José Javier Alonso Ramos jotajota98@correo.ugr.es

PRÁCTICA 2

Programación ensamblador x86-64 Linux

Durante todos los ejercicios realizados utilizaremos las siguientes variables:

lista: almacena todos los numeros con los que vamos a trabjar
longlista: almacena el numero de datos guardados en lista
resultado: guarda el calculo realizado con los datos de lista.

Como es una variable que debe ser de 64 bits y vamos a estar trabajando con registros de 32bits almacenaremos los datos de manera que el registro que guarde la parte más significativa lo almacenaremos en resultado+4 y el registro que guarde la parte menos significativa lo almacenaremos en resultado. De esta manera al acceder a resultado obtendremos en orden los 64 bits que forman el número

formato: guarda la manera en la que serán mostrados por pantalla los datos calculados

En los ejercicios dedicados a calcular la media veremos también:

media: almacena la parte entera de la división euclidiana
resto: almacena el resto de la división

Los códigos fuente serán adjuntados en el archivo .zip

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)

La parte a destacar de este ejercicio es la suma y como añadimos el acarreo mediante un salto condicional.

```
50:         movq     $0, %rsi        # iterador de la lista
51:         movl     $0, %eax        # acumulador de la suma. Representa la parte menos s
                                       ignificativa
52:         movl     $0, %edx        # acumulador de la suma. Representa la parte mÃ¡s si
                                       gnificativa
53: bucle:
54:         addl    (%rbx,%rsi,4), %eax
55:         jnc             no_inc          # si no hay acarreo no incrementamos %edx y
                                               seguimos con las iteraciones del bucle
56:         inc     %edx
57: no_inc:
58:         inc     %rsi
```

Utilizamos los registros **%eax** y **%edx** para poder conformar números de 64bits.
Para imprimir:

```
34: #imprim_C
35:         movq   $formato, %rdi
36:         movq   resultado,%rsi
37:         movq   resultado,%rdx
38:         movl          $0,%eax   # varargin sin xmm
39:         call  printf            # == printf(formato, res, res);
40:
41:         # SegÃºn el manual de 'prinf' formato debe ser especificado en %rdi,
42:         # el primer resultado a mostrar (unsigned long) en %rsi y el segundo (hexade
              cimal long) en %rdx
```

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

En este ejercicio en vez de tener una única lista con números definiremos una macro que nos permitirá seleccionar distintas listas para realizar los cálculos y tener mayor diversidad de salidas.

```
15: #define TEST 9
16: #endif
17: .macro linea
18: #if TEST==1                                          // 16 â\200\223 ejemplo muy sencillo
19:         .int 1,1,1,1
20: #elif TEST==2                               // 0x0 ffff fff0, casi acarreo
21:         .int 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff
22: #elif TEST==3                               // 0x10000000, justo 1 acarreo
23:         .int 0x10000000,0x10000000,0x10000000,0x10000000
24: #elif TEST==4
25:         .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
26: #elif TEST==5                               // no trabaja con numeros negativos
27:         .int -1,-1,-1,-1
28: #elif TEST==6
29:         .int 200000000,200000000,200000000,200000000
30: #elif TEST==7
31:         .int 300000000,300000000,300000000,300000000
32: #elif TEST==8                               // 11 280 523 264 << 16x5e9= 80e9
33:         .int 5000000000,5000000000,5000000000,5000000000
34: #else
35:         .error "Definir TEST entre 1..8"
36: #endif
37:         .endm
38:
39: lista: .irpc i,1234
40:                         linea
41:         .endr
```

La parte a destacar vuelve a ser la suma y como añadimos el acarreo mediante una operación específica para ello:

```
80: suma:
81:         movq     $0, %rsi       # iterador de la lista
82:         movl     $0, %eax       # acumulador de la suma. Representa la parte menos s
                                      ignificativa
83:         movl     $0, %edx       # acumulador de la suma. Representa la parte mÃ¡s si
                                      gnificativa
84: bucle:
85:         addl    (%rbx,%rsi,4), %eax
86:         adc     $0, %edx
87:         inc     %rsi
88:         cmpq    %rsi,%rcx
89:         jne     bucle
90:
91:         ret
92:
```

La operación **adc** suma el primer operando al segundo junto con el bit de acarreo.
Para su ejecución utilizaremos:
for i in $(seq 1 9); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie media.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done

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

Para su ejecución utilizaremos:
for i in $(seq 1 20); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie media.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done

Hemos añadido más posibilidades a la macro de 'lista'

El cambio importante sigue estando en el código de 'suma':

```
104: suma:
105:          movq     $0, %r8         # iterador de la lista
106:          movl     $0, %eax        # En un principio se usarÃ¡ para extender el signo a
                                          %edx. Representa la parte menos significativa
107:          movl     $0, %esi        # Acumulador de la suma. Representa la parte menos s
                                         ignificativa
108:          movl     $0, %edi        # Acumulador de la suma. Representa la parte mÃ¡s si
                                         gnificativa
109: bucle:
110:          movl    (%rbx,%r8,4), %eax
111:          cdq
112:          add     %eax, %esi
113:          adc     %edx, %edi
114:          inc     %r8
115:          cmpq    %r8,%rcx
116:          jne     bucle
117:
118:          ret
119:
```

Ya no acumulamos en **%eax** sino que copiamos en él el número de la lista con el que toca trabajar. Con la operación **cdq** extendemos el signo de **%eax a %edx.** Acumulamos la parte menos significativa en **%esi** y la más significativa en **%edi.** La operación **adc** ya no sumará siempre *CF+0+%reg* sino que sumará *CF+%edx+reg* siendo **%edx** 0x0000 0000 ó 0xffff ffff dependiendo si el número a sumar es positivo o negativo respectivamente.

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

Para su ejecución utilizaremos:
for i in $(seq 1 20); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie media.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done

Hemos añadido más posibilidades a la macro de 'lista' y además nos apoyamos en otra macro para su formación:

```
14: #ifndef TEST
15: #define TEST 20
16: #endif
17: .macro linea
18: #if  TEST==1                    //1             8
19:         .int 1,2,1,2
20: #elif TEST==2                   //-1    -8
21:         .int -1,-2,-1,-2
22: #elif TEST==3                   //2147483647    0
23:         .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
24: #elif TEST==4                   //-2147483648   0
25:         .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
26: #elif TEST==5                   //-1    0
27:         .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
28: #elif TEST==6                   //2000000000    0
29:         .int 2000000000, 2000000000, 2000000000, 2000000000
30: #elif TEST==7                   //desbordamiento--> -1294967296 0
31:         .int 3000000000, 3000000000, 3000000000, 3000000000
32: #elif TEST==8                   //-2000000000   0
33:         .int -2000000000, -2000000000, -2000000000, -2000000000
34: #elif TEST==9                   //desbordamiento--> 1294967296  0
35:         .int -3000000000, -3000000000, -3000000000, -3000000000
36: #elif TEST>=10 && TEST<=14
37:         .int 1, 1, 1, 1
38: #elif TEST>=15 && TEST<=19
39:         .int -1, -1, -1, -1
40: #else                   //
41:         .error "Definir TEST ente 1..19"
42: #endif                  //
43:         .endm
44:
45: .macro linea0
46: #if TEST>=1 && TEST<=9
47:         linea
48: #elif TEST==10                  //1     0
49:         .int 0,2,1,1
50: #elif TEST==11                  //1     1
51:         .int 1,2,1,1
52: #elif TEST==12                  //1     8
53:         .int 8,2,1,1
54: #elif TEST==13                  //1     15
55:         .int 15,2,1,1
56: #elif TEST==14                  //2     0
57:         .int 16,2,1,1
58: #elif TEST==15                  //-1    0
59:         .int 0,-2,-1,-1
60: #elif TEST==16                  //-1    -1
61:         .int -1,-2,-1,-1
62: #elif TEST==17                  //-1    -8
63:         .int -8,-2,-1,-1
64: #elif TEST==18                  //-1    -15
```

```
65:         .int -15,-2,-1,-1
66: #elif TEST==19                  //-2    0
67:         .int -16,-2,-1,-1
68: #else
69:         .error "Definir TEST ente 1..19"
70: #endif
71:         .endm
72: lista:          linea0
73:         .irpc i,123
74:                 linea
75:         .endr
```

Hemos cambiado el formato para imprimir correctamente los datos.
En la función imprimir pasamos los parámetros a determinados registros para que salgan en orden:

rdi, rsi, rdx, rcx, r8 y r9 en ese orden

La media la calculamos con la operación **idivl** que divide **edx:eax** entre el parámetro pasado y almacena el cociente en **eax** y el resto en **edx**. En este caso como dividendo pasamos **ecx** que contiene la longitud de la lista.

En printf usamos 4 registros para imprimir la media y el resto en decimal y en hexadecimal

```
81: formato: .ascii "media \t = %11d \t resto \t = %11d\n"
82:                 .asciz "media \t = 0x %08x \t resto \t = 0x %08x\n"
83:
84: .section .text
85: main: .global  main
86:
87: #trabajar
88:         movq     $lista, %rbx
89:         movl  longlista, %ecx
90:         call suma               # == suma(&lista, longlista);
91:         mov     %esi, %eax
92:         mov     %edi, %edx
93:         idivl %ecx
94:         movl  %eax, media
95:         movl %edx, resto
96:
```

```
101: #imprim_C
102:         movq   $formato, %rdi
103:         movl   media,%esi
104:         movl   resto,%edx
105:         movl   media, %ecx
106:         movl   resto, %r8d
107:         movl          $0,%eax   # varargin sin xmm
108:         call  printf            # == printf(formato, res, res);
```

José Javier Alonso Ramos jotajota98@correo.ugr.es

5.5 Media y resto de N enteros calculada en 32 y en 64 bits (N≈16)

Para su ejecución utilizaremos:
for i in $(seq 1 20); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie media.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done

En este ejercicio creamos los códigos para trabajar, imprimir y sumar de 64 bits.
Para diferenciar la impresión de 32bits de la de 64bits creamos también un nuevo formato.

```
84: formatoq: .ascii "media_x64 \t = %11d \t resto_x64 \t = %11d\n"
85:                  .asciz "media_x64 \t = 0x %08x \t resto_x64 \t = 0x %08x\n"
```

Ahora en vez de almacenar la suma en dos registros de 32bits lo haremos en tan solo uno de 64bits.
Copiamos el valor del dato de la lista a **eax** y con la operación **cdqe** extendemos el signo de **eax** → **rax** y este dato lo vamos acumulando en **rdi.**
Una vez acaba el código de suma copiamos el resultado en r**ax** para que con la operación **cqto** extendamos el signo a **rdx** de tal forma que al realizar la división se tenga en cuenta el signo del dato. De nuevo el cociente se almacena en el registro A y el resto en el registro D.

```
149: sumaq:
150:         movq     $0, %r8        # iterador de la lista
151:         movq     $0, %rax       # En un principio se usarÃ¡ para extender el signo a
                                            %edx. Representa la parte menos significativa
152:         movq     $0, %rdi       # Acumulador de la suma
153: bucleq:
154:         movl    (%rbx,%r8,4), %eax
155:         cdqe
156:         add     %rax, %rdi
157:         inc     %r8
158:         cmpq    %r8,%rcx
159:         jne     bucleq
160:
161:         ret
162:


110:         movq     $lista, %rbx
111:         movq  longlista, %rcx
112:         call sumaq              # == suma(&lista, longlista);
113:         mov     %rdi, %rax
114:         cqto
115:         idivq %rcx
116:         movq  %rax, media
117:         movq %rdx, resto
118:
119: #imprim_C_q
120:         movq   $formatoq, %rdi
121:         movq   media,%rsi
122:         movq   resto,%rdx
123:         movq   media, %rcx
124:         movq   resto, %r8
125:         movq          $0,%rax   # varargin sin xmm
126:         call  printf            # == printf(formato, res, res);
```

José Javier Alonso Ramos jotajota98@correo.ugr.es

Durante la realización de la práctica se ha dedicado a la resolución de los problemas una semana habiendo dedicado las primeras dos a la lectura, comprensión y estudio de las prácticas y lenguaje ensamblador. Desde el lunes hasta el viernes se han realizado los ejercicios (uno por día). Siendo hoy domingo el último día disponible para la entrega de la práctica se ha realizado una revisión del trabajo realizado días atrás corrigiendo así algunos fallos y adecuando mejor el código y sus comentarios. También se ha llevado a cabo hoy la realización de la memoria.

A continuación pondré todo el código de los programas (en las páginas anteriores solo se adjunta parte del código referente a la explicación) y las salidas de su ejecución:

5.1 código

```
1: #EJERCICIO 5.1
2: #
3: # Se han seguido los pasos tal y como se indican en el guiÃ³n.
4: # 'lista' se ha dividido en cuatro lineas para una lectura mÃ¡s cÃ³moda del cÃ³digo
5: # 'resultado' se ha cambiado por tipo de dato 'quad' para permitir imprimir datos de
   64 bits
6: # 'formato' ha sido modificado para que la impresion mediante 'printf' fuera posible
   y mostrase adecuadamente los datos (unsigned long y hexadecimal long)
7: # '_start' fue cambiado por 'main' para poder compilar mediante gcc y las funciones
  fueron sustituidas por su cÃ³digo directamente
8: # A muchas instrucciones se les ha aÃ±adido un subfijo para ir acostumbrandonos a su
   uso y significado
9:
10:
11: .section .data
12: lista:          .int 0x10000000,0x10000000,0x10000000,0x10000000
13:                       .int 0x10000000,0x10000000,0x10000000,0x10000000
14:                       .int 0x10000000,0x10000000,0x10000000,0x10000000
15:                       .int 0x10000000,0x10000000,0x10000000,0x10000000
16: longlista:      .int   (.-lista)/4
17: resultado:      .quad   0
18: formato:        .asciz  "suma = %lu = 0x%lx hex\n"
19:
20: .section .text
21: main: .global  main
22:
23: #trabajar
24:         movq     $lista, %rbx
25:         movl  longlista, %ecx
26:         call suma                # == suma(&lista, longlista);
27:         movl  %eax, resultado
28:         movl %edx, resultado+4
29:
30:         # Como 'resultado' es de 64 bits, es almacenado en pila y la arquitectura ut
              ilizada almacena los datos en 'little endian'
31:         # su parte mÃ¡s significativa (%edx) tiene que ser guarda antes que la menos
              significativa (%eax)
32:         # por eso almacenamos %edx en resultado+4 y %eax en resultado
33:
34: #imprim_C
35:         movq   $formato, %rdi
36:         movq   resultado,%rsi
37:         movq   resultado,%rdx
38:         movl           $0,%eax   # varargin sin xmm
39:         call  printf             # == printf(formato, res, res);
40:
41:         # SegÃºn el manual de 'prinf' formato debe ser especificado en %rdi,
42:         # el primer resultado a mostrar (unsigned long) en %rsi y el segundo (hexade
              cimal long) en %rdx
43:
44: #acabar_C
45:         mov  resultado, %edi
46:         call _exit               # ==  exit(resultado)
47:         ret
48:
49: suma:
50:         movq     $0, %rsi        # iterador de la lista
51:         movl     $0, %eax        # acumulador de la suma. Representa la parte menos s
                                       ignificativa
52:         movl     $0, %edx        # acumulador de la suma. Representa la parte mÃ¡s si
                                       gnificativa
53: bucle:
54:         addl    (%rbx,%rsi,4), %eax
55:         jnc              no_inc          # si no hay acarreo no incrementamos %edx y
                                               seguimos con las iteraciones del bucle
56:         inc     %edx
57: no_inc:
58:         inc     %rsi
```

```
media.s       Sun Oct 14 19:19:35 2018       2
    59:         cmpq    %rsi,%rcx
    60:         jne     bucle
    61:
    62:         ret
    63:
```

5.1 salida

suma = 4294967296 = 0x100000000 hex

5.2 código

```
    1: #EJERCICIO 5.2
    2: #
    3: # Se ha usado el uso de una macro para poder cambiar la 'lista' de nÃºmeros que se v
an a sumar
    4: # A la hora de imprimir hemos usado los registros %ecx y %r8d para imprimir el numer
o hexadecimal en dos partes
    5: # En el cÃ³digo 'suma' hemos cambiado la implementaciÃ³n de la suma del acarreo
    6: # en vez de usar un salto condicional utilizamos la instrucciÃ³n 'adc' que contempla
 el acarreo
    7: # Por lo demÃ¡s el cÃ³digo es igual al del 5.1
    8:
    9: #COMANDO PARA LA EJECUCIÃ\223N:
   10: #for i in $(seq 1 9); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie medi
a.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done
   11:
   12:
   13: .section .data
   14: #ifndef TEST
   15: #define TEST 9
   16: #endif
   17: .macro linea
   18: #if TEST==1                                        // 16 â\200\223 ejemplo muy sencillo
   19:         .int 1,1,1,1
   20: #elif TEST==2                                     // 0x0 ffff fff0, casi acarreo
   21:         .int 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff
   22: #elif TEST==3                                     // 0x10000000, justo 1 acarreo
   23:         .int 0x10000000,0x10000000,0x10000000,0x10000000
   24: #elif TEST==4
   25:         .int 0xffffffff,0xffffffff,0xffffffff,0xffffffff
   26: #elif TEST==5                                     // no trabaja con numeros negativos
   27:         .int -1,-1,-1,-1
   28: #elif TEST==6
   29:         .int 200000000,200000000,200000000,200000000
   30: #elif TEST==7
   31:         .int 300000000,300000000,300000000,300000000
   32: #elif TEST==8                                     // 11 280 523 264 << 16x5e9= 80e9
   33:         .int 5000000000,5000000000,5000000000,5000000000
   34: #else
   35:         .error "Definir TEST entre 1..8"
   36: #endif
   37:         .endm
   38:
   39: lista: .irpc i,1234
   40:                         linea
   41:         .endr
   42:
   43: longlista:      .int   (.-lista)/4
   44: resultado:      .quad   0
   45: formato: .ascii "resultado \t = %18lu (uns)\n"
   46:                 .ascii "\t\t = 0x%18lx (hex)\n"
   47:                 .asciz "\t\t = 0x %08x %08x \n"
   48:
   49: .section .text
   50: main: .global  main
```

```
   52: #trabajar
   53:         movq     $lista, %rbx
   54:         movl  longlista, %ecx
   55:         call suma               # == suma(&lista, longlista);
   56:         movl  %eax, resultado
   57:         movl %edx, resultado+4
   58:
   59:         # Como 'resultado' es de 64 bits, es almacenado en pila y la arquitectura ut
ilizada almacena los datos en 'little endian'
   60:         # su parte mÃ¡s significativa (%edx) tiene que ser guarda antes que la menos
 significativa (%eax)
   61:         # por eso almacenamos %edx en resultado+4 y %eax en resultado
   62:
   63: #imprim_C
   64:         movq   $formato, %rdi
   65:         movq   resultado,%rsi
   66:         movq   resultado,%rdx
   67:         movl          $0,%eax   # varargin sin xmm
   68:         movl   resultado+4, %ecx
   69:         movl   resultado, %r8d
   70:         call  printf            # == printf(formato, res, res);
   71:
   72:         # SegÃºn el manual de 'prinf' formato debe ser especificado en %rdi,
   73:         # el primer resultado a mostrar (unsigned long) en %rsi y el segundo (hexade
cimal long) en %rdx
   74:
   75: #acabar_C
   76:         mov  resultado, %edi
   77:         call _exit              # ==  exit(resultado)
   78:         ret
   79:
   80: suma:
   81:         movq     $0, %rsi       # iterador de la lista
   82:         movl     $0, %eax       # acumulador de la suma. Representa la parte menos s
ignificativa
   83:         movl     $0, %edx       # acumulador de la suma. Representa la parte mÃ¡s si
gnificativa
   84: bucle:
   85:         addl    (%rbx,%rsi,4), %eax
   86:         adc     $0, %edx
   87:         inc     %rsi
   88:         cmpq    %rsi,%rcx
   89:         jne     bucle
   90:
   91:         ret
   92:
```

5.2 salida

```
rm: no se puede borrar 'media': No existe el archivo o el directorio
__TEST01__---------------------------------
resultado       =              16 (uns)
                = 0x            10 (hex)
                = 0x 00000000 00000010
__TEST02__---------------------------------
resultado       =      4294967280 (uns)
                = 0x        fffffff0 (hex)
                = 0x 00000000 fffffff0
__TEST03__---------------------------------
```

```
resultado      =       4294967296 (uns)
               = 0x       100000000 (hex)
               = 0x 00000001 00000000
__TEST04__----------------------------------
resultado      =      68719476720 (uns)
               = 0x       fffffff0 (hex)
               = 0x 0000000f fffffff0
__TEST05__----------------------------------
resultado      =      68719476720 (uns)
               = 0x       fffffff0 (hex)
               = 0x 0000000f fffffff0
__TEST06__----------------------------------
resultado      =       3200000000 (uns)
               = 0x        bebc2000 (hex)
               = 0x 00000000 bebc2000
__TEST07__----------------------------------
resultado      =       4800000000 (uns)
               = 0x       11e1a3000 (hex)
               = 0x 00000001 1e1a3000
media.s: Mensajes del ensamblador:
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
media.s:41: Aviso: valora 0x12a05f200 truncado a 0x2a05f200
__TEST08__----------------------------------
resultado      =      11280523264 (uns)
               = 0x       2a05f2000 (hex)
               = 0x 00000002 a05f2000
media.s: Mensajes del ensamblador:
media.s:41: Error: Definir TEST entre 1..8
media.s:41: Error: Definir TEST entre 1..8
media.s:41: Error: Definir TEST entre 1..8
media.s:41: Error: Definir TEST entre 1..8
__TEST09__----------------------------------
```

5.3 código

```
 1: #EJERCICIO 5.3
 2: #
 3: # Se ha usado el uso de una macro para poder cambiar la ’lista’ de nÃºmeros que se v
    an a sumar
 4: # A la hora de imprimir hemos usado los registros %ecx y %r8d para imprimir el numer
    o hexadecimal en dos partes
 5: # En la suma hemos tenido en cuenta el signo a la hora de hacer operaciones.
 6: # Gracias a la operaciÃ³n ’cdq’ extendemos el signo de eax a edx.
 7: # Utilizamos los registros esi y edi como acumuladores.
 8: # Por lo demÃ¡s el cÃ³digo es igual al del 5.2
 9:
10: #COMANDO PARA LA EJECUCIÃ\223N:
11: #for i in $(seq 1 20); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie med
    ia.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done
12:
13: .section .data
14: #ifndef TEST
15: #define TEST 20
16: #endif
17: .macro linea
18: #if TEST==1                                        // 16 â\200\223 ejemplo muy sencillo
19:         .int -1,-1,-1,-1
20: #elif TEST==2                                     // 1073741824
21:         .int 0x04000000, 0x04000000, 0x04000000, 0x04000000
22: #elif TEST==3                                     // 2147483648
23:         .int 0x08000000,0x08000000,0x08000000,0x08000000
24: #elif TEST==4                                     // 4294967296
25:         .int 0x10000000,0x10000000,0x10000000,0x10000000
26: #elif TEST==5                                     // 34359738352
27:         .int 0x7FFFFFFF,0x7FFFFFFF,0x7FFFFFFF,0x7FFFFFFF
28: #elif TEST==6                                     // -34359738368
29:         .int 0x80000000,0x80000000,0x80000000,0x80000000
30: #elif TEST==7                                     // -4294967296
31:         .int 0xF0000000,0xF0000000,0xF0000000,0xF0000000
32: #elif TEST==8                                     // -2147483648
33:         .int 0xF8000000,0xF8000000,0xF8000000,0xF8000000
34: #elif TEST==9                                     // -2147483664
35:         .int 0xF7FFFFFF,0xF7FFFFFF,0xF7FFFFFF,0xF7FFFFFF
36: #elif TEST==10                                    // 1600000000
37:         .int 100000000,100000000,100000000,100000000
38: #elif TEST==11                                    // 3200000000
39:         .int 200000000,200000000,200000000,200000000
40: #elif TEST==12                                    // 4800000000
41:         .int 300000000,300000000,300000000,300000000
42: #elif TEST==13                                    // 32000000000
43:         .int 2000000000,2000000000,2000000000,2000000000
44: #elif TEST==14                                    // -20719476736 no representable sgn32b(>=2G
                                                                                          i)
45:         .int 3000000000,3000000000,3000000000,3000000000
46: #elif TEST==15                                    // -1600000000
47:         .int -100000000,-100000000,-100000000,-100000000
48: #elif TEST==16                                    // -3200000000
49:         .int -200000000,-200000000,-200000000,-200000000
50: #elif TEST==17                                    // -4800000000
51:         .int -300000000,-300000000,-300000000,-300000000
52: #elif TEST==18                                    // -32000000000
53:         .int -2000000000,-2000000000,-2000000000,-2000000000
54: #elif TEST==19                                    // 20719476736 no representable sgn32b(<-2Gi
                                                                                          )
55:         .int -3000000000,-3000000000,-3000000000,-3000000000
56: #else
57:         .error "Definir TEST entre 1..20"
58: #endif
59:         .endm
60:
61: lista: .irpc i,1234
62:                         linea
63:         .endr
```

```
   64:
   65: longlista:      .int   (.-lista)/4
   66: resultado:      .quad   0
   67: formato: .ascii "resultado \t = %18ld (sgn)\n"
   68:                 .ascii "\t\t = 0x%18lx (hex)\n"
   69:                 .asciz "\t\t = 0x %08x %08x \n"
   70:
   71: .section .text
   72: main: .global  main
   73:
   74: #trabajar
   75:         movq     $lista, %rbx
   76:         movl  longlista, %ecx
   77:         call suma               # == suma(&lista, longlista);
   78:         mov     %esi, %eax
   79:         mov     %edi, %edx
   80:         movl  %eax, resultado
   81:         movl %edx, resultado+4
   82:
   83:         # Como ’resultado’ es de 64 bits, es almacenado en pila y la arquitectura ut
ilizada almacena los datos en ’little endian’
   84:         # su parte mÃ¡s significativa (%edx) tiene que ser guarda antes que la menos
 significativa (%eax)
   85:         # por eso almacenamos %edx en resultado+4 y %eax en resultado
   86:
   87: #imprim_C
   88:         movq   $formato, %rdi
   89:         movq   resultado,%rsi
   90:         movq   resultado,%rdx
   91:         movl   resultado+4, %ecx
   92:         movl   resultado, %r8d
   93:         movl          $0,%eax   # varargin sin xmm
   94:         call  printf            # == printf(formato, res, res);
   95:
   96:         # SegÃºn el manual de ’prinf’ formato debe ser especificado en %rdi,
   97:         # el primer resultado a mostrar (unsigned long) en %rsi y el segundo (hexade
cimal long) en %rdx
   98:
   99: #acabar_C
  100:         mov  resultado, %edi
  101:         call _exit              # ==  exit(resultado)
  102:         ret
  103:
  104: suma:
  105:         movq     $0, %r8        # iterador de la lista
  106:         movl     $0, %eax       # En un principio se usarÃ¡ para extender el signo a
 %edx. Representa la parte menos significativa
  107:         movl     $0, %esi       # Acumulador de la suma. Representa la parte menos s
ignificativa
  108:         movl     $0, %edi       # Acumulador de la suma. Representa la parte mÃ¡s si
gnificativa
  109: bucle:
  110:         movl    (%rbx,%r8,4), %eax
  111:         cdq
  112:         add     %eax, %esi
  113:         adc     %edx, %edi
  114:         inc     %r8
  115:         cmpq    %r8,%rcx
  116:         jne     bucle
  117:
  118:         ret
  119:
```

5.3 salida

```
rm: no se puede borrar 'media': No existe el archivo o el directorio
__TEST01__---------------------------------
resultado       =               -16 (sgn)
                = 0x  fffffffffffffff0 (hex)
                = 0x ffffffff fffffff0
__TEST02__---------------------------------
resultado       =        1073741824 (sgn)
                = 0x         40000000 (hex)
                = 0x 00000000 40000000
__TEST03__---------------------------------
resultado       =        2147483648 (sgn)
                = 0x         80000000 (hex)
                = 0x 00000000 80000000
__TEST04__---------------------------------
resultado       =        4294967296 (sgn)
                = 0x        100000000 (hex)
                = 0x 00000001 00000000
__TEST05__---------------------------------
resultado       =       34359738352 (sgn)
                = 0x        7fffffff0 (hex)
                = 0x 00000007 fffffff0
__TEST06__---------------------------------
resultado       =      -34359738368 (sgn)
                = 0x  fffffff800000000 (hex)
                = 0x fffffff8 00000000
__TEST07__---------------------------------
resultado       =       -4294967296 (sgn)
                = 0x  ffffffff00000000 (hex)
                = 0x ffffffff 00000000
__TEST08__---------------------------------
resultado       =       -2147483648 (sgn)
                = 0x  ffffffff80000000 (hex)
                = 0x ffffffff 80000000
__TEST09__---------------------------------
resultado       =       -2147483664 (sgn)
                = 0x  ffffffff7ffffff0 (hex)
                = 0x ffffffff 7ffffff0
__TEST10__---------------------------------
resultado       =        1600000000 (sgn)
                = 0x         5f5e1000 (hex)
                = 0x 00000000 5f5e1000
__TEST11__---------------------------------
resultado       =        3200000000 (sgn)
                = 0x         bebc2000 (hex)
                = 0x 00000000 bebc2000
```

```
__TEST12__----------------------------------
resultado       =        4800000000 (sgn)
                = 0x        11e1a3000 (hex)
                = 0x 00000001 1e1a3000
__TEST13__----------------------------------
resultado       =       32000000000 (sgn)
                = 0x        773594000 (hex)
                = 0x 00000007 73594000
__TEST14__----------------------------------
resultado       =      -20719476736 (sgn)
                = 0x  fffffffb2d05e000 (hex)
                = 0x fffffffb 2d05e000
__TEST15__----------------------------------
resultado       =       -1600000000 (sgn)
                = 0x  ffffffffa0a1f000 (hex)
                = 0x ffffffff a0a1f000
__TEST16__----------------------------------
resultado       =       -3200000000 (sgn)
                = 0x  ffffffff4143e000 (hex)
                = 0x ffffffff 4143e000
__TEST17__----------------------------------
resultado       =       -4800000000 (sgn)
                = 0x  fffffffee1e5d000 (hex)
                = 0x fffffffe e1e5d000
__TEST18__----------------------------------
resultado       =      -32000000000 (sgn)
                = 0x  fffffff88ca6c000 (hex)
                = 0x fffffff8 8ca6c000
media.s: Mensajes del ensamblador:
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:63: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
__TEST19__----------------------------------
```

resultado = 20719476736 (sgn)
= 0x 4d2fa2000 (hex)
= 0x 00000004 d2fa2000
media.s: Mensajes del ensamblador:
media.s:63: Error: Definir TEST entre 1..5
media.s:63: Error: Definir TEST entre 1..5
media.s:63: Error: Definir TEST entre 1..5
media.s:63: Error: Definir TEST entre 1..5
__TEST20__----------
bash: ./media: No existe el archivo o el directorio

5.4 código

```
    1: #EJERCICIO 5.4
    2: #
    3: # Se ha usado el uso de dos macros para poder cambiar la 'lista' de nÃºmeros que se
van a sumar segÃºn explica el guiÃ³n
    4: # A la hora de imprimir hemos usado los registros %rsi y %rdx para imprimir los resu
ltados en decimal de la media y resto respectivamente
    5: # y %ecx y %r8d para imprimirlos en hexadecimal
    6: # La media la hemos realizado con 'idiv'. Divide edx:eax entre el parametro pasado.
Almacena el cociente en eax y el resto en edx
    7: # Hemos creado dos variables 'media' y 'resto' para hacer mas legible el codigo
    8: # Por lo demÃ¡s el cÃ³digo es igual al del 5.3
    9:
   10: #COMANDO PARA LA EJECUCIÃ\223N:
   11: #for i in $(seq 1 20); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie med
ia.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done
   12:
   13: .section .data
   14: #ifndef TEST
   15: #define TEST 20
   16: #endif
   17: .macro linea
   18: #if  TEST==1                    //1             8
   19:         .int 1,2,1,2
   20: #elif TEST==2                    //-1    -8
   21:         .int -1,-2,-1,-2
   22: #elif TEST==3                    //2147483647    0
   23:         .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
   24: #elif TEST==4                    //-2147483648   0
   25:         .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
   26: #elif TEST==5                    //-1    0
   27:         .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
   28: #elif TEST==6                    //2000000000    0
   29:         .int 2000000000, 2000000000, 2000000000, 2000000000
   30: #elif TEST==7                    //desbordamiento--> -1294967296 0
   31:         .int 3000000000, 3000000000, 3000000000, 3000000000
   32: #elif TEST==8                    //-2000000000   0
   33:         .int -2000000000, -2000000000, -2000000000, -2000000000
   34: #elif TEST==9                    //desbordamiento--> 1294967296  0
   35:         .int -3000000000, -3000000000, -3000000000, -3000000000
   36: #elif TEST>=10 && TEST<=14
   37:         .int 1, 1, 1, 1
   38: #elif TEST>=15 && TEST<=19
   39:         .int -1, -1, -1, -1
   40: #else                   //
   41:         .error "Definir TEST ente 1..19"
   42: #endif                  //
   43:         .endm
```

```
   45: .macro linea0
   46: #if TEST>=1 && TEST<=9
   47:         linea
   48: #elif TEST==10                  //1     0
   49:         .int 0,2,1,1
   50: #elif TEST==11                  //1     1
   51:         .int 1,2,1,1
   52: #elif TEST==12                  //1     8
   53:         .int 8,2,1,1
   54: #elif TEST==13                  //1     15
   55:         .int 15,2,1,1
   56: #elif TEST==14                  //2     0
   57:         .int 16,2,1,1
   58: #elif TEST==15                  //-1    0
   59:         .int 0,-2,-1,-1
   60: #elif TEST==16                  //-1    -1
   61:         .int -1,-2,-1,-1
   62: #elif TEST==17                  //-1    -8
   63:         .int -8,-2,-1,-1
   64: #elif TEST==18                  //-1    -15
   65:         .int -15,-2,-1,-1
   66: #elif TEST==19                  //-2    0
   67:         .int -16,-2,-1,-1
   68: #else
   69:         .error "Definir TEST ente 1..19"
   70: #endif
   71:         .endm
   72: lista:          linea0
   73:         .irpc i,123
   74:                 linea
   75:         .endr
   76:
   77: longlista:      .int   (.-lista)/4
   78: resultado:      .quad   0
   79: media:          .int    0
   80: resto:          .int    0
   81: formato: .ascii "media \t = %11d \t resto \t = %11d\n"
   82:                  .asciz "media \t = 0x %08x \t resto \t = 0x %08x\n"
   83:
   84: .section .text
   85: main: .global  main
   86:
   87: #trabajar
   88:         movq     $lista, %rbx
   89:         movl  longlista, %ecx
   90:         call suma               # == suma(&lista, longlista);
   91:         mov     %esi, %eax
   92:         mov     %edi, %edx
   93:         idivl %ecx
   94:         movl  %eax, media
   95:         movl %edx, resto
   96:
   97:         # Como ’resultado’ es de 64 bits, es almacenado en pila y la arquitectura ut
ilizada almacena los datos en ’little endian’
   98:         # su parte mÃ¡s significativa (%edx) tiene que ser guarda antes que la menos
 significativa (%eax)
   99:         # por eso almacenamos %edx en resultado+4 y %eax en resultado
  100:
  101: #imprim_C
  102:         movq   $formato, %rdi
  103:         movl   media,%esi
  104:         movl   resto,%edx
  105:         movl   media, %ecx
  106:         movl   resto, %r8d
  107:         movl          $0,%eax   # varargin sin xmm
  108:         call  printf            # == printf(formato, res, res);
  109:
  110:         # SegÃºn el manual de ’prinf’ formato debe ser especificado en %rdi,
  111:         # el primer resultado a mostrar (unsigned long) en %rsi y el segundo (hexade
cimal long) en %rdx
```

```
 113: #acabar_C
 114:         mov  resultado, %edi
 115:         call _exit              # ==  exit(resultado)
 116:         ret
 117:
 118: suma:
 119:         movq     $0, %r8        # iterador de la lista
 120:         movl     $0, %eax       # En un principio se usarÃ¡ para extender el signo a
%edx. Representa la parte menos significativa
 121:         movl     $0, %esi       # Acumulador de la suma. Representa la parte menos s
ignificativa
 122:         movl     $0, %edi       # Acumulador de la suma. Representa la parte mÃ¡s si
gnificativa
 123: bucle:
 124:         movl    (%rbx,%r8,4), %eax
 125:         cdq
 126:         add     %eax, %esi
 127:         adc     %edx, %edi
 128:         inc     %r8
 129:         cmpq    %r8,%rcx
 130:         jne     bucle
 131:
 132:         ret
 133:
```

5.4 salida

```
rm: no se puede borrar 'media': No existe el archivo o el directorio
__TEST01__---------------------------------
media  =         1      resto  =         8
media  = 0x 00000001       resto  = 0x 00000008
__TEST02__---------------------------------
media  =        -1      resto  =        -8
media  = 0x ffffffff   resto  = 0x fffffff8
__TEST03__---------------------------------
media  =  2147483647       resto  =         0
media  = 0x 7fffffff   resto  = 0x 00000000
__TEST04__---------------------------------
media  = -2147483648       resto  =         0
media  = 0x 80000000       resto  = 0x 00000000
__TEST05__---------------------------------
media  =        -1      resto  =         0
media  = 0x ffffffff   resto  = 0x 00000000
__TEST06__---------------------------------
media  =  2000000000       resto  =         0
media  = 0x 77359400       resto  = 0x 00000000
__TEST07__---------------------------------
media  = -1294967296       resto  =         0
media  = 0x b2d05e00       resto  = 0x 00000000
__TEST08__---------------------------------
media  = -2000000000       resto  =         0
media  = 0x 88ca6c00       resto  = 0x 00000000
media.s: Mensajes del ensamblador:
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
```

```
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
__TEST09__----------------------------------
media  =  1294967296      resto   =         0
media  = 0x 4d2fa200      resto   = 0x 00000000
__TEST10__----------------------------------
media  =         1    resto   =         0
media  = 0x 00000001      resto   = 0x 00000000
__TEST11__----------------------------------
media  =         1    resto   =         1
media  = 0x 00000001      resto   = 0x 00000001
__TEST12__----------------------------------
media  =         1    resto   =         8
media  = 0x 00000001      resto   = 0x 00000008
__TEST13__----------------------------------
media  =         1    resto   =        15
media  = 0x 00000001      resto   = 0x 0000000f
__TEST14__----------------------------------
media  =         2    resto   =         0
media  = 0x 00000002      resto   = 0x 00000000
__TEST15__----------------------------------
media  =        -1    resto   =         0
media  = 0x ffffffff  resto   = 0x 00000000
__TEST16__----------------------------------
media  =        -1    resto   =        -1
media  = 0x ffffffff  resto   = 0x ffffffff
__TEST17__----------------------------------
media  =        -1    resto   =        -8
media  = 0x ffffffff  resto   = 0x fffffff8
__TEST18__----------------------------------
media  =        -1    resto   =       -15
media  = 0x ffffffff  resto   = 0x fffffff1
__TEST19__----------------------------------
```

```
media =       -2     resto  =        0
media  = 0x fffffffe   resto  = 0x 00000000
media.s: Mensajes del ensamblador:
media.s:72: Error: Definir TEST ente 1..19
media.s:75: Error: Definir TEST ente 1..19
media.s:75: Error: Definir TEST ente 1..19
media.s:75: Error: Definir TEST ente 1..19
__TEST20__--------------------------------
bash: ./media: No existe el archivo o el directorio
```

5.5 código

```
    1: #EJERCICIO 5.5
    2: #
    3: # Se ha usado el uso de dos macros para poder cambiar la 'lista' de nÃºmeros que se
van a sumar segÃºn explica el guiÃ³n
    4: # A la hora de imprimir hemos usado los registros %rsi y %rdx para imprimir los resu
ltados en decimal de la media y resto respectivamente
    5: # y %rcx y %r8 para imprimirlos en hexadecimal
    6: # La media la hemos realizado con 'idiv'. Divide edx:eax entre el parametro pasado.
Almacena el cociente en eax y el resto en edx
    7: # La media la hemos realizado con 'idiv'. Divide rdx:rax entre el parametro pasado.
Almacena el cociente en rax y el resto en rdx
    8: # Por lo demÃ¡s el cÃ³digo es igual al del 5.4
    9:
   10: #COMANDO PARA LA EJECUCIÃ\223N:
   11: #for i in $(seq 1 20); do rm media; gcc -x assembler-with-cpp -D TEST=$i -no-pie med
ia.s -o media; printf "__TEST%02d__%35s\n" $i "" | tr " " "-" ; ./media; done
   12:
   13: .section .data
   14: #ifndef TEST
   15: #define TEST 20
   16: #endif
   17: .macro linea
   18: #if  TEST==1                    //1             8
   19:         .int 1,2,1,2
   20: #elif TEST==2                    //-1    -8
   21:         .int -1,-2,-1,-2
   22: #elif TEST==3                    //2147483647    0
   23:         .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
   24: #elif TEST==4                    //-2147483648   0
   25:         .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
   26: #elif TEST==5                    //-1    0
   27:         .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
   28: #elif TEST==6                    //2000000000    0
   29:         .int 2000000000, 2000000000, 2000000000, 2000000000
   30: #elif TEST==7                    //desbordamiento--> -1294967296 0
   31:         .int 3000000000, 3000000000, 3000000000, 3000000000
   32: #elif TEST==8                    //-2000000000   0
   33:         .int -2000000000, -2000000000, -2000000000, -2000000000
   34: #elif TEST==9                    //desbordamiento--> 1294967296  0
   35:         .int -3000000000, -3000000000, -3000000000, -3000000000
   36: #elif TEST>=10 && TEST<=14
   37:         .int 1, 1, 1, 1
   38: #elif TEST>=15 && TEST<=19
   39:         .int -1, -1, -1, -1
   40: #else                   //
   41:         .error "Definir TEST ente 1..19"
   42: #endif                  //
   43:         .endm
   44:
```

```
 45: .macro linea0
 46: #if TEST>=1 && TEST<=9
 47:         linea
 48: #elif TEST==10                  //1     0
 49:         .int 0,2,1,1
 50: #elif TEST==11                  //1     1
 51:         .int 1,2,1,1
 52: #elif TEST==12                  //1     8
 53:         .int 8,2,1,1
 54: #elif TEST==13                  //1     15
 55:         .int 15,2,1,1
 56: #elif TEST==14                  //2     0
 57:         .int 16,2,1,1
 58: #elif TEST==15                  //-1    0
 59:         .int 0,-2,-1,-1
 60: #elif TEST==16                  //-1    -1
 61:         .int -1,-2,-1,-1
 62: #elif TEST==17                  //-1    -8
 63:         .int -8,-2,-1,-1
 64: #elif TEST==18                  //-1    -15
 65:         .int -15,-2,-1,-1
 66: #elif TEST==19                  //-2    0
 67:         .int -16,-2,-1,-1
 68: #else
 69:         .error "Definir TEST ente 1..19"
 70: #endif
 71:         .endm
 72: lista:          linea0
 73:         .irpc i,123
 74:                 linea
 75:         .endr
 76:
 77: longlista:      .int   (.-lista)/4
 78: resultado:      .quad   0
 79: media:          .quad   0
 80: resto:          .quad   0
 81: formato: .ascii "media \t = %11d \t resto \t = %11d\n"
 82:                  .asciz "media \t = 0x %08x \t resto \t = 0x %08x\n"
 83:
 84: formatoq: .ascii "media_x64 \t = %11d \t resto_x64 \t = %11d\n"
 85:                  .asciz "media_x64 \t = 0x %08x \t resto_x64 \t = 0x %08x\n"
 86:
 87: .section .text
 88: main: .global  main
 89:
 90: #trabajar
 91:         movq     $lista, %rbx
 92:         movl  longlista, %ecx
 93:         call suma               # == suma(&lista, longlista);
 94:         mov     %esi, %eax
 95:         mov     %edi, %edx
 96:         idivl %ecx
 97:         movl  %eax, media
 98:         movl %edx, resto
 99:
100: #imprim_C
101:         movq   $formato, %rdi
102:         movl   media,%esi
103:         movl   resto,%edx
104:         movl   media, %ecx
105:         movl   resto, %r8d
106:         movl          $0,%eax   # varargin sin xmm
107:         call  printf            # == printf(formato, res, res);
108:
109: #trabajar_q
110:         movq     $lista, %rbx
111:         movq  longlista, %rcx
112:         call sumaq              # == suma(&lista, longlista);
113:         mov     %rdi, %rax
114:         cqto
115:         idivq %rcx
116:         movq  %rax, media
117:         movq %rdx, resto
118:
```

```
  132:
  133: suma:
  134:          movq      $0, %r8         # iterador de la lista
  135:          movl      $0, %eax        # En un principio se usarÃ¡ para extender el signo a
 %edx. Representa la parte menos significativa
  136:          movl      $0, %esi        # Acumulador de la suma. Representa la parte menos s
ignificativa
  137:          movl      $0, %edi        # Acumulador de la suma. Representa la parte mÃ¡s si
gnificativa
  138: bucle:
  139:          movl     (%rbx,%r8,4), %eax
  140:          cltd
  141:          add      %eax, %esi
  142:          adc      %edx, %edi
  143:          inc      %r8
  144:          cmpq     %r8,%rcx
  145:          jne      bucle
  146:
  147:          ret
  148:
  149: sumaq:
  150:          movq      $0, %r8         # iterador de la lista
  151:          movq      $0, %rax        # En un principio se usarÃ¡ para extender el signo a
 %edx. Representa la parte menos significativa
  152:          movq      $0, %rdi        # Acumulador de la suma
  153: bucleq:
  154:          movl     (%rbx,%r8,4), %eax
  155:          cdqe
  156:          add      %rax, %rdi
  157:          inc      %r8
  158:          cmpq     %r8,%rcx
  159:          jne      bucleq
  160:
  161:          ret
  162:
```

5.5 salida

```
rm: no se puede borrar 'media': No existe el archivo o el directorio
__TEST01__---------------------------------
media  =         1      resto  =         8
media  = 0x 00000001        resto  = 0x 00000008
media_x64     =         1      resto_x64     =         8
media_x64     = 0x 00000001        resto_x64     = 0x 00000008
__TEST02__---------------------------------
media  =        -1      resto  =        -8
media  = 0x ffffffff    resto  = 0x fffffff8
media_x64     =        -1      resto_x64     =        -8
media_x64     = 0x ffffffff    resto_x64     = 0x fffffff8
__TEST03__---------------------------------
media  =  2147483647        resto  =         0
media  = 0x 7fffffff    resto  = 0x 00000000
media_x64     =  2147483647        resto_x64     =         0
media_x64     = 0x 7fffffff    resto_x64     = 0x 00000000
__TEST04__---------------------------------
media  = -2147483648        resto  =         0
media  = 0x 80000000        resto  = 0x 00000000
```

```
media_x64    = -2147483648      resto_x64     =         0
media_x64    = 0x 80000000      resto_x64     = 0x 00000000
__TEST05__----------------------------------
media  =         -1     resto   =         0
media  = 0x ffffffff   resto   = 0x 00000000
media_x64    =         -1     resto_x64     =         0
media_x64    = 0x ffffffff   resto_x64     = 0x 00000000
__TEST06__----------------------------------
media  =  2000000000      resto   =         0
media  = 0x 77359400      resto   = 0x 00000000
media_x64    =  2000000000      resto_x64     =         0
media_x64    = 0x 77359400      resto_x64     = 0x 00000000
__TEST07__----------------------------------
media  = -1294967296      resto   =         0
media  = 0x b2d05e00      resto   = 0x 00000000
media_x64    = -1294967296      resto_x64     =         0
media_x64    = 0x b2d05e00      resto_x64     = 0x 00000000
__TEST08__----------------------------------
media  = -2000000000      resto   =         0
media  = 0x 88ca6c00      resto   = 0x 00000000
media_x64    = -2000000000      resto_x64     =         0
media_x64    = 0x 88ca6c00      resto_x64     = 0x 00000000
media.s: Mensajes del ensamblador:
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:72: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
media.s:75: Aviso: valora 0xffffffff4d2fa200 truncado a 0x4d2fa200
__TEST09__----------------------------------
media  =  1294967296      resto   =         0
media  = 0x 4d2fa200      resto   = 0x 00000000
media_x64    =  1294967296      resto_x64     =         0
media_x64    = 0x 4d2fa200      resto_x64     = 0x 00000000
__TEST10__----------------------------------
media  =          1     resto   =         0
```

```
media  = 0x 00000001       resto   = 0x 00000000
media_x64     =          1      resto_x64     =          0
media_x64     = 0x 00000001       resto_x64     = 0x 00000000
__TEST11__----------------------------------
media  =          1      resto   =          1
media  = 0x 00000001       resto   = 0x 00000001
media_x64     =          1      resto_x64     =          1
media_x64     = 0x 00000001       resto_x64     = 0x 00000001
__TEST12__----------------------------------
media  =          1      resto   =          8
media  = 0x 00000001       resto   = 0x 00000008
media_x64     =          1      resto_x64     =          8
media_x64     = 0x 00000001       resto_x64     = 0x 00000008
__TEST13__----------------------------------
media  =          1      resto   =         15
media  = 0x 00000001       resto   = 0x 0000000f
media_x64     =          1      resto_x64     =         15
media_x64     = 0x 00000001       resto_x64     = 0x 0000000f
__TEST14__----------------------------------
media  =          2      resto   =          0
media  = 0x 00000002       resto   = 0x 00000000
media_x64     =          2      resto_x64     =          0
media_x64     = 0x 00000002       resto_x64     = 0x 00000000
__TEST15__----------------------------------
media  =         -1      resto   =          0
media  = 0x ffffffff       resto   = 0x 00000000
media_x64     =         -1      resto_x64     =          0
media_x64     = 0x ffffffff       resto_x64     = 0x 00000000
__TEST16__----------------------------------
media  =         -1      resto   =         -1
media  = 0x ffffffff       resto   = 0x ffffffff
media_x64     =         -1      resto_x64     =         -1
media_x64     = 0x ffffffff       resto_x64     = 0x ffffffff
__TEST17__----------------------------------
media  =         -1      resto   =         -8
media  = 0x ffffffff       resto   = 0x fffffff8
media_x64     =         -1      resto_x64     =         -8
media_x64     = 0x ffffffff       resto_x64     = 0x fffffff8
__TEST18__----------------------------------
media  =         -1      resto   =        -15
media  = 0x ffffffff       resto   = 0x fffffff1
media_x64     =         -1      resto_x64     =        -15
media_x64     = 0x ffffffff       resto_x64     = 0x fffffff1
__TEST19__----------------------------------
media  =         -2      resto   =          0
media  = 0x fffffffe       resto   = 0x 00000000
```

media_x64 = -2 resto_x64 = 0
media_x64 = 0x fffffffe resto_x64 = 0x 00000000
media.s: Mensajes del ensamblador:
media.s:72: Error: Definir TEST ente 1..19
media.s:75: Error: Definir TEST ente 1..19
media.s:75: Error: Definir TEST ente 1..19
media.s:75: Error: Definir TEST ente 1..19
__TEST20__----------
bash: ./media: No existe el archivo o el directorio