```c
#include <stdio.h>              // para printf()
#include <stdlib.h>             // para exit()
#include <sys/time.h>           // para gettimeofday(), struct timeval

int resultado = 0;

#ifndef TEST
#define TEST 5
#endif

/* ------------------------------------------------------------------------ */
        #if TEST==1
/* ------------------------------------------------------------------------ */
        #define SIZE 4
        unsigned lista[SIZE]={0x80000000, 0x00400000, 0x00000200, 0x00000001};
        #define RESULT 4
/* ------------------------------------------------------------------------ */
#elif TEST==2
/* ------------------------------------------------------------------------ */
        #define SIZE 8
        unsigned lista[SIZE]={0x7fffffff, 0xffbfffff, 0xffffdfff, 0xfffffffe,
                              0x01000023, 0x00456700, 0x8900ab00, 0x00cd00ef};
        #define RESULT 8
/* ------------------------------------------------------------------------ */
#elif TEST==3
/* ------------------------------------------------------------------------ */
        #define SIZE 8
        unsigned lista[SIZE]={0x0       , 0x01020408, 0x35906a0c, 0x70b0d0e0,
                              0xffffffff, 0x12345678, 0x9abcdef0, 0xdeadbeef};
        #define RESULT 8
/* ------------------------------------------------------------------------ */
#elif TEST==4 || TEST==0
/* ------------------------------------------------------------------------ */
        #define NBITS 20
        #define SIZE (1<<NBITS)         // tamaÃ±o suficiente para tiempo apreciable
        unsigned lista[SIZE];         // unsigned para desplazamiento derecha lÃ³gico
        #define RESULT ( NBITS * ( 1 << NBITS-1 ) )
/* ------------------------------------------------------------------------ */
#else
                #error "Definir TEST entre 0..4"
#endif
/* ------------------------------------------------------------------------ */

int popcount1(unsigned* array, size_t len)
{
        size_t i, j;
        int result = 0;
        unsigned x;

        for (i = 0; i < len; i++){ // Recorrer el vector
                x = array[i];
                for (j = 0; j < sizeof(int) * 8; j++){ // Recorremos cada entero del
 array por sus bits
                        unsigned bit = (x >> j) & 0x1; // Desplazamos los bits neces
arios a la derecha y aplicamos la mÃ¡scara
                        result += bit; // AÃ±adimos el bit al resultado
                }
        }

        return result;
}

/* ------------------------------------------------------------------------ */

int popcount2(unsigned* array, size_t len)
{
        size_t i;
        int result = 0;
```

```
    67:     unsigned x;
    68:
    69:         for (i = 0; i < len; i++){ // Recorremos el vector
    70:         x = array[i];
    71:                 while (x){    // Recorremos cada entero del array por sus bits, nos
salimos del bucle cuando sea 0
    72:                         result += x & 0x1; // AÃ±adimos al resultado el entero con l
a mÃ¡scara aplicada
    73:                         x >>= 1; // Desplazamos un bit a la derecha
    74:                 }
    75:         }
    76:
    77:         return result;
    78: }
    79:
    80: /* ---------------------------------------------------------------------- */
    81:
    82: int popcount3(unsigned* array, size_t len)
    83: {
    84:         int result = 0;
    85:         unsigned x;
    86:         size_t i;
    87:
    88:         for (i = 0; i < len; i++){ // Recorremos el vector
    89:                 x = array[i];
    90:                 asm("\n"
    91:         "ini3:                          \n\t"
    92:                         "shr %[x]       \n\t"   // Desplaza un bit a la derecha
    93:                         "adc $0, %[r] \n\t" // Sumamos el Ãºltimo bit a result
    94:                         "test %[x], %[x] \n\t"
    95:                         "jnz ini3               \n\t" // Si x no es 0, salta a ini3
    96:
    97:                         : [r]"+r" (result)
    98:                         : [x] "r" (x)           );
    99:         }
   100:         return result;
   101: }
   102:
   103: /* ---------------------------------------------------------------------- */
   104:
   105: int popcount4(unsigned* array, size_t len)
   106: {
   107:         int result = 0;
   108:         unsigned x;
   109:         size_t i;
   110:
   111:         for (i = 0; i < len; i++){
   112:                 x = array[i];
   113:                 asm("\n"
   114:                         "clc            \n\t" // Limpiamos el flag de acarreo
   115:         "ini4:                          \n\t"
   116:                         "adc $0, %[r] \n\t" // Sumamos el Ãºltimo bit a result
   117:                         "shr %[x]       \n\t" // Desplaza un bit a la derecha
   118:                         "jnz ini4       \n\t" // Si x no es cero, vuelve a ini4
   119:         "fin4:                          \n\t"
   120:                         "adc $0, %[r]   \n\t" // Si x es cero, aÃ±ade el Ãºltimo bit
 a result
   121:
   122:                         : [r] "+r" (result)
   123:                         : [x] "r" (x)           );
   124:
   125:         }
   126:         return result;
   127: }
   128:
   129: /* ---------------------------------------------------------------------- */
   130:
   131: int popcount5(unsigned* array, size_t len)
```

```
  132: {
  133:         int result = 0, val = 0;
  134:         size_t i, j;
  135:         unsigned x;
  136:
  137:         for (i = 0; i < len; i++){ // Recorremos el vector
  138:                 x = array[i];
  139:                 val = 0; // Variable local para acumular los bits
  140:                 for (j = 0; j < 8; j++){ // Recorremos cada entero
  141:                         val += x & 0x01010101; // Aplicamos la mÃ¡scara (para 32 bit
s)
  142:                         x >>= 1; // Desplazamos un bit a la derecha
  143:                 }
  144:                 val += (val >> 16); // Sumamos los bits
  145:                 val += (val >> 8);
  146:                 result += val & 0xFF;
  147:         }
  148:         return result;
  149: }
  150:
  151: /* ------------------------------------------------------------------------ */
  152:
  153: int popcount6(unsigned* array, size_t len)
  154: {
  155:         const unsigned m1 = 0x55555555;
  156:         const unsigned m2 = 0x33333333;
  157:         const unsigned m4 = 0x0f0f0f0f;
  158:         const unsigned m8 = 0x00ff00ff;
  159:         const unsigned m16 = 0x0000ffff;
  160:
  161:         int result = 0;
  162:         size_t i;
  163:         unsigned x;
  164:
  165:         for (i = 0; i < len; i++){ // Recorremos el vector
  166:                 x = array[i];
  167:
  168:                 x = (x & m1) + ((x >> 1) & m1);  // Sumamos en Ã¡rbol los bits
  169:                 x = (x & m2) + ((x >> 2) & m2);
  170:                 x = (x & m4) + ((x >> 4) & m4);
  171:                 x = (x & m8) + ((x >> 8) & m8);
  172:                 x = (x & m16) + ((x >> 16) & m16);
  173:
  174:                 result += x;
  175:         }
  176:         return result;
  177: }
  178:
  179: /* ------------------------------------------------------------------------ */
  180:
  181: int popcount7(unsigned* array, size_t len)
  182: {
  183:         size_t i;
  184:         unsigned long x1, x2;
  185:         int result = 0;
  186:
  187:         const unsigned long m1 = 0x5555555555555555;
  188:         const unsigned long m2 = 0x3333333333333333;
  189:         const unsigned long m4 = 0x0f0f0f0f0f0f0f0f;
  190:         const unsigned long m8 = 0x00ff00ff00ff00ff;
  191:         const unsigned long m16 = 0x0000ffff0000ffff;
  192:         const unsigned long m32 = 0x00000000ffffffff;
  193:
  194:         if (len & 0x3) printf("leyendo128b pero len no mÃºltiplo de 4\n");
  195:
  196:         for (i = 0; i < len; i +=4) // Recorremos el vector
  197:         {
  198:                 x1 = *(unsigned long*) &array[i];
```

```
  199:                 x2 = *(unsigned long*) &array[i+2];
  200:
  201:                 x1 = (x1 & m1) + ((x1 >> 1) & m1); // Sumas en Ã¡rbol
  202:                 x1 = (x1 & m2) + ((x1 >> 2) & m2);
  203:                 x1 = (x1 & m4) + ((x1 >> 4) & m4);
  204:                 x1 = (x1 & m8) + ((x1 >> 8) & m8);
  205:                 x1 = (x1 & m16) + ((x1 >> 16) & m16);
  206:                 x1 = (x1 & m32) + ((x1 >> 32) & m32);
  207:
  208:                 x2 = (x2 & m1) + ((x2 >> 1) & m1);
  209:                 x2 = (x2 & m2) + ((x2 >> 2) & m2);
  210:                 x2 = (x2 & m4) + ((x2 >> 4) & m4);
  211:                 x2 = (x2 & m8) + ((x2 >> 8) & m8);
  212:                 x2 = (x2 & m16) + ((x2 >> 16) & m16);
  213:                 x2 = (x2 & m32) + ((x2 >> 32) & m32);
  214:
  215:                 result += x1+x2;
  216:         }
  217:         return result;
  218: }
  219:
  220: /* ------------------------------------------------------------------------ */
  221: int popcount8(unsigned *array, size_t len){
  222:         size_t i;
  223:    int val, result=0;
  224:    int SSE_mask[] = {0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f};
  225:    int SSE_LUTb[] = {0x02010100, 0x03020201, 0x03020201, 0x04030302};
  226:    //                 3 2 1 0     7 6 5 4    11 10 9 8   15 14 13 12
  227:
  228:    if (len & 0x3) printf("leyendo 128b pero len no mÃºltiplo de 4\n");
  229:
  230:    for (i=0; i<len; i+=4){
  231:         asm("movdqu %[x], %%xmm0          \n\t"
  232:         "movdqa %%xmm0, %%xmm1            \n\t"  //  x: two copies xmm0-1
  233:         "movdqu %[m], %%xmm6              \n\t"  // mask: xmm6
  234:         "psrlw $4 , %%xmm1                \n\t"
  235:         "pand %%xmm6, %%xmm0              \n\t"  //; xmm0 â\200\223 lower nibbles
  236:         "pand %%xmm6, %%xmm1              \n\t"  //; xmm1 â\200\223 higher nibbles
  237:
  238:         "movdqu %[l], %%xmm2              \n\t"  //; since instruction pshufb modifies
 LUT
  239:         "movdqa %%xmm2, %%xmm3            \n\t"  //; we need 2 copies
  240:         "pshufb %%xmm0, %%xmm2            \n\t"  //; xmm2 = vector of popcount lower n
ibbles
  241:         "pshufb %%xmm1, %%xmm3            \n\t"  //; xmm3 = vector of popcount upper n
ibbles
  242:
  243:         "paddb %%xmm2, %%xmm3             \n\t"  //; xmm3 - vector of popcount for byt
es
  244:         "pxor %%xmm0, %%xmm0              \n\t"  //; xmm0 = 0,0,0,0
  245:         "psadbw %%xmm0, %%xmm3            \n\t"  //; xmm3 = [pcnt bytes0..7|pcnt bytes
8..15]
  246:         "movhlps %%xmm3, %%xmm0           \n\t"  //; xmm0 = [     0         |pcnt bytes
0..7 ]
  247:         "paddd %%xmm3, %%xmm0             \n\t"  //; xmm0 = [  not needed   |pcnt bytes
0..15]
  248:         "movd %%xmm0, %[val]"
  249:
  250:         : [val]"=r" (val)
  251:         : [x] "m" (array[i]),
  252:         [m] "m" (SSE_mask[0]),
  253:         [l] "m" (SSE_LUTb[0])
  254:         );
  255:
  256:         result += val;
  257:     }
  258:
  259:     return result;
```

```
260:
261: }
262:
263: /* ------------------------------------------------------------------------ */
264:
265: int popcount9(unsigned* array, size_t len)
266: {
267:         size_t i;
268:         unsigned x;
269:         int val, result = 0;
270:
271:         for (i = 0; i < len; i++) // Recorremos el vector
272:         {
273:                 x = array[i];
274:                 asm("popcnt %[x],%[val]" // Guardamos el popcount de x en val
275:
276:                         :[val] "=r" (val)
277:                         :  [x]  "r" (x)
278:                 );
279:                 result += val;
280:         }
281:         return result;
282: }
283:
284: /* ------------------------------------------------------------------------ */
285:
286: int popcount10(unsigned* array, size_t len)
287: {
288:         size_t i;
289:         unsigned long x1, x2;
290:         long val = 0;
291:         int result = 0;
292:
293:         if (len & 0x3) printf("leyendo 128b pero len no mÃºltiplo de 4\n");
294:                 for (i=0; i<len; i+=4){ // Recorremos el vector con un paso de 4
295:                         x1 = *(unsigned long*) &array[i];
296:                         x2 = *(unsigned long*) &array[i+2];
297:                         asm("popcnt %[x1], %[val] \n\t" // Realizamos el popcount de
 x1 y lo guardamos val
298:                                 "popcnt %[x2], %[x1] \n\t"  // Realizamos el popcoun
t de x2 y lo guardamos en x1
299:                                 "add    %[x1], %[val] \n\t" // Sumamos x1 y val y gu
ardamos el resultado en val
300:                         : [val]"=&r" (val)
301:                         :  [x1]  "r" (x1),
302:                            [x2]  "r" (x2)
303:                         );
304:                         result += val;
305:                 }
306:                 return result;
307: }
308:
309: /* ------------------------------------------------------------------------ */
310:
311: void crono(int (*func)(), char* msg){
312:     struct timeval tv1,tv2;                     // gettimeofday() secs-usecs
313:     long           tv_usecs;                    // y sus cuentas
314:
315:     gettimeofday(&tv1,NULL);
316:     resultado = func(lista, SIZE);
317:     gettimeofday(&tv2,NULL);
318:
319:     tv_usecs=(tv2.tv_sec -tv1.tv_sec )*1E6+
320:              (tv2.tv_usec-tv1.tv_usec);
321: #if TEST==0
322:         printf( "%ld" "\n",      tv_usecs);
323: #else
324:     printf("resultado = %d\t", resultado);
```

```
325:     printf("%s:%9ld us\n", msg, tv_usecs);
326: #endif
327: }
328:
329: int main()
330: {
331:         #if TEST==0 || TEST==4
332:     size_t i;                                   // inicializar array
333:     for (i=0; i<SIZE; i++)                      // se queda en cache
334:          lista[i]=i;
335:         #endif
336:
337:     crono(popcount1 , "popcount1 (lenguaje C -       for)");
338:         crono(popcount2 , "popcount2 (lenguaje C -     while)");
339:         crono(popcount3 , "popcount3 (leng.ASM-body while 4i)");
340:         crono(popcount4 , "popcount4 (leng.ASM-body while 3i)");
341:         crono(popcount5 , "popcount5 (CS:APP2e 3.49-group 8b)");
342:         crono(popcount6 , "popcount6 (Wikipedia- naive - 32b)");
343:         crono(popcount7 , "popcount7 (Wikipedia- naive -128b)");
344:         crono(popcount8 , "popcount8 (asm SSE3 - pshufb 128b)");
345:         crono(popcount9 , "popcount9 (asm SSE4- popcount 32b)");
346:         crono(popcount10, "popcount10(asm SSE4- popcount128b)");
347:
348:         #if TEST != 0
349:     printf("calculado %d\n", RESULT);
350:         #endif
351:
352:     exit(0);
353: }
354:
```