PRÁCTICA 2: PROGRAMACIÓN ENSAMBLADOR x86-64 LINUX

Juan Manuel Rodríguez Gómez

Doble Grado en Ingeniería Informática y Matemáticas

Estructura de Computadores

Curso 2020 – 2021

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)

A continuación se puede observar el código en ensamblador para realizar la suma pedida:

```
# Sección de datos

.section .data

   lista:   .int 0x100000000, 0x100000000, 0x100000000, 0x100000000        # Lista de números
            .int 0x100000000, 0x100000000, 0x100000000, 0x100000000
            .int 0x100000000, 0x100000000, 0x100000000, 0x100000000
            .int 0x100000000, 0x100000000, 0x100000000, 0x100000000

   longlista:   .int   (.-lista)/4                    # Longitud de la lista
   resultado: .quad   0                               # Resultado de la suma
   formato:  .ascii "resultado \t =  %18lu (uns)\n"   # Formato de la salida en la terminal del resultado
             .ascii "\t\t = 0x%18lx (hex)\n"
             .asciz "\t\t = 0x %08x %08x\n"

# Sección de código

.section .text

   main: .global  main

         call trabajar                          # Suma sin signo de los números de la lista
         call imprimir                          # printf() de libC
         call acabar                            # exit() de libC
         ret

   trabajar:

         mov  $lista, %rbx                      # Movemos el primer valor de lista al registro %rbx
         mov  longlista, %rcx                   # Movemos la longitud de la lista al registro %rcx
         call   suma                            # == suma(&lista, longlista);
         mov  %eax, resultado
         mov  %edx, resultado+4

   imprimir:                  # requiere libC

         mov   $formato, %rdi
         mov   resultado, %rsi
         mov   resultado, %rdx
         mov   resultado, %ecx
         mov   resultado, %r8d
         mov   $0, %eax
         call    printf                         # == printf(formato, res, res);
```

```
acabar:                 # requiere libC

        movl $1, %eax
        xor  %ebx, %ebx
        int  $0x80                      # En caso de que %eax valga 1, terminamos la
                                        # ejecución y retornamos %ebx

# EDX:EAX

suma:

        mov  $0, %eax                   # Registro de la suma (lo inicializamos a 0)
        mov  $0, %edx                   # Registro del acarreo (lo inicializamos a 0)
        mov  $0, %rsi                   # Registro del índice iterador del bucle (lo
                                        # inicializamos a 0)

bucle:

        add  (%rbx, %rsi, 4), %eax      # Acumulamos las sumas de cada elemento de lista
        jnc  nocarry                    # Saltamos a nocarry si no hay acarreo (CF = 0)
        inc  %edx                       # Si hay acarreo, incrementamos %edx

nocarry:

        inc  %rsi                       # Incrementamos el índice del bucle
        cmp  %rsi, %rcx                 # Comparamos si el índice actual de la lista es igual
                                        # a la longitud de la lista
        jne  bucle                      # Si no es igual, saltamos al bucle de nuevo

        ret
```

Finalmente, este es el resultado que devuelve el programa tras ejecutarlo en la terminal de Linux:

```
juanma@juanma-VirtualBox:~/Escritorio$ gcc  suma_5_1_JMRG.s -o suma_5_1_JMRG;
./suma_5_1_JMRG
suma_5_1_JMRG.s: Mensajes del ensamblador:
suma_5_1_JMRG.s:5: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:5: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:5: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:5: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:6: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:6: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:6: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:6: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:7: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:7: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:7: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:7: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:8: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:8: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:8: Aviso: el valor 0x100000000 se truncó a 0x0
suma_5_1_JMRG.s:8: Aviso: el valor 0x100000000 se truncó a 0x0
resultado       =                       0 (uns)
                = 0x                    0 (hex)
                = 0x 00000000 00000000
```

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

A continuación se puede observar el código en ensamblador para realizar la suma pedida:

```
# Sección de datos

.section .data

   #ifndef  TEST
   #define TEST 9
   #endif
      .macro linea

   #if     TEST==1
                  .int 1, 1, 1, 1
   #elif   TEST==2
                  .int 0x0fffffff, 0x0fffffff, 0x0fffffff, 0x0fffffff
   #elif   TEST==3
                  .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
   #elif   TEST==4
                  .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
   #elif   TEST==5
                  .int -1, -1, -1, -1
   #elif   TEST==6
                  .int 200000000, 200000000, 200000000, 200000000
   #elif   TEST==7
                  .int 300000000, 300000000, 300000000, 300000000
   #elif   TEST==8
                  .int 5000000000, 5000000000, 5000000000, 5000000000
   #else
                  .error "DEFINIR TEST ENTRE 1...8"
   #endif
          .endm

   lista:        .irpc i, 1234                          # Lista de números
                        linea
                 .endr

   longlista:   .int   (.-lista)/4                      # Longitud de la lista
   resultado: .quad   0                                 # Resultado de la suma
   formato:   .ascii "resultado \t =  %18lu (uns)\n"    # Formato de la salida en la terminal del resultado
              .ascii "\t\t = 0x%18lx (hex)\n"
              .asciz "\t\t = 0x %08x %08x\n"

# Sección de código

.section .text
```

```
main: .global  main

      call trabajar                    # Suma sin signo de los números de la lista
      call imprimir                    # printf() de libC
      call acabar                      # exit() de libC
      ret

trabajar:

      mov  $lista, %rbx                # Movemos el primer valor de lista al registro %rbx
      mov  longlista, %rcx             # Movemos la longitud de la lista al registro %rcx
      call  suma                       # == suma(&lista, longlista);
      mov  %eax, resultado
      mov  %edx, resultado+4

imprimir:              # requiere libC

      mov   $formato, %rdi
      mov   resultado, %rsi
      mov   resultado, %rdx
      mov   resultado, %ecx
      mov   resultado, %r8d
      mov   $0, %eax
      call    printf                   # == printf(formato, res, res);


acabar:                # requiere libC

      movl $1, %eax
      xor   %ebx, %ebx
      int    $0x80                     # En caso de que %eax valga 1, terminamos la
                                       # ejecución y retornamos %ebx

# EDX:EAX

suma:

      mov  $0, %eax                    # Registro de la suma (lo inicializamos a 0)
      mov  $0, %edx                    # Registro del acarreo (lo inicializamos a 0)
      mov  $0, %rsi                    # Registro del índice iterador del bucle (lo
                                       # inicializamos a 0)

bucle:

      add (%rbx, %rsi, 4), %eax        # Acumulamos las sumas de cada elemento de lista
      adc  $0, %edx                    # Sumamos el acarreo
      inc   %rsi                       # Incrementamos el índice del bucle
      cmp  %rsi, %rcx                  # Comparamos si el índice actual de la lista es igual
                                       # a la longitud de la lista
      jne   bucle                      # Si no es igual, saltamos al bucle de nuevo

      ret
```

Finalmente, este es el resultado que devuelve el programa tras ejecutarlo en la terminal de Linux:

```
juanma@juanma-VirtualBox:~/Escritorio$ for i in $(seq 1 8); do
> rm suma_5_2_JMRG
> gcc -x assembler-with-cpp -D TEST=$i -no-pie suma_5_2_JMRG.s -o suma_5_2_JMRG
> printf "__TEST%02d__%35s\n" $i "" | tr " " "-"; ./suma_5_2_JMRG
> done
__TEST01__-----------------------------------
resultado       =              16 (sgn)
                = 0x           10 (hex)
                = 0x 00000010 00000010
__TEST02__-----------------------------------
resultado       =      4294967280 (sgn)
                = 0x     fffffff0 (hex)
                = 0x fffffff0 fffffff0
__TEST03__-----------------------------------
resultado       =      4294967296 (sgn)
                = 0x    100000000 (hex)
                = 0x 00000000 00000000
__TEST04__-----------------------------------
resultado       =     68719476720 (sgn)
                = 0x    ffffffff0 (hex)
                = 0x fffffff0 fffffff0
__TEST05__-----------------------------------
resultado       =     68719476720 (sgn)
                = 0x    ffffffff0 (hex)
                = 0x fffffff0 fffffff0
__TEST06__-----------------------------------
resultado       =      3200000000 (sgn)
                = 0x     bebc2000 (hex)
                = 0x bebc2000 bebc2000
__TEST07__-----------------------------------
resultado       =      4800000000 (sgn)
                = 0x    11e1a3000 (hex)
                = 0x 1e1a3000 1e1a3000
suma_5_2_JMRG.s: Mensajes del ensamblador:
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
suma_5_2_JMRG.s:33: Aviso: el valor 0x12a05f200 se truncó a 0x2a05f200
__TEST08__-----------------------------------
resultado       =     11280523264 (sgn)
                = 0x    2a05f2000 (hex)
                = 0x a05f2000 a05f2000
```

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

A continuación se puede observar el código en ensamblador para realizar la suma pedida:

```
# Sección de datos

.section .data

   #ifndef  TEST
   #define TEST 20
   #endif
      .macro linea

   #if     TEST==1
                  .int -1, -1, -1, -1
   #elif   TEST==2
                  .int 0x04000000, 0x04000000, 0x04000000, 0x04000000
   #elif   TEST==3
                  .int 0x08000000, 0x08000000, 0x08000000, 0x08000000
   #elif   TEST==4
                  .int 0x10000000, 0x10000000, 0x10000000, 0x10000000
   #elif   TEST==5
                  .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
   #elif   TEST==6
                  .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
   #elif   TEST==7
                  .int 0xf0000000, 0xf0000000, 0xf0000000, 0xf0000000
   #elif   TEST==8
                  .int 0xf8000000, 0xf8000000, 0xf8000000, 0xf8000000
   #elif   TEST==9
                  .int 0xf7ffffff, 0xf7ffffff, 0xf7ffffff, 0xf7ffffff
   #elif   TEST==10
                  .int 100000000, 100000000, 100000000, 100000000
   #elif   TEST==11
                  .int 200000000, 200000000, 200000000, 200000000
   #elif   TEST==12
                  .int 300000000, 300000000, 300000000, 300000000
   #elif   TEST==13
                  .int 2000000000, 2000000000, 2000000000, 2000000000
   #elif   TEST==14
                  .int 3000000000, 3000000000, 3000000000, 3000000000
   #elif   TEST==15
                  .int -100000000, -100000000, -100000000, -100000000
   #elif   TEST==16
                  .int -200000000, -200000000, -200000000, -200000000
   #elif   TEST==17
                  .int -300000000, -300000000, -300000000, -300000000
   #elif   TEST==18
                  .int -2000000000, -2000000000, -2000000000, -2000000000
   #elif   TEST==19
                  .int -3000000000, -3000000000, -3000000000, -3000000000
   #else
                  .error "DEFINIR TEST ENTRE 1...19"
```

```
#endif
        .endm

lista:          .irpc i, 1234                       # Lista de números
                    linea
                .endr

longlista:   .int   (.-lista)/4                     # Longitud de la lista
resultado: .quad   0                                # Resultado de la suma
formato:  .ascii "resultado \t =  %18lu (uns)\n"    # Formato de la salida en la terminal del resultado
              .ascii "\t\t = 0x%18lx (hex)\n"
              .asciz "\t\t = 0x %08x %08x\n"

# Sección de código

.section .text

    main: .global  main

          call trabajar                             # Suma sin signo de los números de la lista
          call imprimir                             # printf() de libC
          call acabar                               # exit() de libC
          ret

    trabajar:

          mov  $lista, %rdi                         # Movemos el primer valor de lista al registro %rdi
          mov  longlista, %rcx                      # Movemos la longitud de la lista al registro %rcx
          call   suma                               # == suma(&lista, longlista);
          mov  %eax, resultado
          mov  %edx, resultado+4

    imprimir:                   # requiere libC

          mov      $formato, %rdi
          mov     resultado, %rsi
          mov     %edx, %ecx                        # Cambiamos de posición para que no se pierda el
                                                    # valor de %edx
          mov      resultado, %rdx
          movsx   %eax, %r8
          mov      %eax, %eax
          call      printf                          # == printf(formato, res, res);


    acabar:                     # requiere libC

          movl $1, %eax
          xor    %ebx, %ebx
          int     $0x80                             # En caso de que %eax valga 1, terminamos la
                                                    # ejecución y retornamos %ebx
```

```
# EDX:EAX

suma:

        mov  $0, %ebx                   # Registro de la suma (lo inicializamos a 0)
        mov  $0, %ecx                   # Registro de la extensión de signo (lo inicializamos
                                        # a 0)
        mov  $0, %ebp                   # Registro del índice iterador del bucle (lo
                                        # inicializamos a 0)

bucle:

        mov (%rdi, %rbp, 4), %eax       # eax = lista[i]
        cdq   $0, %edx                  # Extensión de signo de edx:eax

        add    %eax , %ebx
        add    %edx , %ecx

        inc  %ebp, %rcx                 # Incrementamos el índice del bucle
        cmp  %ebp, %esi                 # Comparamos si el índice actual de la lista es igual
                                        # a la longitud de la lista
        jne   bucle                     # Si no es igual, saltamos al bucle de nuevo

        mov  %ecx, %edx                 # Una vez acabado el bucle, retornamos %eax y
                                        # %edx
        mov  %ebx, %eax

        ret
```

Finalmente, este es el resultado que devuelve el programa tras ejecutarlo en la terminal de Linux:

```
juanma@juanma-VirtualBox:~/Escritorio$ for i in $(seq 1 19); do
> rm suma_5_3_JMRG
> gcc -x assembler-with-cpp -D TEST=$i -no-pie suma_5_3_JMRG.s -o suma_5_3_JMRG
> printf "__TEST%02d__%35s\n" $i "" | tr " " "-"; ./suma_5_3_JMRG
> done
rm: no se puede borrar 'suma_5_3_JMRG': 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 fffffffa0a1f000 (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

```
suma_5_3_JMRG.s: Mensajes del ensamblador:
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
suma_5_3_JMRG.s:55: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
__TEST19__----------------------------------
resultado        =       20719476736 (sgn)
                 = 0x      4d2fa2000 (hex)
                 = 0x 00000004 d2fa2000
```

4. Media y Resto de N enteros *con* signo de 32 bits calculada usando registros de 32 bits (N ≈ 16)

A continuación se puede observar el código en ensamblador para realizar la suma pedida:

```
# Sección de datos

.section .data

   #ifndef  TEST
   #define TEST 20
   #endif

   # Doble macro: lista definida por una lista0 y 3 líneas normales
   # En la mayoría de ejemplos, linea0 = línea => lista tiene 4 líneas normales

      .macro linea

   #if     TEST==1
                   .int 1, 2, 1, 2
   #elif   TEST==2
                   .int -1, -2, -1, -2
   #elif   TEST==3
                   .int 0x7fffffff, 0x7fffffff, 0x7fffffff, 0x7fffffff
   #elif   TEST==4
                   .int 0x80000000, 0x80000000, 0x80000000, 0x80000000
   #elif   TEST==5
                   .int 0xffffffff, 0xffffffff, 0xffffffff, 0xffffffff
   #elif   TEST==6
                   .int 2000000000, 2000000000, 2000000000, 2000000000
```

```
#elif   TEST==7
                .int 3000000000, 3000000000, 3000000000, 3000000000
#elif   TEST==8
                .int -2000000000, -2000000000, -2000000000, -2000000000
#elif   TEST==9
                .int -3000000000, -3000000000, -3000000000, -3000000000
#elif   TEST>=10 && TEST<=14
                .int 1, 1, 1, 1                         # linea0 + 3lineas, casi todo a 1
#elif   TEST>=15 && TEST<=19
                .int -1, -1, -1, -1                     # linea0 + 3lineas, casi todo a -1
#else
                .error "DEFINIR TEST ENTRE 1...19"
#endif
        .endm

# En la mayoría de ejemplos, linea0 = línea => lista tiene 4 líneas normales

    .macro linea0

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

# En la mayoría de ejemplos, linea0 = línea => lista tiene 4 líneas normales

lista:  linea0                                          # Lista de números
            .irpc i, 123
                    linea
            .endr

longlista:   .int   (.-lista)/4                         # Longitud de la lista
resultado: .quad   0                                    # Resultado de la suma
formato:    .ascii "media \t =   %11d \t resto \t = %11d \n" # Formato de la salida en la terminal del
            .ascii "\t = 0x %08x \t \t = 0x %08x\n"      # resultado
```

```
# Sección de código

.section .text

    main: .global  main

          call trabajar                    # Suma sin signo de los números de la lista
          call imprimir                    # printf() de libC
          call acabar                      # exit() de libC
          ret

    trabajar:

          mov  $lista, %rbx                # Movemos el primer valor de lista al registro %rbx
          mov  longlista, %ecx             # Movemos la longitud de la lista al registro %ecx
          call   suma                      # == suma(&lista, longlista);
          mov  $16, %r8d
          idiv  %r8d                       # Realizamos la división
          mov  %eax, media                 # Cociente de la división
          mov  %edx, resto                 # Resto de la división

    imprimir:                 # requiere libC

          mov      $formato, %rdi
          mov     media, %rsi
          mov     resto, %edx
          mov     $0, %eax
          call      printf                 # == printf(formato, res, res);


    acabar:                   # requiere libC

          mov   media, %edi
          call   exit
          int    $0x80

    # EDX:EAX

    suma:

          mov %ecx,%esi                    # Guardamos la dirección de longlista en %ebp
          mov %rbx,%rdi                    # Dirección donde comienza la lista de enteros
          mov $0, %ebx                     # Registro de la suma (lo inicializamos a 0)
          mov $0, %ecx                     # Registro de la extensión de signo (lo inicializamos
                                           # a 0). Usamos estos 2 registros en lugar de %eax y
                                           # %edx porque cdq utiliza %edx y %eax
          mov $0, %rdx                     # Registro del acarreo (lo inicializamos a 0)
          mov $0, %ebp                     # Registro del índice iterador del bucle (el bucle
                                           # acaba cuando %ebp = %esi)
```

```
bucle:

        mov (%rdi, %rbp , 4), %eax          # %rdi es la dirección donde comienza la lista de
                                            # enteros
        cdq                                 # Extensión de signo de edx:eax
        add  %eax, %ebx
        adc  %edx, %ecx
        inc  %ebp

        cmp %ebp, %esi                      # Comparamos si el índice actual de la lista es igual
                                            # a la longitud de la lista
        jne  bucle                          # Si no es igual, saltamos al bucle de nuevo

        mov %ebx, %eax                      # Una vez acabado el bucle, retornamos %eax y
                                            # %edx
        mov %ecx, %edx

        ret
```

Finalmente, este es el resultado que devuelve el programa tras ejecutarlo en la terminal de Linux:

```
juanma@juanma-VirtualBox:~/Escritorio$ for i in $(seq 1 19); do
> rm media_5_4_JMRG
> gcc -x assembler-with-cpp -D TEST=$i -no-pie media_5_4_JMRG.s -o media_5_4_JMRG
> printf "__TEST%02d__%35s\n" $i "" | tr " " "-"; ./media_5_4_JMRG
> done
__TEST01__-----------------------------------
media    =         1          resto    =         8
         = 0x 00000000                 = 0x 00000010
__TEST02__-----------------------------------
media    =        -1          resto    =        -8
         = 0x ffffffff                 = 0x 00000010
__TEST03__-----------------------------------
media    =  2147483647        resto    =         0
         = 0x 00000007                 = 0x 00000010
__TEST04__-----------------------------------
media    = -2147483648        resto    =         0
         = 0x fffffff8                 = 0x 00000010
__TEST05__-----------------------------------
media    =        -1          resto    =         0
         = 0x ffffffff                 = 0x 00000010
__TEST06__-----------------------------------
media    =  2000000000        resto    =         0
         = 0x 00000007                 = 0x 00000010
__TEST07__-----------------------------------
media    = -1294967296        resto    =         0
         = 0x fffffffb                 = 0x 00000010
__TEST08__-----------------------------------
media    = -2000000000        resto    =         0
         = 0x fffffff8                 = 0x 00000010
media_5_4_JMRG.s: Mensajes del ensamblador:
media_5_4_JMRG.s:76: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:76: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:76: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:76: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
```

```
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
media_5_4_JMRG.s:79: Aviso: el valor 0xffffffff4d2fa200 se truncó a 0x4d2fa200
__TEST09__----------------------------------
media    =  1294967296    resto    =      0
         = 0x 00000004             = 0x 00000010
__TEST10__----------------------------------
media    =        1       resto    =      0
         = 0x 00000000             = 0x 00000010
__TEST11__----------------------------------
media    =        1       resto    =      1
         = 0x 00000000             = 0x 00000010
__TEST12__----------------------------------
media    =        1       resto    =      8
         = 0x 00000000             = 0x 00000010
__TEST13__----------------------------------
media    =        1       resto    =     15
         = 0x 00000000             = 0x 00000010
__TEST14__----------------------------------
media    =        2       resto    =      0
         = 0x 00000000             = 0x 00000010
__TEST15__----------------------------------
media    =       -1       resto    =      0
         = 0x ffffffff             = 0x 00000010
__TEST16__----------------------------------
media    =       -1       resto    =     -1
         = 0x ffffffff             = 0x 00000010
__TEST17__----------------------------------
media    =       -1       resto    =     -8
         = 0x ffffffff             = 0x 00000010
__TEST18__----------------------------------
media    =       -1       resto    =    -15
         = 0x ffffffff             = 0x 00000010
__TEST19__----------------------------------
media    =       -2       resto    =      0
         = 0x ffffffff             = 0x 00000010
```