0 hex
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$

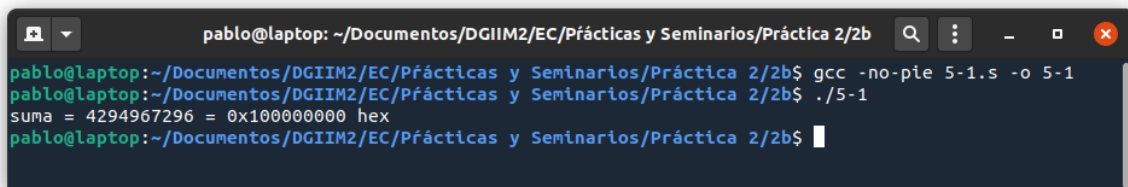
```

Como hemos podido ver, este cambio ha solucionado el problema. Ahora veamos el anterior ejemplo y otro más para mostrar el correcto funcionamiento de acarreo:

```

.section .data
lista:      .int 0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000,
0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000,
0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000
longlista:  .int (.-lista)/4
resultado:  .quad 0
formato:    .asciz "suma = %lu = 0x%lx hex\n"

```



```

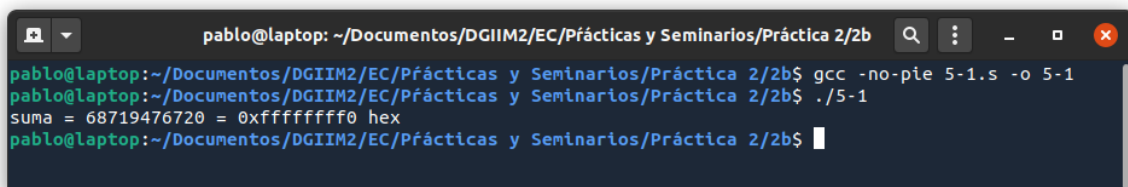
pablo@laptop: ~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ gcc -no-pie 5-1.s -o 5-1
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ ./5-1
suma = 4294967296 = 0x100000000 hex
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$

```

```

.section .data
lista:      .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
longlista:  .int (.-lista)/4
resultado:  .quad 0
formato:    .asciz "suma = %lu = 0x%lx hex\n"

```



```

pablo@laptop: ~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ gcc -no-pie 5-1.s -o 5-1
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ ./5-1
suma = 68719476720 = 0xffffffff0 hex
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$

```

Efectivamente, estos son los resultados esperados.

Ejercicio 5.2 - Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits mediante extensión con ceros (N≈16).

Realicemos ahora el programa:

```
# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
lista:      .int 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
longlista:  .int  (-lista)/4 # . = contador posiciones. Aritm.etiq.
resultado:  .quad  0

formato:    .asciz "suma = %lu = 0x%x hex\n" # fmt para printf() libc

# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text # PROGRAMA PRINCIPAL
main: .global main # Programa principal si se usa C runtime

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

trabajar:
    mov     $lista, %rbx # dirección del array lista
    mov     longlista, %ecx # número de elementos a sumar
    call suma # == suma(&lista, longlista);
    mov     %eax, resultado # salvar resultado
    mov     %edx, resultado+4 # incluye el acarreo
    ret

# SUBROUTINA: int suma(int* lista, int longlista);
# entrada:  1) %rbx = dirección inicio array
#           2) %ecx = número de elementos a sumar
# salida:   %eax = resultado de la suma
suma:
    mov     $0, %edx # poner a 0 acarreo
    mov     $0, %eax # poner a 0 acumulador
    mov     $0, %rsi # poner a 0 índice
bucle:
    add     (%rbx,%rsi,4), %eax # acumular i-ésimo elemento
    adc     $0, %edx # añade el acarreo
    inc     %rsi # incrementa el índice
    cmp     %rsi, %rcx # compara si son iguales
    jne     bucle # si no lo son, vuelve al bucle

    ret

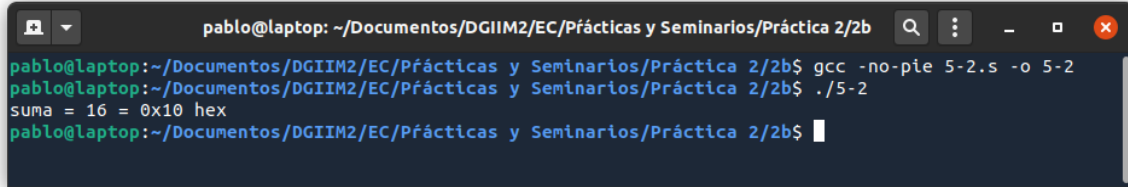
imprim_C: # requiere libc, void
    mov     $formato, %rdi # traduce resultado a decimal/hex
    mov     resultado,%rsi # versión libc de syscall __NR_write
    mov     resultado,%rdx # ventaja: printf() con fmt "%u" / "%x"
    mov     $0,%eax # varargin sin xmm
    call    printf # == printf(formato, res, res);
    ret
```

```

acabar_C:                                # requiere libC, void exit(int status);
    mov     resultado, %edi               # status: código a retornar (la suma)
    call    _exit                        # == exit(resultado)
    ret

```

Comprobemos si funciona con varios ejemplos con y sin acarreo:



```

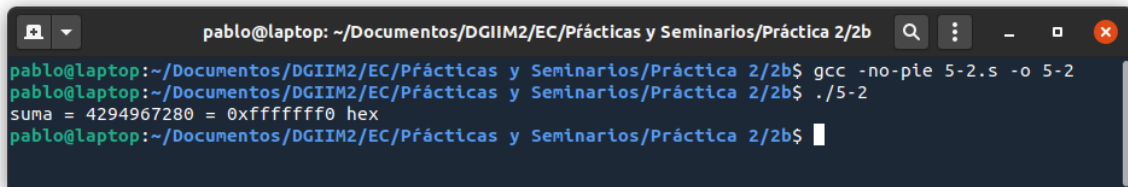
pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ gcc -no-pie 5-2.s -o 5-2
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ ./5-2
suma = 16 = 0x10 hex
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$

```

```

.section .data
lista:      .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
longlista:  .int (.-lista)/4

```



```

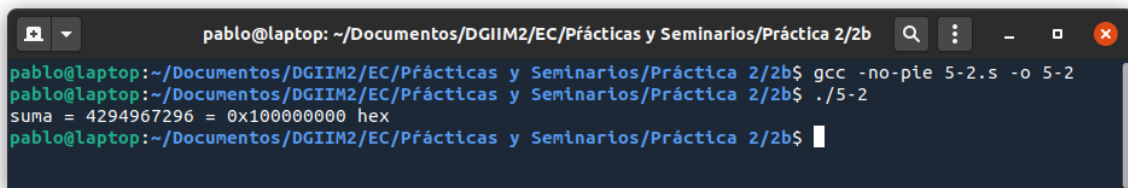
pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ gcc -no-pie 5-2.s -o 5-2
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ ./5-2
suma = 4294967280 = 0xffffffff0 hex
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$

```

```

.section .data
lista:      .int 0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000,
0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000,
0x10000000, 0x10000000, 0x10000000, 0x10000000, 0x10000000
longlista:  .int (.-lista)/4

```



```

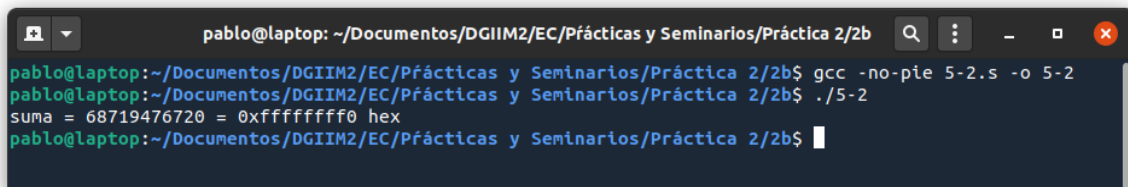
pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ gcc -no-pie 5-2.s -o 5-2
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ ./5-2
suma = 4294967296 = 0x100000000 hex
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$

```

```

.section .data
lista:      .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff,
0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
longlista:  .int (.-lista)/4

```



```

pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ gcc -no-pie 5-2.s -o 5-2
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$ ./5-2
suma = 68719476720 = 0xfffffffff0 hex
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$

```

Como hemos podido comprobar, todos los ejemplos funcionan correctamente. Sin embargo, estar probando constantemente distintas listas de esta forma es muy ineficiente. Por ello, vamos a crear un script y una batería de tests las cuales se van a encargar de hacer las pruebas de golpe y ahorrándonos mucho tiempo:

```
for i in $(seq 1 8); do
    rm 5-2
    gcc -x assembler-with-cpp -D TEST=$i -no-pie $1 -o 5-2
    printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./5-2
done
```

```
.section .data
#ifndef TEST
#define TEST 8
#endif

.macro linea
#if TEST==1
.int 1,1,1,1
#elif TEST==2
.int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
#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 5000000000, 5000000000, 5000000000, 5000000000
#else
.error "Definir TEST entre 1..8"
#endif
.endm

lista: .irpc i,1234
    linea
.endr

longlista: .int    (-lista)/4 # . = contador posiciones. Aritm.etiq.
resultado: .quad    0

formato:
.ascii "resultado \t = %18lu (uns)\n"
.ascii "\t\t = 0x%18lx (hex)\n"
.asciz "\t\t = 0x %08x %08x\n"
```

Se implementaría de esta forma y para probarlo, bastaría con ejecutar el script:

```
./run_tests.sh
```

```
pablo@laptop: ~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ ./run_tests.sh 5-2.s
__TEST01__-----
resultado          =          16 (uns)
                   = 0x          10 (hex)
                   = 0x 00000010 00000000
__TEST02__-----
resultado          =      4294967280 (uns)
                   = 0x      ffffffff (hex)
                   = 0x 00000010 00000000
__TEST03__-----
resultado          =      4294967296 (uns)
                   = 0x      10000000 (hex)
                   = 0x 00000010 00000000
__TEST04__-----
resultado          =      68719476720 (uns)
                   = 0x      ffffffff0 (hex)
                   = 0x 00000010 00000000
__TEST05__-----
resultado          =      68719476720 (uns)
                   = 0x      ffffffff0 (hex)
                   = 0x 00000010 00000000
__TEST06__-----
resultado          =      3200000000 (uns)
                   = 0x      bebc2000 (hex)
                   = 0x 00000010 00000000
__TEST07__-----
resultado          =      4800000000 (uns)
                   = 0x      11e1a3000 (hex)
                   = 0x 00000010 00000000
5-2.s: Mensajes del ensamblador:
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
5-2.s:34: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
__TEST08__-----
resultado          =      11280523264 (uns)
                   = 0x      2a05f2000 (hex)
                   = 0x 00000010 00000000
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$
```

Como podemos observar, todo ha funcionado correctamente. No obstante, no vendría mal resaltar ese aviso, que nos muestra que se han tenido que truncar números ya que no cabían en un registro de 32 bits. Además de que el TEST05 es incorrecto, pues es con números negativos, pero es un caso que solucionaremos en el siguiente apartado.

Ejercicio 5.3 - Sumar N enteros con signo de 32 bits sobre dos registros de 32 bits (mediante extensión de signo, naturalmente) (N≈16).

Realizamos el programa y modifiquemos el script para los nuevos tests:

```
# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
#ifdef TEST
#define TEST 19
#endif

    .macro linea
    #if TEST==1
        .int -1 ,-1 ,-1, -1
    #elif TEST==2
        .int 0x04000000, 0x04000000, 0x04000000, 0x04000000
    #elif TEST==3
```



```

        .int 0x08000000, 0x08000000, 0x08000000, 0x08000000
#elif TEST==4
        .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
#elif TEST==5
        .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
#elif TEST==6
        .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
#elif TEST==7
        .int 0xF0000000, 0xF0000000, 0xF0000000, 0xF0000000
#elif TEST==8
        .int 0xF8000000, 0xF8000000, 0xF8000000, 0xF8000000
#elif TEST==9
        .int 0xF7FFFFFF, 0xF7FFFFFF, 0xF7FFFFFF, 0xF7FFFFFF
#elif TEST==10
        .int 100000000, 100000000, 100000000, 100000000
#elif TEST==11
        .int 200000000, 200000000, 200000000, 200000000
#elif TEST==12
        .int 300000000, 300000000, 300000000, 300000000
#elif TEST==13
        .int 2000000000, 2000000000, 2000000000, 2000000000
#elif TEST==14
        .int 3000000000, 3000000000, 3000000000, 3000000000
#elif TEST==15
        .int -1000000000, -1000000000, -1000000000, -1000000000
#elif TEST==16
        .int -2000000000, -2000000000, -2000000000, -2000000000
#elif TEST==17
        .int -3000000000, -2000000000, -2000000000, -2000000000
#elif TEST==18
        .int -2000000000, -2000000000, -2000000000, -2000000000
#elif TEST==19
        .int -3000000000, -3000000000, -3000000000, -3000000000
#else
        .error "Definir TEST entre 1..19"
#endif

.endm

lista: .irpc i,1234
        linea
        .endr

longlista: .int    (-lista)/4  # . = contador posiciones. Aritm.etiq.
resultado: .quad    0

formato:
        .ascii "resultado \t = %18ld (sgn)\n"
        .ascii "\t\t = 0x%18lx (hex)\n"
        .asciz "\t\t = 0x %08x %08x\n"

# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text                # PROGRAMA PRINCIPAL
main: .global main            # Programa principal si se usa C runtime

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

```

```

trabajar:
    mov     $lista, %rbx    # dirección del array lista
    mov     longlista, %ecx  # número de elementos a sumar
    call    suma            # == suma(&lista, longlista);
    mov     %eax, resultado  # salvar resultado
    mov     %edx, resultado+4 # incluye el acarreo
    ret

# SUBROUTINA: int suma(int* lista, int longlista);
# entrada:  1) %rbx = dirección inicio array
#           2) %ecx = número de elementos a sumar
# salida:    %eax = resultado de la suma
suma:
    mov     $0, %edi        # ponemos a 0 los registros a usar
    mov     $0, %rsi
    mov     $0, %r8d
bucle:
    mov     (%rbx,%rsi,4), %eax # mueve el siguiente elemento de la lista
    cltd                      # extiende el signo de %eax
    add     %eax, %edi         # suma %eax a %edi
    adc     %edx, %r8d         # hace la suma con el acarreo
    inc     %rsi              # incrementa el índice
    cmp     %rsi,%rcx         # compara si son iguales
    jne     bucle             # si no lo, vuelve a hacer el bucle

    mov     %edi, %eax        # mueve la suma a %eax
    mov     %r8d, %edx        # mueve el acarreo a %edx

    ret

    ret

imprim_C:                    # requiere libC, void
    mov     $formato, %rdi    # traduce resultado a decimal/hex
    mov     resultado,%rsi    # versión libC de syscall __NR_write
    mov     resultado,%rdx    # ventaja: printf() con fmt "%u" / "%x"
    mov     $0,%eax           # varargin sin xmm
    call    printf            # == printf(formato, res, res);
    ret

acabar_C:                    # requiere libC, void exit(int status);
    mov     resultado, %edi    # status: código a retornar (la suma)
    call    _exit             # == exit(resultado)
    ret

```

```

for i in $(seq 1 19); do
rm 5-3;
gcc -x assembler-with-cpp -D TEST=$i 5-3.s -no-pie -o 5-3;
printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./5-3;
done

```

```
pablo@laptop: ~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$ ./run_tests.sh
rm: no se puede borrar '5-3': No existe el archivo o el directorio
__TEST01__
resultado      =      4294967280 (sgn)
               = 0x      ffffffff (hex)
               = 0x 00000010 ffffffff
__TEST02__
resultado      =      1073741824 (sgn)
               = 0x      40000000 (hex)
               = 0x 00000010 00000000
__TEST03__
resultado      =      2147483648 (sgn)
               = 0x      80000000 (hex)
               = 0x 00000010 00000000
__TEST04__
resultado      =              0 (sgn)
               = 0x              0 (hex)
               = 0x 00000010 00000001
__TEST05__
resultado      =      4294967280 (sgn)
               = 0x      ffffffff (hex)
               = 0x 00000010 00000007
__TEST06__
resultado      =              0 (sgn)
               = 0x              0 (hex)
               = 0x 00000010 ffffffff8
__TEST07__
resultado      =              0 (sgn)
               = 0x              0 (hex)
               = 0x 00000010 ffffffff
__TEST08__
resultado      =      2147483648 (sgn)
               = 0x      80000000 (hex)
               = 0x 00000010 ffffffff
__TEST09__
resultado      =      2147483632 (sgn)
               = 0x      7fffffff (hex)
               = 0x 00000010 ffffffff
__TEST10__
resultado      =      1600000000 (sgn)
               = 0x      5f5e1000 (hex)
               = 0x 00000010 00000000
__TEST11__
resultado      =      3200000000 (sgn)
               = 0x      bebc2000 (hex)
               = 0x 00000010 00000000
```

```
pablo@laptop: ~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b
TEST12_-----
resultado      =          505032704 (sgn)
               = 0x          1e1a3000 (hex)
               = 0x 00000010 00000001
TEST13_-----
resultado      =          1935228928 (sgn)
               = 0x          73594000 (hex)
               = 0x 00000010 00000007
TEST14_-----
resultado      =          755359744 (sgn)
               = 0x          2d05e000 (hex)
               = 0x 00000010 ffffffff
TEST15_-----
resultado      =          2694967296 (sgn)
               = 0x          a0a1f000 (hex)
               = 0x 00000010 ffffffff
TEST16_-----
resultado      =          1094967296 (sgn)
               = 0x          4143e000 (hex)
               = 0x 00000010 ffffffff
TEST17_-----
resultado      =          694967296 (sgn)
               = 0x          296c5c00 (hex)
               = 0x 00000010 ffffffff
TEST18_-----
resultado      =          2359738368 (sgn)
               = 0x          8ca6c000 (hex)
               = 0x 00000010 ffffffff8
5-3.s: Mensajes del ensamblador:
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
5-3.s:56: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
TEST19_-----
resultado      =          3539607552 (sgn)
               = 0x          d2fa2000 (hex)
               = 0x 00000010 00000004
pablo@laptop:~/Documentos/DGIIM2/EC/Prácticas y Seminarios/Práctica 2/2b$
```

Todo ha salido correctamente y como podemos ver, soluciona el problema que tenía el programa del 5.2. Observar que el TEST14 sale un resultado erróneo por desbordamiento.

Ejercicio 5.4 - Media y resto de N enteros con signo de 32 bits calculada usando registros de 32 bits (N≈16).

Hagamos de nuevo el programa y modifiquemos el script y los tests:

```
# SECCIÓN DE DATOS (.data, variables globales inicializadas)
.section .data
#ifdef TEST
#define TEST 19
#endif

    .macro linea
    #if TEST==1
        .int 1,2,1,2
    #elif TEST==2
        .int -1,-2,-1,-2
    #elif TEST==3
        .int 0x7fffffff,0x7fffffff,0x7fffffff,0x7fffffff
    #elif TEST==4
        .int 0x80000000,0x80000000,0x80000000,0x80000000
    #elif TEST==5
```

```

        .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
#elif TEST==6
        .int 2000000000,2000000000,2000000000,2000000000
#elif TEST==7
        .int 3000000000,3000000000,3000000000,3000000000
#elif TEST==8
        .int -2000000000,-2000000000,-2000000000,-2000000000
#elif TEST==9
        .int -3000000000,-3000000000,-3000000000,-3000000000
#elif TEST>=10 && TEST <=14
        .int 1,1,1,1
#elif TEST >= 15 && TEST<=19
        .int -1,-1,-1,-1
#else
        .error "Definir test"
#endif
    .endm

    .macro linea0
#if TEST>=1 && TEST<=9
    linea
#elif TEST==10
        .int 0,2,1,1
#elif TEST==11
        .int 1,2,1,1
#elif TEST==12
        .int 8,2,1,1
#elif TEST==13
        .int 15,2,1,1
#elif TEST==14
        .int 16,2,1,1
#elif TEST==15
        .int 0,-2,-1,-1
#elif TEST==16
        .int -1,-2,-1,-1
#elif TEST==17
        .int -8,-2,-1,-1
#elif TEST==18
        .int -15,-2,-1,-1
#elif TEST==19
        .int -16,-2,-1,-1#else
        .error "Definir test"
#endif
    .endm

lista:      linea0
            .irpc i,123
            linea
            .endr

longlista:  .int    (.-lista)/4  # . = contador posiciones. Aritm.etiq.
media:      .int 0
resto:      .int 0

formato:
    .ascii "resultado \t = %18ld (sgn)\n"
    .ascii "\t\t = 0x%18lx (hex)\n"
    .asciz "\t\t = 0x %08x %08x\n"

```

```

# SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
.section .text          # PROGRAMA PRINCIPAL
main: .global main      # Programa principal si se usa C runtime

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

trabajar:
    mov     $lista, %rbx  # dirección del array lista
    mov     longlista, %ecx # número de elementos a sumar
    call suma             # == suma(&lista, longlista);
    mov     %eax, media   # salvar media
    mov     %edx, resto   # salvar resto
    ret

# SUBROUTINA: int suma(int* lista, int longlista);
# entrada:  1) %rbx = dirección inicio array
#           2) %ecx = número de elementos a sumar
# salida:    %eax = resultado de la suma
suma:
    mov     $0, %edi      # ponemos a 0 los registros a usar
    mov     $0, %rsi
    mov     $0, %r8d
bucle:
    mov     (%rbx,%rsi,4), %eax # mueve el siguiente elemento de la lista
    cltd                      # extiende el signo de %eax
    add     %eax, %edi         # suma %eax a %edi
    adc     %edx, %r8d         # hace la suma con el acarreo
    inc     %rsi              # incrementa el índice
    cmp     %rsi,%rcx         # compara si son iguales
    jne     bucle             # si no lo, vuelve a hacer el bucle

    mov     %edi, %eax        # mueve la suma a %eax
    mov     %r8d, %edx        # mueve el acarreo a %edx

    idiv    %ecx              # hace la división entera con EDX:EAX

    ret

imprim_C:                  # requiere libc
    mov     $formato, %rdi    # mueven los datos a los registros de salida
    mov     media, %esi
    mov     resto, %ecx
    mov     $0, %eax
    call    printf
    ret

acabar_C:
    mov     media, %edi
    call    exit              # ==exit(resultado)
    ret

```

```
for i in $(seq 1 19); do
rm 5-4;
gcc -x assembler-with-cpp -D TEST=$i 5-4.s -no-pie -o 5-4;
printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./5-4;
done
```

[illegible]

```
pablo@laptop: ~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b
__TEST09__ -----
Media = 1294967296 = 0x0 hex
Resto = 0 = 0x4 hex
__TEST10__ -----
Media = 1 = 0x0 hex
Resto = 0 = 0x0 hex
__TEST11__ -----
Media = 1 = 0x1 hex
Resto = 1 = 0x0 hex
__TEST12__ -----
Media = 1 = 0x8 hex
Resto = 8 = 0x0 hex
__TEST13__ -----
Media = 1 = 0xf hex
Resto = 15 = 0x0 hex
__TEST14__ -----
Media = 2 = 0x0 hex
Resto = 0 = 0x0 hex
__TEST15__ -----
Media = -1 = 0x0 hex
Resto = 0 = 0xffffffff hex
__TEST16__ -----
Media = -1 = 0xffffffff hex
Resto = -1 = 0xffffffff hex
__TEST17__ -----
Media = -1 = 0xffffffff8 hex
Resto = -8 = 0xffffffff hex
__TEST18__ -----
Media = -1 = 0xffffffff1 hex
Resto = -15 = 0xffffffff hex
5-4.s: Mensajes del ensamblador:
5-4.s:65: Error: Definir test
__TEST19__ -----
./run_tests.sh: línea 4: ./5-4: No existe el archivo o el directorio
pablo@laptop:~/Documentos/DGIIM2/EC/Pfácticas y Seminarios/Práctica 2/2b$
```

Todo ha salido como se esperaba. Cabe destacar que el resto puede ser negativo, siendo entonces una división truncada en vez de una de enteros, pues la división de enteros no acepta restos negativos.