Universidad de Granada

Grado en Ingeniería Informática

Estructura de Computadores

Práctica 2:
Suma y media con signo
usando registros de 32 y 64 bits

Memoria realizada por: Ángel Robledillo Perea
Grupo: A

Indice

5.1 - Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits usando uno de ellos como acumulador de acarreos (N≈16)

➢ Código

En esta primera versión se ha hecho uso de dos registros de 32 bits, uno para la suma y otro para los acarreos. De esta forma evitamos un overflow de menos de 64 bits. Se emplearán saltos condicionales ara desarrollar la solución.

```
01: # media5.1.s:
02: #   Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits
03: #   usando uno de ellos como acumulador de acarreos
04: #   comprobar con depurador gdb/ddd y salida printf
05: # SECCIÓN DE DATOS (.data, variables globales inicializadas)
06: .section .data
07: lista:              .int 0xffffffff, 1 # ejs. binario 0b / hex 0x
08: longlista:  .int   (.-lista)/4      # . = contador posiciones. Aritm.etiq.
09: resultado:  .quad   0               #necesitamos un tipo mayor que int para
almacenar un numero superior a 32 bits
10: formato:    .asciz  "suma = %lu = 0x%lx hex\n"      # fmt para printf()
libC
11: # el string “formato” sirve como argumento a la llamada printf opcional
12:
13: # opción: 1) no usar printf, 2)3) usar printf/fmt/exit, 4) usar tb main
14: # 1) as  suma.s -o suma.o
15: #    ld  suma.o -o suma                                    1232 B
16: # 2) as  suma.s -o suma.o                          6520 B
17: #    ld  suma.o -o suma -lc -dynamic-linker /lib64/ld-linux-x86-64.so.2
18: # 3) gcc suma.s -o suma -no-pie –nostartfiles              6544 B
19: # 4) gcc suma.s -o suma     -no-pie                        8664 B
20:
21: # SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
22: .section .text             # PROGRAMA PRINCIPAL
23: #_start: .global _start     # se puede abreviar de esta forma
24: main: .global  main # Programa principal si se usa C runtime
25:
26:     call trabajar          # subrutina de usuario
27:     call imprim_C          # printf()  de libC
28:     call acabar_L          # exit()   del kernel Linux
29: #   call acabar_C          # exit()    de libC
30:     ret
31:
32: trabajar:
33:     mov $lista, %rbx       # dirección del array lista
34:     mov longlista, %ecx    # número de elementos a sumar
35:     call suma                      # == suma(&lista, longlista);
```

```
36:     mov %eax, resultado     # Guardo resultado
37:     mov %edx, resultado+4 # Guardo acarreo en la otra mitad de resultado
38:     ret
39:
40: # SUBRUTINA: suma(int* lista, int longlista);
41: # entrada:  1) %rbx = dirección inicio array
42: #                   2) %rcx = número de elementos a sumar
43: # salida:   %eax = resultado de la suma
44: suma:
45:     mov $0, %eax            # poner a 0 acumulador
46:     mov $0, %edx            # poner a 0 indice de acarreos
47:     mov $0, %esi            # poner a 0 índice de noacarreos
48: bucle:
49:     add (%ebx,%esi,4), %eax # acumular i-ésimo elemento
50:     jnc jump                        #Salto a jum si no hay acarreo
51:     inc %edx
52:
53: jump:
54:     inc %esi                        # Incremento el indice de no acarreo
55:     cmp %esi, %ecx          # Comparación del indice y el numero de
elementos d ela lista
56:     jne bucle               # Si son iguales repito etiqueta bucle
57:
58:     ret
59:
60: imprim_C:                   # requiere libC
61: # si se usa esta subrutina, usar también la línea que define formato
62: # se puede linkar con ld -lc -dyn ó gcc -nostartfiles, o usar main
63:     mov $formato, %rdi      # traduce resultado a decimal/hex
64:     mov resultado, %rsi     # versión libC de syscall __NR_write
65:     mov resultado,%rdx      # ventaja: printf() con fmt "%u" / "%x"
66:     call printf                     # == printf(formato, res, res);
67:     mov $0,%rax                     # varargin sin xmm
68:     ret
69:
70: acabar_L:                   # void _exit(int status);
71:     mov $60, %rax           # exit: servicio 60 kernel Linux
72:     mov resultado, %edi     # status: código a retornar (la suma)
73:     syscall                         # == _exit(resultado)
74:
75:     ret
76:
77: #acabar_C:                  # requiere libC
78:                             # void exit(int status);
79: #   mov  resultado, %edi    # status: código a retornar (la suma)
80: #   call _exit                      # ==  exit(resultado)
81: #   ret
```

Suma5.1.s

➢ Batería de pruebas

Para esta versión solo se realizará una prueba. Sumaremos 0xFFFFFFFF más 1 ya que debería producir un overflow y con este programa buscamos evitar esta situación, ofreciendo asi un resultado correcto.

```
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.1$ gcc -g suma5.1.s -o suma -no-
pie
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.1$ ./suma
suma = 4294967296 = 0x100000000 hex
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.1$
```

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

➢ Código

En esta segunda versión se ha hecho uso de dos registros de 32 bits con los mismos objetivos que en la versión anterior. La diferencia ahora es el uso del comando de suma con acarreo (ADC), suprimiendo los saltos condicionales.

```
001: # media5.2.s:
002: #  Sumar N enteros sin signo de 32 bits sobre dos registros de 32 bits
003: #  mediante extensión con ceros
004: #  comprobar con depurador gdb/ddd y salida printf
005: # SECCIÓN DE DATOS (.data, variables globales inicializadas)
006: .section .data
007: #ifndef TEST
008: #define TEST 1
009: #endif
010: .macro linea       # Resultado - Comentario
011: #if TEST==1                //16 – ejemplo muy sencillo
012:     .int 1,1,1,1
013: #elif TEST==2      //0xffff fff0, casi acarreo
014:     .int 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff
015: #elif TEST==3      // 4294967296 (usg) = 0x100000000 (hex)16x5G= 80G >> 11
280 523 264
016:     .int 0x10000000,0x10000000,0x10000000,0x10000000
017: #elif TEST==4      // 68719476720 (usg) = 0xfffffff0 (hex) 16x5G= 80G >>
11 280 523 264
018:     .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
019: #elif TEST==5      // 68719476720 (usg) = 0xfffffff0 (hex) 16x5G= 80G >>
```

```
11 280 523 264
020:    .int -1, -1, -1, -1
021: #elif TEST==6      // 3200000000 (usg) = 0xbebc2000 (hex) 16x5G= 80G >> 11
280 523 264
022:    .int 200000000, 200000000, 200000000, 200000000
023: #elif TEST==7      // 4800000000 (usg) = 0x11e1a3000 (hex) 16x5G= 80G >>
11 280 523 264
024:    .int 300000000, 300000000, 300000000, 300000000
025: #elif TEST==8      // 11280523264 (usg) = 0x2a05f2000 (hex) Truncado
16x5G= 80G >> 11 280 523 264
026:    .int 5000000000, 5000000000, 5000000000, 5000000000
027: #else
028:    .error "Definir TEST entre 1..8"
029: #endif
030: .endm
031: # formato:          .asciz  "suma = %lu = 0x%lx hex\n"      # fmt para
printf() libC
032: # el string “formato” sirve como argumento a la llamada printf opcional
033: formato:   .ascii "resultado \t = %18lu (usg)\n"
034:                                                                    .ascii
"\t\t = 0x%18lx (hex)\n"
035:                                                                    .asciz
"\t\t = 0x %08x %08x\n"
036: lista: .irpc i, 1234
037:            linea
038:    .endr
039: longlista: .int   (.-lista)/4      # . = contador posiciones. Aritm.etiq.
040: resultado: .quad   0
#necesitamos un tipo mayor que int para
041:
# almacenar un numero superior a 32 bits
042:
043: # opción: 1) no usar printf, 2)3) usar printf/fmt/exit, 4) usar tb main
044: # 1) as  suma.s -o suma.o
045: #    ld  suma.o -o suma                                  1232 B
046: # 2) as  suma.s -o suma.o                          6520 B
047: #    ld  suma.o -o suma -lc -dynamic-linker /lib64/ld-linux-x86-64.so.2
048: # 3) gcc suma.s -o suma -no-pie –nostartfiles             6544 B
049: # 4) gcc suma.s -o suma    -no-pie                        8664 B
050:
051: # SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
052: .section .text             # PROGRAMA PRINCIPAL
053: #_start: .global _start    # se puede abreviar de esta forma
054: main: .global  main        # Programa principal si se usa C runtime
055:
056:    call trabajar                                  # subrutina de usuario
057:    call imprim_C                                  # printf()  de libC
058:    call acabar_L                                  # exit()   del kernel
Linux
059: #  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                     # Guardo resultado
    mov     %edx, resultado+4 # Guardo acarreo en la otra mitad de resultado
    ret

# SUBRUTINA: suma(int* lista, int longlista);
# entrada:         1) %rbx = dirección inicio array
#                  2) %rcx = número de elementos a sumar
# salida:  %eax = resultado de la suma
suma:
    mov $0, %eax                                            # poner a 0 acumulador de bits menos significativos
    mov $0, %edx                                    # poner a 0 acumulador de bits más significativos
    mov $0, %esi                                            # poner a 0 índice
bucle:
    add (%ebx,%esi,4), %eax # acumular i-ésimo elemento
    adc $0, %edx
    inc %esi                                                 # Incremento el indice
    cmp %esi, %ecx          # Comparación del indice y el numero de elementos de la lista
    jne bucle               # Si son iguales repito bucle

  ret

imprim_C:                   # requiere libC
# si se usa esta subrutina, usar también la línea que define formato
# se puede linkar con ld –lc –dyn ó gcc –nostartfiles, o usar main
    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 resultado+4,%ecx
    mov resultado,%r8d
    call printf             # == printf(formato, res, res);
    mov     $0,%rax # varargin sin xmm

    ret

acabar_L:                   # void _exit(int status);
    mov $60, %rax   # exit: servicio 60 kernel Linux
```

```
102:    mov resultado, %edi     # status: código a retornar (la suma)
103:    syscall                 # == _exit(resultado)
104:
105:    ret
106:
107: #acabar_C:                 # requiere libC
108:                            # void exit(int status);
109: #  mov  resultado, %edi    # status: código a retornar (la suma)
110: #  call _exit              # ==  exit(resultado)
111: #  ret
```

Suma5.2.s

➢ Batería de pruebas

```
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.2$ sh test.sh

T#1 resultado    =                 16 (usg)
                 = 0x                10 (hex)
                 = 0x 00000000 00000010
T#2 resultado    =         4294967280 (usg)
                 = 0x          fffffff0 (hex)
                 = 0x 00000000 fffffff0
T#3 resultado    =         4294967296 (usg)
                 = 0x         100000000 (hex)
                 = 0x 00000001 00000000
T#4 resultado    =        68719476720 (usg)
                 = 0x         ffffffff0 (hex)
                 = 0x 0000000f fffffff0
T#5 resultado    =        68719476720 (usg)
                 = 0x         ffffffff0 (hex)
                 = 0x 0000000f fffffff0
T#6 resultado    =         3200000000 (usg)
                 = 0x          bebc2000 (hex)
                 = 0x 00000000 bebc2000
T#7 resultado    =         4800000000 (usg)
                 = 0x         11e1a3000 (hex)
                 = 0x 00000001 1e1a3000
suma5.2.s: Mensajes del ensamblador:
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
suma5.2.s:38: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
T#8 resultado    =        11280523264 (usg)
                 = 0x          2a05f2000 (hex)
                 = 0x 00000002 a05f2000
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.2$
```

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

➢ Código

Con las versiones anteriores se soluciona el overflow para la suma de 32 bits pero se plantea un nuevo requisito, hasta ahora empleabamos numeros enteros positivos pero necesitamos adaptar el código para interpretar los datos como enteros con signo. Esto nos obliga a utilizar comandos de extensión de signo de un registro a dos registros de 32 bits (CDQ). Además, para que en la salida se

interpreten los enteros como enteros con signo, hay que cambiar el tipo de dato en el formato de salida.

```
001: # media5.3.s:
002: #  Sumar N enteros con signo de 32 bits sobre dos registros de 32 bits
003: #  mediante extensión de signo
004: #  comprobar con depurador gdb/ddd y salida printf
005: # SECCIÓN DE DATOS (.data, variables globales inicializadas)
006: .section .data
007: #ifndef TEST
008: #define TEST 1
009: #endif
010: .macro linea        # Resultado - Comentario
011: #if TEST==1         // -16 (sgn) = 0xfffffffffffffff0 (hex)
012:     .int -1, -1, -1, -1
013: #elif TEST==2       // 1073741824 (sgn) = 0x40000000 (hex) positivo pequeño
(suma cabría en sgn32b)
014:     .int 0x04000000, 0x04000000, 0x04000000, 0x04000000
015: #elif TEST==3       // 2147483648 (sgn) = 0x80000000 (hex) positivo
intermedio (sm. cabría en uns32b)
016:     .int 0x08000000,0x08000000,0x08000000,0x08000000
017: #elif TEST==4       // 4294967296 (sgn) = 0x100000000 (hex) positivo
intermedio (sm. no cabría uns32b)
018:     .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
019: #elif TEST==5       // 34359738352 (sgn) = 0x7fffffff0 (hex) positivo
grande (máximo elem. en sgn32b)
020:     .int 0x7FFFFFFF, 0x7FFFFFFF, 0x7FFFFFFF, 0x7FFFFFFF
021: #elif TEST==6       // -34359738368 (sgn) = 0xfffffff800000000 (hex)
negativo grande (mínimo elem. en sgn32b)
022:     .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
023: #elif TEST==7       // -4294967296 (sgn) = 0xffffffff00000000 (hex)
negativo intermedio (no cabría en sgn32b)
024:     .int 0xF0000000, 0xF0000000, 0xF0000000, 0xF0000000
025: #elif TEST==8       // -2147483648 (sgn) = 0xffffffff80000000 (hex)
negativo pequeño (suma cabría en sgn32b)
026:     .int 0xF8000000, 0xF8000000, 0xF8000000, 0xF8000000
027: #elif TEST==9       // -2147483664 (sgn) = 0xffffffff7ffffff0 (hex)
anterior-1 es interm. (no cabría en sgn32b)
028:     .int 0xF7FFFFFF, 0xF7FFFFFF, 0xF7FFFFFF, 0xF7FFFFFF
029: #elif TEST==10      // 1600000000 (sgn) = 0x5f5e1000 (hex) fácil calcular
q. suma cabe sgn32b (<=2Gi-1)
030:     .int 100000000, 100000000, 100000000, 100000000
031: #elif TEST==11      // 3200000000 (sgn) = 0xbebc2000 (hex) pos+gran A·10^b
suma cabe uns32b (<=4Gi-1)
032:     .int 200000000, 200000000, 200000000, 200000000
033: #elif TEST==12      // 4800000000 (sgn) = 0x11e1a3000 (hex) pos+peq A·10^b
suma no cabe uns32b(>=4Gi)
034:     .int 300000000, 300000000, 300000000, 300000000
035: #elif TEST==13      // 32000000000 (sgn) = 0x773594000 (hex) pos+gran
A·10^b reprsntble sgn32b (<=2Gi-1)
```

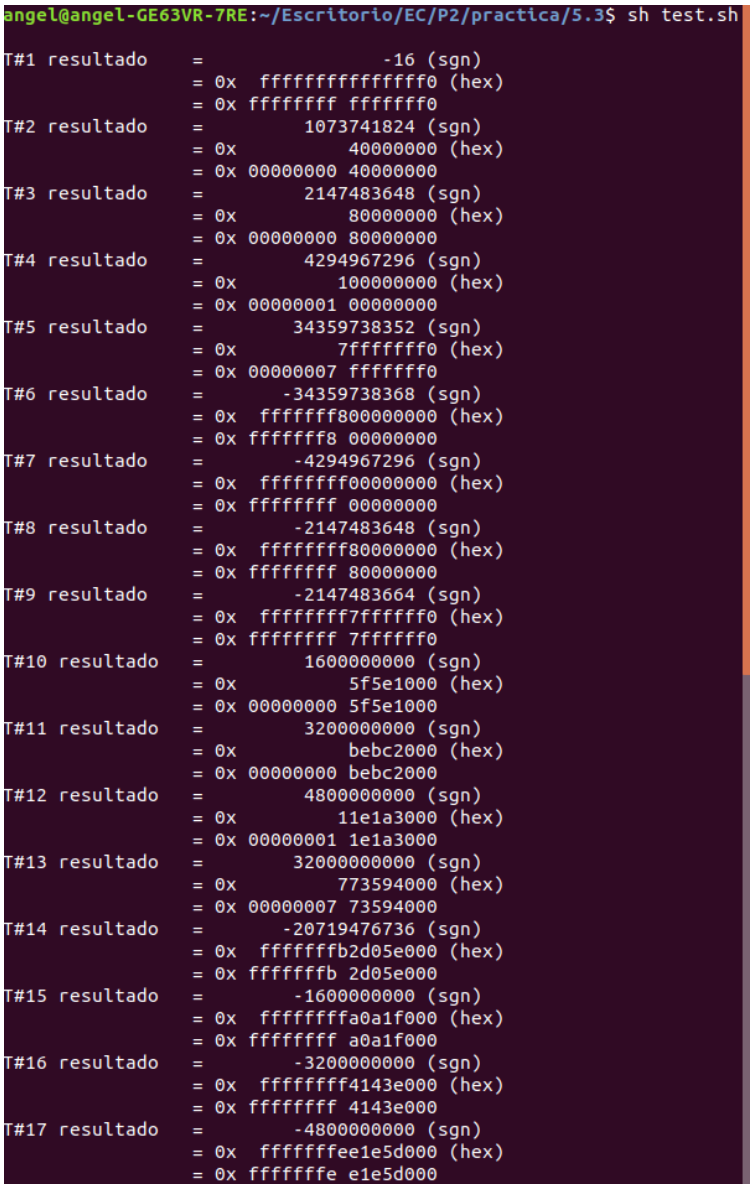
```
036:     .int 2000000000, 2000000000, 2000000000, 2000000000
037: #elif TEST==14     // -20719476736 (sgn) = 0xfffffffb2d05e000 (hex)
pos+peq A·10^b no reprsntble sgn32b(>=2Gi)
038:     .int 3000000000, 3000000000, 3000000000, 3000000000
039: #elif TEST==15     // -1600000000 (sgn) = 0xffffffffa0a1f000 (hex) fácil
calcular q. suma cabe sgn32b (>=-2Gi)
040:     .int -100000000, -100000000, -100000000, -100000000
041: #elif TEST==16     // -3200000000 (sgn) = 0xffffffff4143e000 (hex) neg+peq
-A·10^b suma no cabe sgn32b(<-2Gi)
042:     .int -200000000, -200000000, -200000000, -200000000
043: #elif TEST==17     // -4800000000 (sgn) = 0xfffffffee1e5d000 (hex) aún
menos hubiera cabido
044:     .int -300000000, -300000000, -300000000, -300000000
045: #elif TEST==18     // -32000000000 (sgn) = 0xfffffff88ca6c000 (hex)
neg+gran A·10^b representable sgn32b (>=-2Gi)
046:     .int -2000000000, -2000000000, -2000000000, -2000000000
047: #elif TEST==19     // 20719476736 (sgn) = 0x4d2fa2000 (hex) neg+peq A·10^b
no representable sgn32b(<-2Gi)
048:     .int -3000000000, -3000000000, -3000000000, -3000000000
049: #else
050:     .error "Definir TEST entre 1..19"
051: #endif
052: .endm
053: # formato:          .asciz  "suma = %lu = 0x%lx hex\n"      # fmt para
printf() libC
054: # el string “formato” sirve como argumento a la llamada printf opcional
055: formato:    .ascii "resultado \t = %18ld (sgn)\n"
056:                     .ascii "\t\t = 0x%18lx (hex)\n"
057:                     .asciz "\t\t = 0x %08x %08x\n"
058: lista: .irpc i, 1234
059:             linea
060:     .endr
061: longlista: .int   (.-lista)/4      # . = contador posiciones. Aritm.etiq.
062: resultado: .quad   0                       #necesitamos un tipo mayor que
int para
063:                                                    #almacenar un
numero superior a 32 bits
064:
065: # opción: 1) no usar printf, 2)3) usar printf/fmt/exit, 4) usar tb main
066: # 1) as  suma.s -o suma.o
067: #    ld  suma.o -o suma                                  1232 B
068: # 2) as  suma.s -o suma.o                          6520 B
069: #    ld  suma.o -o suma -lc -dynamic-linker /lib64/ld-linux-x86-64.so.2
070: # 3) gcc suma.s -o suma -no-pie –nostartfiles             6544 B
071: # 4) gcc suma.s -o suma    -no-pie                        8664 B
072:
073: # SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
074: .section .text             # PROGRAMA PRINCIPAL
075: #_start: .global _start    # se puede abreviar de esta forma
076: main: .global  main        # Programa principal si se usa C runtime
```

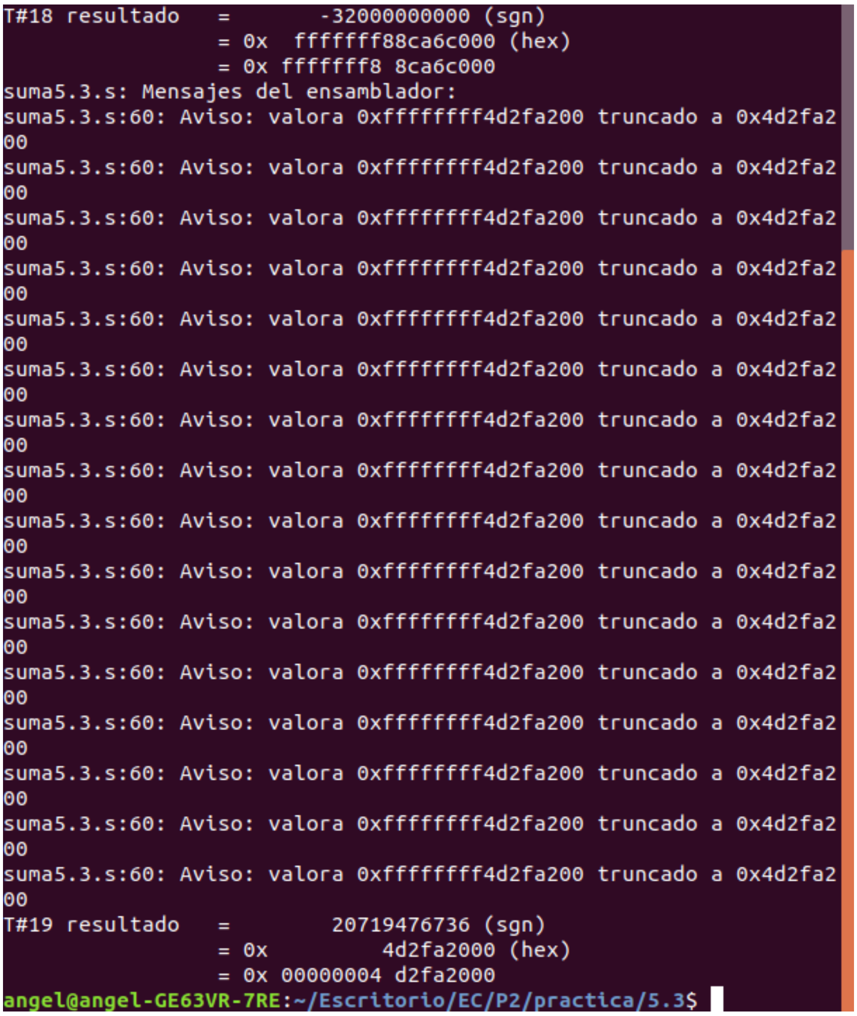
```
077:
078:    call trabajar   # subrutina de usuario
079:    call imprim_C   # printf()  de libC
080:    call acabar_L   # exit()   del kernel Linux
081: #  call acabar_C   # exit()    de libC
082:    ret
083:
084: trabajar:
085:    mov $lista, %rbx                # dirección del array lista
086:    mov longlista, %ecx             # número de elementos a sumar
087:    call suma                                  # == suma(&lista, longlista);
088:    mov %eax, resultado             # Guardo resultado
089:    mov %edx, resultado+4   # Guardo acarreo en la otra mitad de resultado
090:
091:    ret
092:
093: # SUBRUTINA: suma(int* lista, int longlista);
094: # entrada:          1) %rbx = dirección inicio array
095: #                   2) %rcx = número de elementos a sumar
096: # salida:  %eax = resultado de la suma
097: suma:
098:    mov $0, %eax    # poner a 0 registro de bits menos significativos
099:    mov $0, %edx    # poner a 0 registro de bits más significativos
100:    mov $0, %esi    # poner a 0 acumulador de bits menos significativos
101:    mov $0, %edi    # poner a 0 acumulador de bits más significativos
102:    mov $0, %ebp    # poner a 0 indice de bucle
103: bucle:
104:    mov (%ebx,%ebp,4), %eax   # Copio el valor i-ésimo al registro
105:    cdq                  # Transforma el registro %EAX
106:                          # en EDX:EAX con extensión de signo
107:    add %eax, %esi      # Acumulo parte menos significativa con add
108:    adc %edx, %edi      # Acumulo la parte más significativa con adc
109:
110:    inc %ebp            # Incremento cnt
111:    cmp %ebp, %ecx      # Comparación del indice y el numero de elementos
de la lista
112:    jne bucle           # Si son iguales repito bucle
113:
114:    mov %edi, %edx      # Cuando acaba copia los acumuladores
115:    mov %esi, %eax          # en los registros que necesitamos --> EDX:EAX
116:
117:    ret
118:
119: imprim_C:                  # requiere libC
120: # si se usa esta subrutina, usar también la línea que define formato
121: # se puede linkar con ld -lc -dyn ó gcc -nostartfiles, o usar main
122:    mov     $formato, %rdi  # traduce resultado a decimal/hex
123:    mov     resultado, %rsi # versión libC de syscall __NR_write
124:    mov resultado,%rdx      # ventaja: printf() con fmt "%u" / "%x"
125:    mov resultado+4,%ecx
```

```
126:     mov resultado,%r8d
127:     call printf                        # == printf(formato, res, res);
128:     mov $0,%rax                        # varargin sin xmm
129:
130:     ret
131:
132: acabar_L:                              # void _exit(int status);
133:     mov $60, %rax              # exit: servicio 60 kernel Linux
134:     mov resultado, %edi        # status: código a retornar (la suma)
135:     syscall                            # == _exit(resultado)
136:
137:     ret
138:
139: #acabar_C:                     # requiere libC
140:                                # void exit(int status);
141: #  mov  resultado, %edi        # status: código a retornar (la suma)
142: #  call _exit                  # ==  exit(resultado)
143: #  ret
```

Suma5.3.s

➢ Batería de pruebas


angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.3$ sh test.sh
T#1 resultado = -16 (sgn)
= 0x fffffffffffffff0 (hex)
= 0x ffffffff fffffff0
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 fffffff0
T#6 resultado = -34359738368 (sgn)
= 0x fffffff800000000 (hex)
= 0x fffffff8 00000000
T#7 resultado = -4294967296 (sgn)
= 0x ffffffff00000000 (hex)
= 0x ffffffff 00000000
T#8 resultado = -2147483648 (sgn)
= 0x ffffffff80000000 (hex)
= 0x ffffffff 80000000
T#9 resultado = -2147483664 (sgn)
= 0x ffffffff7ffffff0 (hex)
= 0x ffffffff 7ffffff0
T#10 resultado = 1600000000 (sgn)
= 0x 5f5e1000 (hex)
= 0x 00000000 5f5e1000
T#11 resultado = 3200000000 (sgn)
= 0x bebc2000 (hex)
= 0x 00000000 bebc2000
T#12 resultado = 4800000000 (sgn)
= 0x 11e1a3000 (hex)
= 0x 00000001 1e1a3000
T#13 resultado = 32000000000 (sgn)
= 0x 773594000 (hex)
= 0x 00000007 73594000
T#14 resultado = -20719476736 (sgn)
= 0x fffffffb2d05e000 (hex)
= 0x fffffffb 2d05e000
T#15 resultado = -1600000000 (sgn)
= 0x fffffffa0a1f000 (hex)
= 0x ffffffff a0a1f000
T#16 resultado = -3200000000 (sgn)
= 0x fffffff4143e000 (hex)
= 0x ffffffff 4143e000
T#17 resultado = -4800000000 (sgn)
= 0x fffffffee1e5d000 (hex)
= 0x fffffffe e1e5d000


```
T#18 resultado   =       -32000000000 (sgn)
                 = 0x  ffffff88ca6c000 (hex)
                 = 0x fffffff8 8ca6c000
suma5.3.s: Mensajes del ensamblador:
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
suma5.3.s:60: Aviso: valora 0xfffffff4d2fa200 truncado a 0x4d2fa2
00
T#19 resultado   =        20719476736 (sgn)
                 = 0x         4d2fa2000 (hex)
                 = 0x 00000004 d2fa2000
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.3$
```

Versión Final (32 Bits) - Media y resto de N enteros con signo de 32 bits calculada usando registros de 32 bits (N≈16)

➢ Código

Partiendo de la última versión ya podemos realizar suma de numeros enteros de 32 bits con signo. Ahora solo tenemos que realizar la media del conjunto de datos. Para esto usaremos el comando de división con signo (IDIV), el cual utiliza los registros EDX:EAX como dividendo y el registro que especifiquemos como divisor. El resultado se almacena en EAX y el resto de la división en EDX. Mientras el resultado y el resto sean enteros con signo de 32 bits debe de funcionar el programa correctamente.

```
# media5.4.s:
#  Media y resto de N enteros con signo de 32 bits calculada usando
#  registros de 32 bits
#  comprobar con depurador gdb/ddd y salida printf
```

```
005: # SECCIÓN DE DATOS (.data, variables globales inicializadas)
006: .section .data
007: #ifndef TEST
008: #define TEST 1
009: #endif
010: .macro linea       # Media/Resto - Comentario
011: #if TEST==1                // 1/6
012:     .int 1, 2, 1, 2
013: #elif TEST==2      // -1/-6
014:     .int -1,-2,-1,-2
015: #elif TEST==3      // 2147483647/0
016:     .int 0x7FFFFFFF
017: #elif TEST==4      // -2147483648/0
018:     .int 0x80000000
019: #elif TEST==5      // -1/0
020:     .int 0xFFFFFFFF
021: #elif TEST==6      // 2000000000/0
022:     .int 2000000000
023: #elif TEST==7      // -1294967296/0
024:     .int 3000000000
025: #elif TEST==8      // -2000000000/0
026:     .int -2000000000
027: #elif TEST==9      // 1294967296/0 Truncado
028:     .int -3000000000
029: #elif TEST==10     // 1/0
030:     .int 0, 2, 1, 1, 1, 1
031: #elif TEST==11     // 1/3
032:     .int 1, 2, 1, 1, 1, 1
033: #elif TEST==12     // 2/6
034:     .int 8, 2, 1, 1, 1, 1
035: #elif TEST==13     // 3/9
036:     .int 15, 2, 1, 1, 1, 1
037: #elif TEST==14     // 3/12
038:     .int 16, 2, 1, 1, 1, 1
039: #elif TEST==15     // -1/0
040:     .int 0, -2, -1, -1, -1, -1
041: #elif TEST==16     // -1/-3
042:     .int -1, -2, -1, -1, -1, -1
043: #elif TEST==17     // -2/-6
044:     .int -8, -2, -1, -1, -1, -1
045: #elif TEST==18     // -3/-9
046:     .int -15, -2, -1, -1, -1, -1
047: #elif TEST==19     // -3/-12
048:     .int -16, -2, -1, -1, -1, -1
049: #else
050:     .error "Definir TEST entre 1..19"
051: #endif
052: .endm
053:
054: # formato:          .asciz  "suma = %lu = 0x%lx hex\n"      # fmt para
```

```
printf() libC
055: # el string “formato” sirve como argumento a la llamada printf opcional
056: formato:   .ascii "\nRegistros 32 bits: \n"
057:                 .ascii "\n\tMedia: \n"
058:                 .ascii "\t\t = %d (sgn) = 0x%x  hex\n"
059:                 .ascii "\tResto: \n"
060:                 .asciz "\t\t= %d (sgn) = 0x%x hex\n"
061: lista: .irpc i, 123
062:           linea
063:    .endr
064: #lista:           .int 0xffffffff, 1      # ejs. binario 0b / hex 0x
065: longlista: .int   (.-lista)/4      # . = contador posiciones. Aritm.etiq.
066: media:     .int   0
067: resto:     .int   0
068: # formato:         .asciz  "suma = %lu = 0x%lx hex\n"      # fmt para
printf() libC
069: # el string “formato” sirve como argumento a la llamada printf opcional
070:
071: # opción: 1) no usar printf, 2)3) usar printf/fmt/exit, 4) usar tb main
072: # 1) as  suma.s -o suma.o
073: #    ld  suma.o -o suma                                    1232 B
074: # 2) as  suma.s -o suma.o                          6520 B
075: #    ld  suma.o -o suma -lc -dynamic-linker /lib64/ld-linux-x86-64.so.2
076: # 3) gcc suma.s -o suma -no-pie –nostartfiles             6544 B
077: # 4) gcc suma.s -o suma    -no-pie                        8664 B
078:
079: # SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
080: .section .text             # PROGRAMA PRINCIPAL
081: #_start: .global _start    # se puede abreviar de esta forma
082: main: .global  main        # Programa principal si se usa C runtime
083:
084:    call trabajar   # subrutina de usuario
085:    call imprim_C   # printf()  de libC
086:    call acabar_L   # exit()   del kernel Linux
087: #  call acabar_C   # exit()    de libC
088:    ret
089:
090: trabajar:
091:    mov $lista, %rbx        # dirección del array lista
092:    mov longlista, %ecx     # número de elementos a sumar
093:    call suma               # == suma(&lista, longlista);
094:    mov %eax, media # Guardo cociente de la división
095:    mov %edx, resto # Guardo resto de la división
096:
097:    ret
098:
099: # SUBRUTINA: suma(int* lista, int longlista);
100: # entrada:         1) %rbx = dirección inicio array
101: #                  2) %rcx = número de elementos a sumar
102: # salida:  %eax = resultado de la suma
```

```
103: suma:
104:     mov $0, %eax            # poner a 0 registro de bits menos
significativos
105:     mov $0, %edx            # poner a 0 registro de bits más significativos
106:     mov $0, %esi            # poner a 0 acumulador de bits menos
significativos
107:     mov $0, %edi            # poner a 0 acumulador de bits más
significativos
108:     mov $0, %ebp            # poner a 0 indice de bucle
109: bucle:
110:     mov (%ebx,%ebp,4), %eax   # Copio el valor i-ésimo al registro
111:     cdq                 # Transforma el registro %EAX
112:                          # en EDX:EAX con extensión de signo
113:     add %eax, %esi          # Acumulo parte menos significativa con add
114:     adc %edx, %edi          # Acumulo la parte más significativa con adc
115:
116:     inc %ebp                # Incremento cnt
117:     cmp %ebp, %ecx          # Comparación del indice y el numero de
elementos de la lista
118:     jne bucle               # Si son iguales repito bucle
119:
120:     mov %edi, %edx      # Cuando acaba copia los acumuladores
121:     mov %esi, %eax          # en los registros que necesitamos --> EDX:EAX
122:
123:     idiv %ecx                       #División con signo de EDX:EAX entre el
numero de elementos en la lista
124:                                         #El cociente se almacena en
%EAX
125:                                         #El resto se almacena en %EDX
126:     ret
127:
128: imprim_C:                  # requiere libC
129: # si se usa esta subrutina, usar también la línea que define formato
130: # se puede linkar con ld -lc -dyn ó gcc -nostartfiles, o usar main
131:     mov     $formato, %rdi  # traduce resultado a decimal/hex
132:     mov     media, %esi             # versión libC de syscall __NR_write
133:     mov media,%edx          # ventaja: printf() con fmt "%u" / "%x"
134:     mov resto,%ecx
135:     mov resto,%r8d
136:     call printf                     # == printf(formato, res, res);
137:     mov $0,%rax                     # varargin sin xmm
138:
139:     ret
140:
141: acabar_L:                          # void _exit(int status);
142:     mov $60, %rax           # exit: servicio 60 kernel Linux
143:     mov $0, %edi            # status: código a retornar (la suma)
144:     syscall                         # == _exit(resultado)
145:
146:     ret
```

```
147:
148: #acabar_C:                  # requiere libC
149:                            # void exit(int status);
150: #  mov  resultado, %edi     # status: código a retornar (la suma)
151: #  call _exit                      # ==  exit(resultado)
152: #  ret
```

Suma5.4.s

➢ Batería de pruebas

```
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.4$ sh test.sh
T#1
Registros 32 bits:

        Media:
                 = 1 (sgn) = 0x1  hex
        Resto:
                = 6 (sgn) = 0x6 hex
T#2
Registros 32 bits:

        Media:
                 = -1 (sgn) = 0xffffffff  hex
        Resto:
                = -6 (sgn) = 0xfffffffa hex
T#3
Registros 32 bits:

        Media:
                 = 2147483647 (sgn) = 0x7fffffff  hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#4
Registros 32 bits:

        Media:
                 = -2147483648 (sgn) = 0x80000000  hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#5
Registros 32 bits:

        Media:
                 = -1 (sgn) = 0xffffffff  hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#6
Registros 32 bits:

        Media:
                 = 2000000000 (sgn) = 0x77359400  hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#7
Registros 32 bits:

        Media:
                 = -1294967296 (sgn) = 0xb2d05e00  hex
        Resto:
                = 0 (sgn) = 0x0 hex
```

```
T#8
Registros 32 bits:

        Media:
                 = -2000000000 (sgn) = 0x88ca6c00  hex
        Resto:
                = 0 (sgn) = 0x0 hex
suma5.4.s: Mensajes del ensamblador:
suma5.4.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa20
0
suma5.4.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa20
0
suma5.4.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa20
0
T#9
Registros 32 bits:

        Media:
                 = 1294967296 (sgn) = 0x4d2fa200  hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#10
Registros 32 bits:

        Media:
                 = 1 (sgn) = 0x1  hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#11
Registros 32 bits:

        Media:
                 = 1 (sgn) = 0x1  hex
        Resto:
                = 3 (sgn) = 0x3 hex
T#12
Registros 32 bits:

        Media:
                 = 2 (sgn) = 0x2  hex
        Resto:
                = 6 (sgn) = 0x6 hex
T#13
Registros 32 bits:

        Media:
                 = 3 (sgn) = 0x3  hex
        Resto:
                = 9 (sgn) = 0x9 hex
```

```
T#14
Registros 32 bits:

        Media:
                 = 3 (sgn) = 0x3  hex
        Resto:
                = 12 (sgn) = 0xc hex
T#15
Registros 32 bits:

        Media:
                 = -1 (sgn) = 0xffffffff  hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#16
Registros 32 bits:

        Media:
                 = -1 (sgn) = 0xffffffff  hex
        Resto:
                = -3 (sgn) = 0xfffffffd hex
T#17
Registros 32 bits:

        Media:
                 = -2 (sgn) = 0xfffffffe  hex
        Resto:
                = -6 (sgn) = 0xfffffffa hex
T#18
Registros 32 bits:

        Media:
                 = -3 (sgn) = 0xfffffffd  hex
        Resto:
                = -9 (sgn) = 0xfffffff7 hex
T#19
Registros 32 bits:

        Media:
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.4$
```

Versión Final (64 Bits) - Media y resto de N enteros calculada en 32 y en 64 bits (N≈16)

➢ Código

Esta versión final unicamente se diferencia de la anterior en que se emplean directamente registros de 64 bits para realizar los calculos de suma y división. La mayor diferencia es el uso del comando de copia con extensión de signo aumentando el tamaño del registro incluido (MOVSLQ) y del comando para extender signo de un registro de 64 bits a dos registros del mismo tamaño (CQO), para transformar RAX en RDX:RAX y poder realizar la división de la misma forma que se hizo en la version anterior.

```
001: # media.s:
002: #  Media y resto de N enteros calculada en 32 y en 64 bits
003: #  comprobar con depurador gdb/ddd y salida printf
004: # SECCIÓN DE DATOS (.data, variables globales inicializadas)
005: .section .data
006: #ifndef TEST
007: #define TEST 1
008: #endif
009: .macro linea       # Media/Resto - Comentario
010: #if TEST==1                // 1/6
011:    .int 1, 2, 1, 2
012: #elif TEST==2      // -1/-6
013:    .int -1,-2,-1,-2
014: #elif TEST==3      // 2147483647/0
015:    .int 0x7FFFFFFF
016: #elif TEST==4      // -2147483648/0
017:    .int 0x80000000
018: #elif TEST==5      // -1/0
019:    .int 0xFFFFFFFF
020: #elif TEST==6      // 2000000000/0
021:    .int 2000000000
022: #elif TEST==7      // -1294967296/0
023:    .int 3000000000
024: #elif TEST==8      // -2000000000/0
025:    .int -2000000000
026: #elif TEST==9      // 1294967296/0 Truncado
027:    .int -3000000000
028: #elif TEST==10     // 1/0
029:    .int 0, 2, 1, 1, 1, 1
030: #elif TEST==11     // 1/3
031:    .int 1, 2, 1, 1, 1, 1
032: #elif TEST==12     // 2/6
033:    .int 8, 2, 1, 1, 1, 1
034: #elif TEST==13     // 3/9
```

```
035:    .int 15, 2, 1, 1, 1, 1
036: #elif TEST==14     // 3/12
037:    .int 16, 2, 1, 1, 1, 1
038: #elif TEST==15     // -1/0
039:    .int 0, -2, -1, -1, -1, -1
040: #elif TEST==16     // -1/-3
041:    .int -1, -2, -1, -1, -1, -1
042: #elif TEST==17     // -2/-6
043:    .int -8, -2, -1, -1, -1, -1
044: #elif TEST==18     // -3/-9
045:    .int -15, -2, -1, -1, -1, -1
046: #elif TEST==19     // -3/-12
047:    .int -16, -2, -1, -1, -1, -1
048: #else
049:    .error "Definir TEST entre 1..19"
050: #endif
051: .endm
052:
053: # formato:          .asciz  "suma = %lu = 0x%lx hex\n"      # fmt para
printf() libC
054: # el string “formato” sirve como argumento a la llamada printf opcional
055: formato:   .ascii "\nRegistros 32 bits: \n"
056:                   .ascii "\n\tMedia: \n"
057:                   .ascii "\t\t = %d (sgn) = 0x%x  hex\n"
058:                   .ascii "\tResto: \n"
059:                   .asciz "\t\t= %d (sgn) = 0x%x hex\n"
060: formatoq:  .ascii "\nRegistros 64 bits: \n"
061:                   .ascii "\n\tMedia: \n"
062:                   .ascii "\t\t = %d (sgn) = 0x%x hex\n"
063:                   .ascii "\tResto: \n"
064:                   .asciz "\t\t= %d (sgn) = 0x%x hex\n"
065: lista: .irpc i, 123
066:           linea
067:    .endr
068: longlista: .int   (.-lista)/4      # . = contador posiciones. Aritm.etiq.
069: media:     .int   0
070: resto:     .int   0
071: mediaq:    .quad   0
072: restoq:    .quad   0
073:
074:
075: # opción: 1) no usar printf, 2)3) usar printf/fmt/exit, 4) usar tb main
076: # 1) as  suma.s -o suma.o
077: #    ld  suma.o -o suma                                    1232 B
078: # 2) as  suma.s -o suma.o                          6520 B
079: #    ld  suma.o -o suma -lc -dynamic-linker /lib64/ld-linux-x86-64.so.2
080: # 3) gcc suma.s -o suma -no-pie –nostartfiles              6544 B
081: # 4) gcc suma.s -o suma    -no-pie                         8664 B
082:
083: # SECCIÓN DE CÓDIGO (.text, instrucciones máquina)
```

```
084: .section .text             # PROGRAMA PRINCIPAL
085: #_start: .global _start    # se puede abreviar de esta forma
086: main: .global  main        # Programa principal si se usa C runtime
087:
088:    mov $lista, %rbx        # dirección del array lista
089:    mov longlista, %ecx     # número de elementos a sumar
090:    call suma                       # == suma(&lista, longlista);
091:    mov %eax, media         # Guardo cociente de la división
092:    mov %edx, resto         # Guardo resto de la división
093:    call imprim_C           # printf()  de libC
094:    mov $lista, %rbx        # dirección del array lista
095:    mov longlista, %ecx     # número de elementos a sumar
096:    call sumaq                      # == suma(&lista, longlista);
097:    mov %rax, mediaq        # Guardo cociente de la división
098:    mov %rdx, restoq        # Guardo resto de la división
099:    call imprim_Q
100:    call acabar_L           # exit()   del kernel Linux
101:    #call acabar_C          # exit()    de libC
102:    ret
103:
104: # SUBRUTINA: suma(int* lista, int longlista);
105: # entrada:         1) %rbx = dirección inicio array
106: #                  2) %rcx = número de elementos a sumar
107: # salida:  %eax = resultado de la suma
108: suma:      #Suma con registros de 32 bits
109:    mov  $0, %eax           # poner a 0 registro de bits menos
significativos
110:    mov  $0, %edx           # poner a 0 registro de bits más significativos
111:    mov      $0, %esi               # poner a 0 acumulador de bits menos
significativos
112:    mov      $0, %edi               # poner a 0 acumulador de bits más
significativos
113:    mov      $0, %ebp               # poner a 0 indice de bucle
114: bucle:
115:    mov (%ebx,%ebp,4), %eax   # Copio el valor i-ésimo al registro
116:    cdq                 # Transforma el registro %EAX
117:                         # en EDX:EAX con extensión de signo
118:    add %eax, %esi          # Acumulo parte menos significativa con add
119:    adc %edx, %edi          # Acumulo la parte más significativa con adc
120:
121:    inc %ebp                # Incremento cnt
122:    cmp %ebp, %ecx          # Comparación del indice y el numero de
elementos de la lista
123:    jne bucle               # Si son iguales repito bucle
124:
125:    mov %edi, %edx      # Cuando acaba copia los acumuladores
126:    mov %esi, %eax          # en los registros que necesitamos --> EDX:EAX
127:
128:    idiv %ecx                       #División con signo de EDX:EAX entre el
numero de elementos en la lista
```

```
129:                                            #El cociente se almacena en
%EAX
130:                                            #El resto se almacena en %EDX
131:    ret
132:
133: sumaq: #Suma con registros de 64 bits
134:    mov  $0, %rax           # poner a 0 registro de bits menos
significativos
135:    mov  $0, %rdx           # poner a 0 registro de bits más significativos
136:    mov      $0, %rsi               # poner a 0 indice del bucle
137:    mov      $0, %rdi               # poner a 0 acumulador
138: bucleq:
139:    movslq (%ebx,%esi,4), %rax  # Copio el valor i-ésimo al registro
140:                                                    #y extiendo el
signo con MOVSLQ
141:
142:    add %rax, %rdi          # Acumulo suma
143:
144:    inc %esi                # Incremento cnt
145:    cmp %esi, %ecx          # Comparación del indice y el numero de
elementos d ela lista
146:    jne bucleq              # Si son iguales repito bucle
147:
148:    mov %rdi, %rax      # Guardo la suma total
149:
150:    cqo                     # Transforma el registro %RAX
151:                        # en RDX:RAX con extensión de signo
152:    idiv %rcx                       #División con signo de RDX:RAX entre el
numero de elementos en la lista
153:                                            #El cociente se almacena en
%RAX
154:                                            #El resto se almacena en %RDX
155:    ret
156:
157: imprim_C:                  # requiere libC
158: # si se usa esta subrutina, usar también la línea que define formato
159: # se puede linkar con ld -lc -dyn ó gcc -nostartfiles, o usar main
160:    mov     $formato, %rdi  # traduce resultado a decimal/hex
161:    mov     media, %esi             # versión libC de syscall __NR_write
162:    mov media,%edx          # ventaja: printf() con fmt "%u" / "%x"
163:    mov resto,%ecx
164:    mov resto,%r8d
165:    call printf                     # == printf(formato, res, res);
166:    mov $0,%rax                     # varargin sin xmm
167:
168:    ret
169:
170: imprim_Q:                  # requiere libC
171: # si se usa esta subrutina, usar también la línea que define formato
172: # se puede linkar con ld -lc -dyn ó gcc -nostartfiles, o usar main
```

```
173:    mov $formatoq, %rdi # traduce resultado a decimal/hex
174:    mov media, %rsi             # versión libC de syscall __NR_write
175:    mov media,%rdx          # ventaja: printf() con fmt "%u" / "%x"
176:    mov resto,%rcx
177:    mov resto,%r8
178:    call printf                     # == printf(formato, res, res);
179:    mov $0,%rax                     # varargin sin xmm
180:
181:    ret
182:
183: acabar_L:                          # void _exit(int status);
184:    mov $60, %rax           # exit: servicio 60 kernel Linux
185:    mov $0, %edi            # status: código a retornar (la suma)
186:    syscall                         # == _exit(resultado)
187:
188:    ret
189:
190: #acabar_C:                 # requiere libC
191:                            # void exit(int status);
192: #  mov  resultado, %edi    # status: código a retornar (la suma)
193: #  call _exit              # ==  exit(resultado)
194: #  ret
```

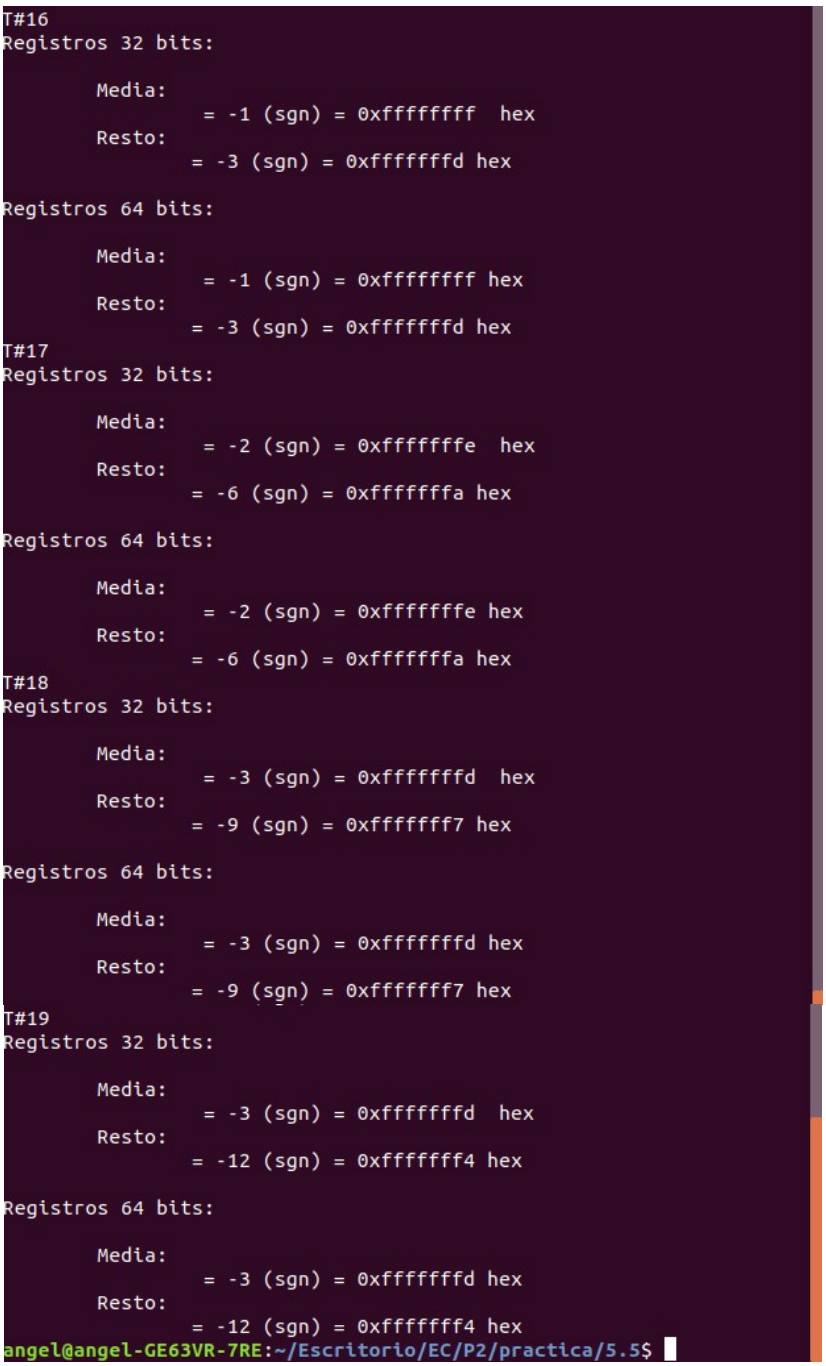
Media.s

➢ Batería de pruebas

```
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.5$ sh test.sh
T#1
Registros 32 bits:

        Media:
                 = 1 (sgn) = 0x1  hex
        Resto:
                = 6 (sgn) = 0x6 hex

Registros 64 bits:

        Media:
                 = 1 (sgn) = 0x1 hex
        Resto:
                = 6 (sgn) = 0x6 hex
T#2
Registros 32 bits:

        Media:
                 = -1 (sgn) = 0xffffffff  hex
        Resto:
                = -6 (sgn) = 0xfffffffa hex

Registros 64 bits:

        Media:
                 = -1 (sgn) = 0xffffffff hex
        Resto:
                = -6 (sgn) = 0xfffffffa hex
T#3
Registros 32 bits:

        Media:
                 = 2147483647 (sgn) = 0x7fffffff  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = 2147483647 (sgn) = 0x7fffffff hex
        Resto:
                = 0 (sgn) = 0x0 hex
```

```
T#4
Registros 32 bits:

        Media:
                 = -2147483648 (sgn) = 0x80000000  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = -2147483648 (sgn) = 0x80000000 hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#5
Registros 32 bits:

        Media:
                 = -1 (sgn) = 0xffffffff  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = -1 (sgn) = 0xffffffff hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#6
Registros 32 bits:

        Media:
                 = 2000000000 (sgn) = 0x77359400  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = 2000000000 (sgn) = 0x77359400 hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#7
Registros 32 bits:

        Media:
                 = -1294967296 (sgn) = 0xb2d05e00  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = -1294967296 (sgn) = 0xb2d05e00 hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#8
Registros 32 bits:

        Media:
                 = -2000000000 (sgn) = 0x88ca6c00  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = -2000000000 (sgn) = 0x88ca6c00 hex
        Resto:
                = 0 (sgn) = 0x0 hex
suma5.5.s: Mensajes del ensamblador:
suma5.5.s:67: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
suma5.5.s:67: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
suma5.5.s:67: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
T#9
Registros 32 bits:

        Media:
                 = 1294967296 (sgn) = 0x4d2fa200  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = 1294967296 (sgn) = 0x4d2fa200 hex
        Resto:
                = 0 (sgn) = 0x0 hex
```

```
T#10
Registros 32 bits:

        Media:
                 = 1 (sgn) = 0x1  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = 1 (sgn) = 0x1 hex
        Resto:
                = 0 (sgn) = 0x0 hex
T#11
Registros 32 bits:

        Media:
                 = 1 (sgn) = 0x1  hex
        Resto:
                = 3 (sgn) = 0x3 hex

Registros 64 bits:

        Media:
                 = 1 (sgn) = 0x1 hex
        Resto:
                = 3 (sgn) = 0x3 hex
T#12
Registros 32 bits:

        Media:
                 = 2 (sgn) = 0x2  hex
        Resto:
                = 6 (sgn) = 0x6 hex

Registros 64 bits:

        Media:
                 = 2 (sgn) = 0x2 hex
        Resto:
                = 6 (sgn) = 0x6 hex
T#13
Registros 32 bits:

        Media:
                 = 3 (sgn) = 0x3  hex
        Resto:
                = 9 (sgn) = 0x9 hex

Registros 64 bits:

        Media:
                 = 3 (sgn) = 0x3 hex
        Resto:
                = 9 (sgn) = 0x9 hex
T#14
Registros 32 bits:

        Media:
                 = 3 (sgn) = 0x3  hex
        Resto:
                = 12 (sgn) = 0xc hex

Registros 64 bits:

        Media:
                 = 3 (sgn) = 0x3 hex
        Resto:
                = 12 (sgn) = 0xc hex
T#15
Registros 32 bits:

        Media:
                 = -1 (sgn) = 0xffffffff  hex
        Resto:
                = 0 (sgn) = 0x0 hex

Registros 64 bits:

        Media:
                 = -1 (sgn) = 0xffffffff hex
        Resto:
                = 0 (sgn) = 0x0 hex
```


T#16
Registros 32 bits:
Media:
= -1 (sgn) = 0xffffffff hex
Resto:
= -3 (sgn) = 0xfffffffd hex
Registros 64 bits:
Media:
= -1 (sgn) = 0xffffffff hex
Resto:
= -3 (sgn) = 0xfffffffd hex
T#17
Registros 32 bits:
Media:
= -2 (sgn) = 0xfffffffe hex
Resto:
= -6 (sgn) = 0xfffffffa hex
Registros 64 bits:
Media:
= -2 (sgn) = 0xfffffffe hex
Resto:
= -6 (sgn) = 0xfffffffa hex
T#18
Registros 32 bits:
Media:
= -3 (sgn) = 0xfffffffd hex
Resto:
= -9 (sgn) = 0xfffffff7 hex
Registros 64 bits:
Media:
= -3 (sgn) = 0xfffffffd hex
Resto:
= -9 (sgn) = 0xfffffff7 hex
T#19
Registros 32 bits:
Media:
= -3 (sgn) = 0xfffffffd hex
Resto:
= -12 (sgn) = 0xfffffff4 hex
Registros 64 bits:
Media:
= -3 (sgn) = 0xfffffffd hex
Resto:
= -12 (sgn) = 0xfffffff4 hex
angel@angel-GE63VR-7RE:~/Escritorio/EC/P2/practica/5.5$