```
    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:
```