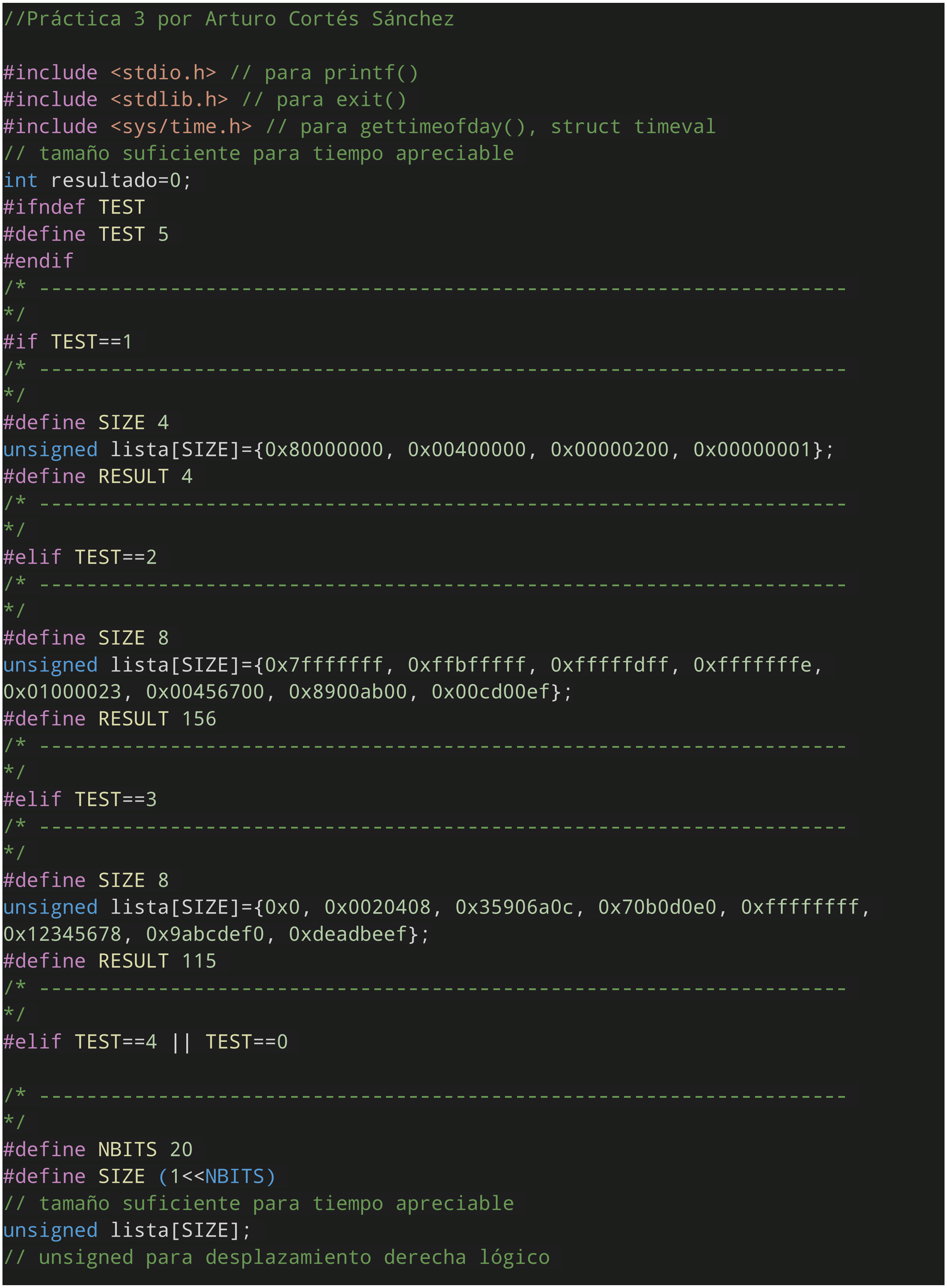
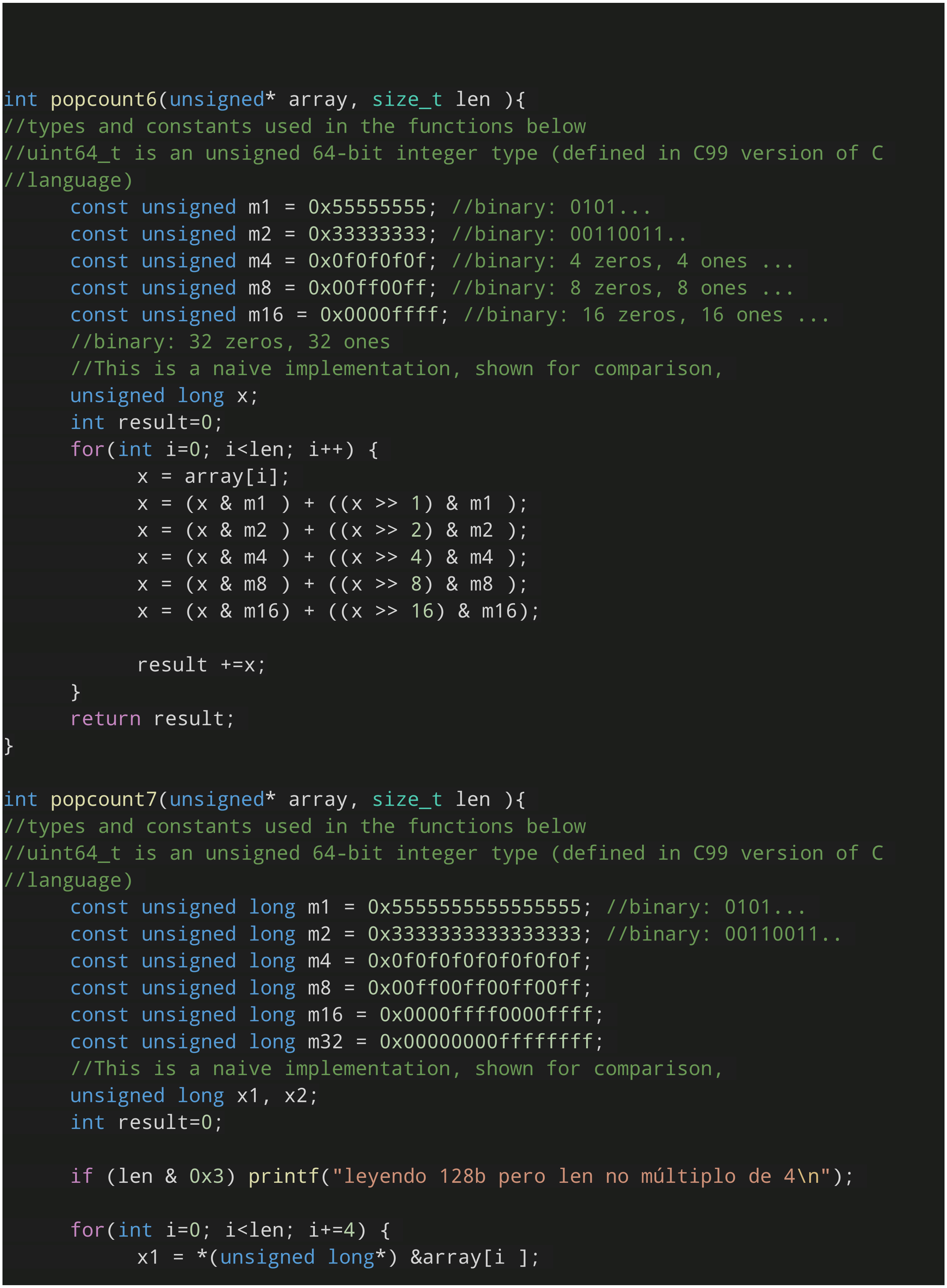
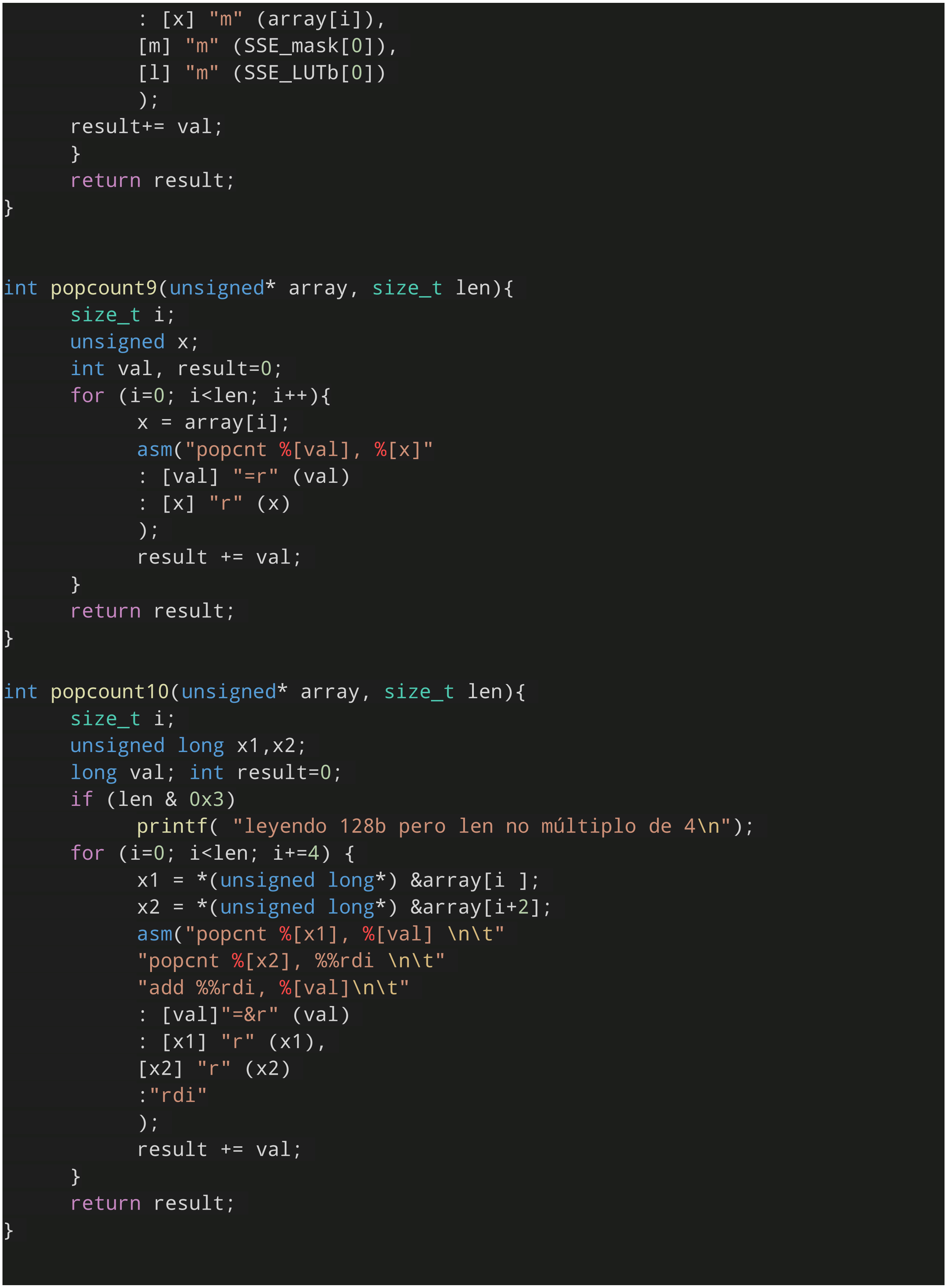


```c
//Práctica 3 por Arturo Cortés Sánchez

#include <stdio.h> // para printf()
#include <stdlib.h> // para exit()
#include <sys/time.h> // para gettimeofday(), struct timeval
// tamaño suficiente para tiempo apreciable
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, 0xfffffdff, 0xfffffffe,
0x01000023, 0x00456700, 0x8900ab00, 0x00cd00ef};
#define RESULT 156
/* --------------------------------------------------------------------
*/
#elif TEST==3
/* --------------------------------------------------------------------
*/
#define SIZE 8
unsigned lista[SIZE]={0x0, 0x0020408, 0x35906a0c, 0x70b0d0e0, 0xffffffff,
0x12345678, 0x9abcdef0, 0xdeadbeef};
#define RESULT 115
/* --------------------------------------------------------------------
*/
#elif TEST==4 || TEST==0

/* --------------------------------------------------------------------
*/
#define NBITS 20
#define SIZE (1<<NBITS)
// tamaño suficiente para tiempo apreciable
unsigned lista[SIZE];
// unsigned para desplazamiento derecha lógico
```

```
#define RESULT ( NBITS * ( 1 << NBITS-1 ) )
// pistas para deducir fórmula
/* --------------------------------------------------------------------
*/
#else
#error "Definir TEST entre 0..4"
#endif
/* --------------------------------------------------------------------
*/

int popcount1(unsigned* array, size_t len ) {
      #define WSIZE 8*sizeof(int)
      size_t i;
      int result = 0;

      for (int j=0; j<len;j++){
            unsigned x = array[j];
            for (i = 0; i < WSIZE; i++) {
                  result += x & 0x1;
                  x >>= 1;
            }
      }
      return result;

}

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

      for (int i=0; i<len;i++){
            unsigned x = array[i];
            while (x){
                  result += x & 0x1;
                  x >>= 1;
            }
      }
      return result;
}

int popcount3(unsigned* array, size_t len ){
      long result=0;
      for(int i=0; i<len; i++) {
            int x = array[i];
            asm("\n"
            "ini3: \n\t" // seguir mientras que x!=0
            "shr %[x] \n\t" // LSB en CF
            "adc $0x0, %[r] \n\t"
            "test %[x], %[x] \n\t"
            "jnz ini3 \n\t"
            : [r]"+r" (result) // e/salida: añadir a lo acumulado por el
momento
            : [x] "r" (x)
            ); // entrada: valor elemento
      }
      return result;
}

int popcount4(unsigned* array, size_t len ){
      long result=0;
      for(int i=0; i<len; i++) {
            unsigned x = array[i];
            asm("\n"
            "clc \n\t" // CLC para poder empezar por ADC
            "ini4: \n\t"
            "adc $0, %[r] \n\t"
            "shr %[x] \n\t" // LSB en CF
            "jnz ini4 \n\t"
            "adc $0, %[r] \n\t"

            : [r]"+r" (result) // e/salida: añadir a lo acumulado por el
momento
            : [x] "r" (x)
            ); // entrada: valor elemento
      }
      return result;
}

int popcount5(unsigned* array, size_t len ){
      long result=0;
      for(int i=0; i<len; i++) {
            unsigned x = array[i];
            unsigned long val = 0;
            for (int j = 0; j < 8; j++) {
                  val += x & 0x0101010101010101L;
                  x >>= 1;
            }
            val += (val >> 32);
            val += (val >> 16);
            val += (val >> 8);
            result += val & 0xFF;
      }
      return result;
}
```



```
int popcount6(unsigned* array, size_t len ){
//types and constants used in the functions below
//uint64_t is an unsigned 64-bit integer type (defined in C99 version of C
//language)
      const unsigned m1 = 0x55555555; //binary: 0101...
      const unsigned m2 = 0x33333333; //binary: 00110011..
      const unsigned m4 = 0x0f0f0f0f; //binary: 4 zeros, 4 ones ...
      const unsigned m8 = 0x00ff00ff; //binary: 8 zeros, 8 ones ...
      const unsigned m16 = 0x0000ffff; //binary: 16 zeros, 16 ones ...
      //binary: 32 zeros, 32 ones
      //This is a naive implementation, shown for comparison,
      unsigned long x;
      int result=0;
      for(int i=0; i<len; i++) {
            x = array[i];
            x = (x & m1 ) + ((x >> 1) & m1 );
            x = (x & m2 ) + ((x >> 2) & m2 );
            x = (x & m4 ) + ((x >> 4) & m4 );
            x = (x & m8 ) + ((x >> 8) & m8 );
            x = (x & m16) + ((x >> 16) & m16);

            result +=x;
      }
      return result;
}

int popcount7(unsigned* array, size_t len ){
//types and constants used in the functions below
//uint64_t is an unsigned 64-bit integer type (defined in C99 version of C
//language)
      const unsigned long m1 = 0x5555555555555555; //binary: 0101...
      const unsigned long m2 = 0x3333333333333333; //binary: 00110011..
      const unsigned long m4 = 0x0f0f0f0f0f0f0f0f;
      const unsigned long m8 = 0x00ff00ff00ff00ff;
      const unsigned long m16 = 0x0000ffff0000ffff;
      const unsigned long m32 = 0x00000000ffffffff;
      //This is a naive implementation, shown for comparison,
      unsigned long x1, x2;
      int result=0;

      if (len & 0x3) printf("leyendo 128b pero len no múltiplo de 4\n");

      for(int i=0; i<len; i+=4) {
            x1 = *(unsigned long*) &array[i ];
            x2 = *(unsigned long*) &array[i+2];

            x1 = (x1 & m1 ) + ((x1 >> 1) & m1 );
            x1 = (x1 & m2 ) + ((x1 >> 2) & m2 );
            x1 = (x1 & m4 ) + ((x1 >> 4) & m4 );
            x1 = (x1 & m8 ) + ((x1 >> 8) & m8 );
            x1 = (x1 & m16 ) + ((x1 >> 16) & m16 );
            x1 = (x1 & m32 ) + ((x1 >> 32) & m32 );

            x2 = (x2 & m1 ) + ((x2 >> 1) & m1 );
            x2 = (x2 & m2 ) + ((x2 >> 2) & m2 );
            x2 = (x2 & m4 ) + ((x2 >> 4) & m4 );
            x2 = (x2 & m8 ) + ((x2 >> 8) & m8 );
            x2 = (x2 & m16 ) + ((x2 >> 16) & m16 );
            x2 = (x2 & m32 ) + ((x2 >> 32) & m32 );

            result +=x1+x2;
      }
      return result;
}

int popcount8(unsigned* array, size_t len){
      size_t i;
      int val, result=0;
      int SSE_mask[] = {0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f};
      int SSE_LUTb[] = {0x02010100, 0x03020201, 0x03020201, 0x04030302};
      // 3 2 1 0 7 6 5 4 1110 9 8 15141312
      if (len & 0x3) printf("leyendo 128b pero len no múltiplo de 4\n");
      for (i=0; i<len; i+=4) {
            asm(
            "movdqu %[x], %%xmm0 \n\t"
            "movdqa %%xmm0, %%xmm1 \n\t" // dos copias de x)
            "movdqu %[m], %%xmm6 \n\t" // mÃ¡scara
            "psrlw $4 , %%xmm1 \n\t"
            "pand %%xmm6, %%xmm0 \n\t" //; xmm0 â€“ nibbles inferiores
            "pand %%xmm6, %%xmm1 \n\t" //; xmm1 â€“ nibbles superiores
            "movdqu %[l], %%xmm2 \n\t" //; ...como pshufb sobrescribe LUT
            "movdqa %%xmm2, %%xmm3 \n\t" //; ...queremos 2 copias
            "pshufb %%xmm0, %%xmm2 \n\t" //; xmm2 = vector popcount inferiores
            "pshufb %%xmm1, %%xmm3 \n\t" //; xmm3 = vector popcount superiores
            "paddb %%xmm2, %%xmm3 \n\t" //; xmm3 -vector popcount bytes
            "pxor %%xmm0, %%xmm0 \n\t" //; xmm0 =0,0,0,0
            "psadbw %%xmm0, %%xmm3 \n\t"
            "movhlps %%xmm3, %%xmm0 \n\t" //; xmm0 = [ 0 |pcnt bytes0..7 ]
            "paddd %%xmm3, %%xmm0 \n\t" //; xmm0 = [ no usado |pcnt bytes0..15]
            "movd %%xmm0, %[val] \n\t"
            : [val] "=r" (val)
```



```
            : [x] "m" (array[i]),
            [m] "m" (SSE_mask[0]),
            [l] "m" (SSE_LUTb[0])
            );
      result+= val;
      }
      return result;
}


int popcount9(unsigned* array, size_t len){
      size_t i;
      unsigned x;
      int val, result=0;
      for (i=0; i<len; i++){
            x = array[i];
            asm("popcnt %[val], %[x]"
            : [val] "=r" (val)
            : [x] "r" (x)
            );
            result += val;
      }
      return result;
}

int popcount10(unsigned* array, size_t len){
      size_t i;
      unsigned long x1,x2;
      long val; int result=0;
      if (len & 0x3)
            printf( "leyendo 128b pero len no múltiplo de 4\n");
      for (i=0; i<len; i+=4) {
            x1 = *(unsigned long*) &array[i ];
            x2 = *(unsigned long*) &array[i+2];
            asm("popcnt %[x1], %[val] \n\t"
            "popcnt %[x2], %%rdi \n\t"
            "add %%rdi, %[val]\n\t"
            : [val]"=&r" (val)
            : [x1] "r" (x1),
            [x2] "r" (x2)
            :"rdi"
            );
            result += val;
      }
      return result;
}
```

```c
void crono(int (*func)(), char* msg){
	struct timeval tv1,tv2; // gettimeofday() secs-usecs
	long tv_usecs; // y sus cuentas

	gettimeofday(&tv1,NULL);
	resultado = func(lista, SIZE);
	gettimeofday(&tv2,NULL);

	tv_usecs=(tv2.tv_sec -tv1.tv_sec )*1E6+
	(tv2.tv_usec-tv1.tv_usec);
#if TEST==0
	printf("%ld" "\n", tv_usecs);
#else
	printf("resultado = %d\t", resultado);
	printf("%s:%9ld us\n", msg, tv_usecs);
#endif
}

int main() {
#if TEST==0 || TEST==4
	size_t i; // inicializar array
	for (i=0; i<SIZE; i++)
		lista[i]=i;
#endif
	crono(popcount1 , "popcount1 (lenguaje C - for)");
	crono(popcount2 , "popcount2 (lenguaje C - while)");
	crono(popcount3 , "popcount3 (leng.ASM-body while 4i)");
	crono(popcount4 , "popcount4 (leng.ASM-body while 3i)");
	crono(popcount5 , "popcount5 (CS:APP2e 3.49-group 8b)");
	crono(popcount6 , "popcount6 (Wikipedia- naive - 32b)");
	crono(popcount7 , "popcount7 (Wikipedia- naive-128b)");
	crono(popcount8 , "popcount8 (asm SSE3 - pshufb 128b)");
	crono(popcount9 , "popcount9 (asm SSE4- popcount 32b)");
	crono(popcount10, "popcount10(asm SSE4- popcount128b)");
#if TEST != 0
	printf("calculado = %d\n", RESULT);
#endif
	exit(0);
}
```

lscpu:	CPU(s): 4 Nombre del modelo: Intel(R) Core(TM) i3 CPU M 380 @ 2.53GHz Virtualización: VT-x Caché L3: 3072K

Prácticas de Estructura de Computadores

por Javier Fernández y Mancia Anguita

POPCOUNT:

```
 for i in 0 g 1 2; do
    printf "__OPTIM%1c__%48s\n" $i "" | tr " " "="
    rm  popcount
    gcc popcount.c -o popcount -O$i -D TEST=0
  for j in $(seq 0 10); do
    echo $j; ./popcount
  done | pr -11 -l 22 -w 80
 done
```

ignorar medición 0, repetir columna si alguna medición se sale demasiado de la media

Optimización -O0	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	140395	117119	132501	134347	117038	145772	151048	141194	149981	135223	151029	137525
popcount2 (lenguaje C - while):	65988	65941	67626	65849	65939	66031	65849	66014	66030	65971	65812	66106
popcount3 (leng.ASM-body while 4i):	21369	21301	21857	21360	21325	21242	21306	21240	21307	21288	21248	21347
popcount4 (leng.ASM-body while 3i):	22000	22031	22619	22112	21975	21980	22056	22136	21976	22021	21991	22090
popcount5 (CS:APP2e 3.49-group 8b):	37069	37237	37993	37085	37088	37086	37076	37094	37138	37098	37096	37199
popcount6 (Wikipedia- naive - 32b):	17591	17579	17919	17596	17522	17621	17666	17548	17586	17567	17534	17614
popcount7 (Wikipedia- naive -128b):	9115	9144	9414	9246	9105	9118	9117	9119	9160	9137	9123	9168
popcount8 (asm SSE3 - pshufb 128b):	1645	1675	1686	1673	1663	1670	1670	1663	1664	1706	1679	1675
popcount9 (asm SSE4- popcount 32b):	5572	5548	5891	5616	5554	5552	5560	5555	5553	5582	5553	5596
popcount10(asm SSE4- popcount128b):	2047	2041	2147	2042	2087	2045	2051	2045	2049	2055	2055	2062

Optimización -Og	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	57345	58271	52552	53239	56343	52862	83383	90431	52623	52802	59686	61219
popcount2 (lenguaje C - while):	34641	35851	35621	36039	35914	35600	34410	36399	35706	35801	36106	35745
popcount3 (leng.ASM-body while 4i):	18423	20731	18644	20795	18678	18484	18746	20934	20999	21641	22959	20261
popcount4 (leng.ASM-body while 3i):	26826	27077	27138	27072	27008	27062	26411	27057	27004	27004	27040	26987
popcount5 (CS:APP2e 3.49-group 8b):	13418	13534	13473	13410	13386	13361	13069	13522	13376	13371	13340	13384
popcount6 (Wikipedia- naive - 32b):	6117	6186	6161	6169	6181	6212	6014	6190	6210	6214	6162	6170
popcount7 (Wikipedia- naive -128b):	3656	3580	3663	3570	3600	3570	3482	3511	3524	3505	3547	3555
popcount8 (asm SSE3 - pshufb 128b):	959	1131	1033	1053	1036	1001	967	1004	992	1046	1006	1027
popcount9 (asm SSE4- popcount 32b):	1427	1427	1473	1385	1438	1371	1374	1443	1418	1375	1441	1415
popcount10(asm SSE4- popcount128b):	624	662	740	629	725	683	662	629	712	616	671	673

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	68791	43726	43744	69594	42874	79335	49495	50085	42875	79773	42813	54431
popcount2 (lenguaje C - while):	19987	19229	19071	18951	19047	20250	19072	19120	18791	20538	19047	19312
popcount3 (leng.ASM-body while 4i):	27172	27222	27205	26649	26630	26641	26570	26622	26757	26683	26580	26756
popcount4 (leng.ASM-body while 3i):	19509	19468	19421	19020	19032	19069	19046	19025	19025	19031	19071	19121
popcount5 (CS:APP2e 3.49-group 8b):	12246	12220	12253	11998	12044	11950	11942	11976	11944	11994	11938	12026
popcount6 (Wikipedia- naive - 32b):	5557	5594	5577	5463	5459	5473	5463	5495	5482	5467	5521	5499
popcount7 (Wikipedia- naive -128b):	3313	3325	3316	3203	3231	3234	3226	3228	3227	3230	3236	3246
popcount8 (asm SSE3 - pshufb 128b):	965	1024	1006	991	993	958	1013	951	995	999	965	990
popcount9 (asm SSE4- popcount 32b):	1089	1055	1050	1051	1056	1040	1108	1048	1051	1056	1086	1060
popcount10(asm SSE4- popcount128b):	690	676	600	599	607	641	612	638	603	615	618	621

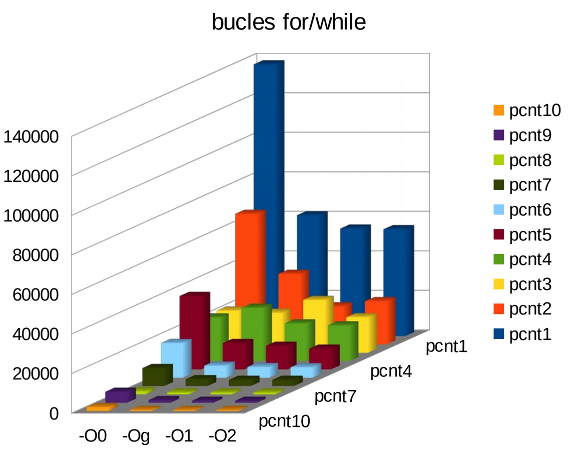
Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	51709	63862	37211	54147	64815	46626	37304	57091	64851	50205	65005	54112
popcount2 (lenguaje C - while):	21489	23973	18868	23339	22860	20253	18834	22716	24199	20725	23324	21909
popcount3 (leng.ASM-body while 4i):	20067	18850	18510	16498	18671	18558	18253	18317	16767	16998	18974	18040
popcount4 (leng.ASM-body while 3i):	18605	16852	18636	16813	18576	16912	18581	18589	18635	18605	18949	18115
popcount5 (CS:APP2e 3.49-group 8b):	10544	10574	10529	10526	10578	10529	10584	10579	10533	10543	10817	10579
popcount6 (Wikipedia- naive - 32b):	5327	5329	5319	5380	5329	5343	5331	5331	5334	5337	5705	5374
popcount7 (Wikipedia- naive -128b):	3197	3201	3194	3203	3197	3205	3197	3196	3198	3194	3255	3204
popcount8 (asm SSE3 - pshufb 128b):	1000	1005	1034	1024	996	996	993	990	999	1004	1087	1013
popcount9 (asm SSE4- popcount 32b):	1048	1078	1042	1072	1034	1038	1034	1072	1036	1038	1097	1054
popcount10(asm SSE4- popcount128b):	607	615	607	669	640	612	599	617	635	640	622	626

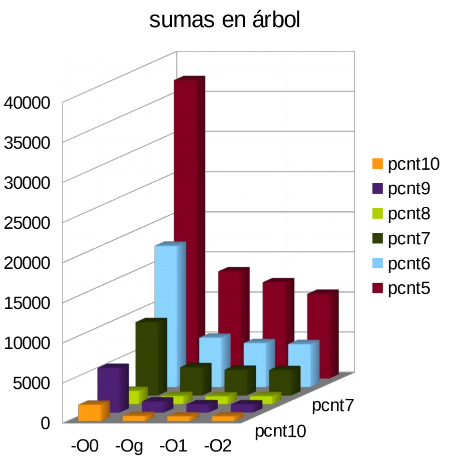
POPCOUNT:	-O0	-Og	-O1	-O2
pcnt1	137525	61219	**54431**	54112
pcnt2	66106	**35745**	19312	21909
pcnt3	21347	20261	**26756**	18040
pcnt4	22090	26987	**19121**	18115
pcnt5	37199	13384	12026	**10579**
pcnt6	17614	6170	5499	**5374**
pcnt7	9168	3555	3246	**3204**
pcnt8	1675	1027	990	**1013**
pcnt9	5596	1415	1060	**1054**
pcnt10	2062	**673**	**621**	**626**

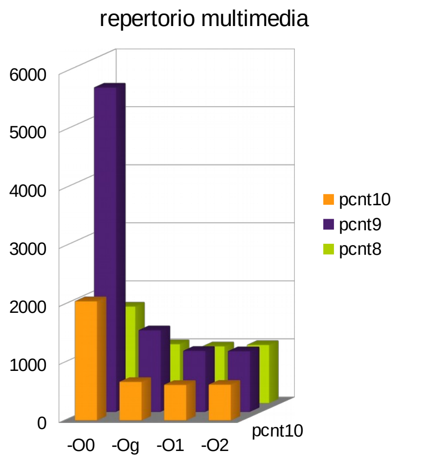
Ganancias:	-O0	-Og	-O1	-O2
pcnt1			1,00	
pcnt2		**1,52**		
pcnt3			2,03	
pcnt4			2,85	
pcnt5				**5,15**
pcnt6				**10,13**
pcnt7				**16,99**
pcnt8				**53,74**
pcnt9				51,64
pcnt10		**80,89**	87,67	87,01

Comentario

comparado con el for más rápido
el while es un 70% más rápido
ASM se queda en un 35%
o en un 43%
sumar en grupos 8b sale 3x más rápido
sumar en árbol 6x
lectura 128b sube a 10x
SSSE3 sube a 35x más rápido
SSE4 sólo 30x por leer 32b
SSE4 128b sube a 44x


bucles for/while


sumas en árbol


repertorio multimedia