```
    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
   51:
   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:
```