```
    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
   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:           .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
  112:
  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:
```