JOSÉ JAVIER ALONSO RAMOS 2º A1

PRÁCTICA 3:

PROGRAMACIÓN MIXTA C-ASM y POPCOUNT

popcount.c Sun Nov 04 20:42:28 2018 1

```
    1: //COMANDO PARA LA EJECUCIÃ\223N:
    2: //for i in $(seq 1 4);do rm popcount; gcc -D TEST=$i popcount.c -o popcount; ./popco
unt ; done
    3:
    4: // COMANDO PARA LA DEPURACION DE UN TEST EN CONCRETO
    5: // gcc popcount.c -o popcount -Og -g -D TEST=1
    6:
    7: /*
    8: === TESTS === ________________________________________
    9: for i in 0 g 1 2; do
   10: printf "__OPTIM%1c__%48s\n" $i "" | tr " " "="
   11: for j in $(seq 1 4); do
   12: printf "__TEST%02d__%48s\n" $j "" | tr " " "-"
   13: rm popcount
   14: gcc popcount.c -o popcount -O$i -D TEST=$j -g
   15: ./popcount
   16: done
   17: done
   18: === CRONOS === _____________________________________
   19: for i in 0 g 1 2; do
   20: printf "__OPTIM%1c__%48s\n" $i "" | tr " " "="
   21: rm popcount
   22: gcc popcount.c -o popcount -O$i -D TEST=0
   23: for j in $(seq 0 10); do
   24: echo $j; ./popcount
   25: done | pr -11 -l 22 -w 80
   26: done
   27: ____________________________________________________
   28: */
   29:
   30: #include <stdio.h>              // para printf()
   31: #include <stdlib.h>             // para exit()
   32: #include <stdint.h>             // para uint32_t y uint64_t
   33: #include <sys/time.h>   // para gettimeofday(), struct timeval
   34:
   35: unsigned resultado=0;   // varable donde se almacenarÃ¡ los resultados de los popcou
nts
   36:
   37: #define LONG_SIZE 8*sizeof(long)        // longitud de una variable de tipo LONG en
nuestra arquitectura
   38: #define INT_SIZE 8*sizeof(int)          // longitud de una variable de tipo INT en n
uestra arquitectura
   39:
   40: //      MÃ¡scaras para aplicar en algunos funciones popcount
   41: const uint32_t m1       = 0x55555555;                   //binary: 010101010101010101
01010101010101
   42: const uint32_t m2       = 0x33333333;                   //binary: 001100110011001100
11001100110011
   43: const uint32_t m4       = 0x0f0f0f0f;                   //binary: 000011110000111100
00111100001111
   44: const uint32_t m8       = 0x00ff00ff;                   //binary: 000000001111111100
00000011111111
   45: const uint32_t m16      = 0x0000ffff;                   //binary: 000000000000000011
11111111111111
   46:
   47: const uint64_t m1_64    = 0x5555555555555555;   //binary: 010101010101010101010101
0101010101010101010101010101010101010101
   48: const uint64_t m2_64    = 0x3333333333333333;   //binary: 001100110011001100110011
1100110011001100110011001100110011001100110011
   49: const uint64_t m4_64    = 0x0f0f0f0f0f0f0f0f;   //binary: 000011110000111100001111
0011110000111100001111000011110000111100001111
   50: const uint64_t m8_64    = 0x00ff00ff00ff00ff;   //binary: 000000001111111100000000
1111111100000000111111110000000011111111
   51: const uint64_t m16_64   = 0x0000ffff0000ffff;   //binary: 000000000000000011111111
1111111100000000000000001111111111111111
   52: const uint64_t m32_64   = 0x00000000ffffffff;   //binary: 000000000000000000000000
0000000011111111111111111111111111111111
   53:
```

popcount.c Sun Nov 04 20:42:28 2018 2

```
   54: /* DEFINICIÃ\223N DE LOS TEST
   55:  * DEPENDIENDO DEL TEST ELEGIDO LAS FUNCIONES
   56:  * POPCOUNT TRABAJARÃ\201N CON UNA LISTA DE VALORES U OTRA
   57:  */
   58: #ifndef TEST
   59: #define TEST 20
   60: #endif
   61:
   62: #if TEST==1
   63:         #define SIZE 4
   64:         unsigned lista[SIZE]={0x80000000, 0x00400000, 0x00000200, 0x00000001};
   65: #elif TEST==2
   66:         #define SIZE 8
   67:         unsigned lista[SIZE]={0x7fffffff, 0xffbfffff, 0xfffffdff, 0xfffffffe, 0x0100
0023, 0x00456700, 0x8900ab00, 0x00cd00ef};
   68: #elif TEST==3
   69:         #define SIZE 8
   70:         unsigned lista[SIZE]={0x0, 0x01020408, 0x35906a0c, 0x70b0d0e0, 0xffffffff, 0
x12345678, 0x9abcdef0, 0xdeadbeef};
   71: #elif TEST==4 || TEST==0
   72:         #define NBITS 20
   73:         #define SIZE (1<<NBITS)
   74:         unsigned lista[SIZE];
   75:         #define RESULT ( NBITS * ( 1 << NBITS-1 ) )
   76: #endif
   77:
   78: ///////   CRONO   ////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
   79:
   80: // FUNCIÃ\223N CRONO PROPORCIONADA EN EL GUIÃ\223N
   81: void crono(int (*func)(), char* msg){
   82:         struct timeval tv1,tv2; // gettimeofday() secs-usecs
   83:         long tv_usecs;                  // y sus cuentas
   84:
   85:         gettimeofday(&tv1,NULL);
   86:         resultado = func(lista, SIZE);
   87:         gettimeofday(&tv2,NULL);
   88:
   89:         tv_usecs = (tv2.tv_sec - tv1.tv_sec ) * 1E6 + (tv2.tv_usec - tv1.tv_usec);
   90: #if TEST==0
   91:         printf( "%ld" "\n",     tv_usecs);
   92: #else
   93:         printf("resultado = %d\t", resultado);
   94:         printf("%s:%9ld us\n", msg, tv_usecs);
   95: #endif
   96: }
   97:
   98: ///////   POPCOUNTs   ////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
//
   99:
  100: /* POPCOUNT1:
  101:  * RECORRE TODOS LOS BITS DE CADA DATO DEL VECTOR
  102:  * APLICÃ\201NDOLE LA MÃ\201SCARA 0X1, ES DECIR, QUEDÃ\201NDONOS
  103:  * CADA VEZ CON EL Ã\232LTIMO BIT DEL DATO.
  104:  * PARA CADA NÃ\232MERO SE APLICA LA MÃ\201SCARA EL MISMO NÃ\232MERO
  105:  * DE VECES (INT_SIZE)
  106:  */
  107: int popcount1(unsigned *v, size_t len)
  108: {
  109:         size_t i;
  110:         size_t j;
  111:         unsigned x;
  112:         for (i = 0; i < len; ++i)
  113:         {
  114:                 x = v[i];
  115:
  116:                 for(j = 0; j < INT_SIZE; ++j)
```

```
117:                 {
118:                     resultado += x & 0x1;
119:                         x >>= 1;
120:                 }
121:         }
122:         return resultado;
123: }
124:
125: /* POPCOUNT2:
126:  * RECORRE LOS BITS DE CADA DATO DEL VECTOR HASTA
127:  * QUE TODOS SUS BITS SEAN 0, ES DECIR, V[I] == 0
128:  * LA MÃ\201SCARA QUE SE LE APLICA ES IGUAL A LA DE POPCOUNT1
129:  */
130: int popcount2(unsigned *v, size_t len)
131: {
132:         size_t i;
133:         unsigned x;
134:         for (i = 0; i < len; ++i)
135:         {
136:                 x = v[i];
137:                 while(x)
138:                 {
139:                     resultado += x & 0x1;
140:                         x >>= 1;
141:                 }
142:         }
143:         return resultado;
144: }
145:
146: /* POPCOUNT3:
147:  * EN ESTA VERSIÃ\223N HACEMOS USO DEL ENSAMBLADOR
148:  * EN LÃ\215NEA. CON LA INSTRUCCIÃ\223N SHR DESPLAZAMOS,
149:  * EN CADA ITERACIÃ\223N, UNA POSICIÃ\223N A LA DERECHA.
150:  * EL BIT DESBORDADO SE ALMACENA EN EL FLAG CF Y
151:  * NOS APROVECHAMOS DE ESTO PARA ACUMULARLOS EN
152:  * 'resultado' CON LA INSTRUCCIÃ\223N ADC.
153:  * EL BUCLE INTERNO, AL IGUAL QUE EN LA VERSIÃ\223N
154:  *  ANTERIOR, FINALIZA CUANDO CUANDO V[I] == 0
155:  */
156: int popcount3(unsigned *v, size_t len)
157: {
158:         size_t i;
159:         unsigned x;
160:         for (i = 0; i < len; ++i)
161:         {
162:                 x = v[i];
163:                 asm("\n"
164:                 "ini3:                          \n\t"
165:                         "shr %[x]               \n\t"
166:                         "adc $0, %[r]   \n\t"
167:                         "test %[x], %[x]        \n\t"
168:                         "jne ini3               \n\t"
169:                         :[r]"+r"(resultado)
170:                         :[x]"r"(x)
171:                         );
172:         }
173:         return resultado;
174: }
175:
176: /* POPCOUNT4:
177:  * EN ESTA VERSIÃ\223N HACEMOS USO DEL ENSAMBLADOR
178:  * EN LÃ\215NEA. ES IGUAL QUE LA VERSIÃ\223N ANTERIOR
179:  * PERO AQUÃ\215 EVITAMOS EL USO DE TEST YA QUE
180:  * LA INSTRUCCIÃ\223N SHR ALTERA TAMBIÃ\211N EL FLAG ZF
181:  * EL USO DE LA ETIQUETA 'fin4' NO LO HE VISTO
182:  * NECESARIO
183:  */
184: int popcount4(unsigned *v, size_t len)
```

```
  185: {
  186:         size_t i;
  187:         unsigned x;
  188:
  189:         for (i = 0; i < len; ++i)
  190:         {
  191:                 x = v[i];
  192:                 asm("clc                        \n\t"   //      LIMPIAMOS EL CONTENI
DO DE LOS FLAGS
  193:                 "ini4:                          \n\t"
  194:                         "adc $0, %[r]   \n\t"
  195:                         "shr %[x]               \n\t"
  196:                         "jne ini4               \n\t"   // SI %[x] ES DISTINTO DE 0
REPITE EL BUCLE
  197:                         "adc $0, %[r]   \n\t"
  198:                         :[r]"+r"(resultado)
  199:                         :[x]"r"(x)
  200:                         :"cc"
  201:                         );
  202:         }
  203:         return resultado;
  204: }
  205:
  206: /* POPCOUNT5:
  207:  * EN ESTA VERSIÃ\223N REALIZAMOS LA SUMA DE BITS
  208:  * ACTIVOS BYTE A BYTE GRACIAS A LA MÃ\201SCARA 0x01010101
  209:  * CON ESTA MÃ\201SCARA OBTENEMOS EL PRIMER BIT DE CADA BYTE
  210:  * DE UN DATO DEL VECTOR. POR ESTO DEBEMOS REALIZAR
  211:  * 8 DESPLAZAMIENTOS. TRAS REALIZAR TODOS LOS DESPLAZAMIENTOS
  212:  * SUMAREMOS LOS BYTES EN FORMA DE ARBOL DE LA VARIABLE
  213:  * 'resultado' Y APLICAREMOS UNA MÃ\201SCARA PARA QUEDARNOS
  214:  * SOLO CON LOS VALORES DEL Ã\232LTIMO BYTE (EL VALOR CORRECTO
  215:  * DE LA SUMA - EL PESO HAMMING)
  216:  */
  217: int popcount5(unsigned *v, size_t len)
  218: {
  219:         unsigned val = 0, x;
  220:         size_t i,j;
  221:         for(j = 0; j < len ; ++j)
  222:         {
  223:                 x = v[j];
  224:                 val = 0;
  225:                 for (i = 0; i < 8; ++i)
  226:                 {
  227:
  228:                         val += x & 0x01010101;
  229:                         x >>= 1;
  230:                 }
  231:                 val += (val >> 16);
  232:                 val += (val >> 8);
  233:
  234:                 resultado += (val & 0xFF);
  235:         }
  236:
  237:         return resultado;
  238: }
  239:
  240: /* POPCOUNT6:
  241:  * ESTA VERSIÃ\223N PODRÃ\215A CONSIDERARSE UN DESENROLLAMIENTO
  242:  * DE LA VERSIÃ\223N ANTERIOR. EN VEZ DE REALIZAR UNA SUMA
  243:  * TRATANDOLA COMO SI FUERA UN Ã\201RBOL, APLICAMOS VARIAS MÃ\201SCARAS
  244:  * PARA IR SUMANDO CADA 2 BITS, 4 BITS, 8 BITS, 16 BITS Y POR
  245:  * Ã\232LTIMO CADA 32
  246:  */
  247: int popcount6(unsigned *v, size_t len)
  248: {
  249:         size_t i;
  250:         unsigned x;
```

```
  251:         for(i=0; i<len; ++i)
  252:         {
  253:                 x = v[i];
  254:                 x = (x & m1 ) + ((x >> 1) & m1 ); //put count of each 2 bits into th
ose 2 bits
  255:                 x = (x & m2 ) + ((x >> 2) & m2 ); //put count of each 4 bits into th
ose 4 bits
  256:                 x = (x & m4 ) + ((x >> 4) & m4 ); //put count of each 8 bits into th
ose 8 bits
  257:                 x = (x & m8 ) + ((x >> 8) & m8 ); //put count of each 16 bits into t
hose 16 bits
  258:                 x = (x & m16) + ((x >> 16) & m16); //put count of each 32 bits into
those 32 bits
  259:
  260:                 resultado+=x;
  261:         }
  262:         return resultado;
  263: }
  264:
  265: /* POPCOUNT7:
  266:  * ESTA VERSIÃ\223N ESTÃ\201 AÃ\232N MÃ\201S DESENROLLADA SI CABE.
  267:  * LAS MÃ\201SCARAS APLICADAS EN LA VERSIÃ\223N ANTERIOR
  268:  * LAS TRANSFORMAMOS EN MÃ\201SCARAS DE 64BITS YA QUE
  269:  * VAMOS A ASIGNAR DOS COMPONENTES DEL VECTOR DE ENTEROS
  270:  * A UNA MISMA VARIABLE (HABRÃ\201 DOS DATOS EN UNA MISMA VARIABLE
  271:  * X = V[i]V[i+1] ). SI ESTO LO REALIZAMOS DOS VECES,
  272:  * ES DECIR, TENEMOS DOS VARIABLES LONG Y EN CADA UNA DE
  273:  * ELLAS 2 VARIABLES DEL VECTOR ESTAREMOS TOMANDO 128 BITS
  274:  * EN UNA SOLA ITERACIÃ\223N.
  275:  */
  276: int popcount7(unsigned *v, size_t len)
  277: {
  278:         size_t i;
  279:         unsigned long x,y;
  280:
  281:         if (len & 0x3)
  282:                 printf("leyendo 128b pero len no mÃºltiplo de 4\n");
  283:
  284:         for(i=0; i<len; i+=4)
  285:         {
  286:                 x = *(unsigned long*) &v[i];
  287:                 y = *(unsigned long*) &v[i+2];
  288:
  289:                 x = (x & m1_64 ) + ((x >> 1) & m1_64 ); //put count of each 2 bits i
nto
  290:                 x = (x & m2_64 ) + ((x >> 2) & m2_64 ); //put count of each 4 bits i
nto
  291:                 x = (x & m4_64 ) + ((x >> 4) & m4_64 ); //put count of each 8 bits i
nto
  292:                 x = (x & m8_64 ) + ((x >> 8) & m8_64 ); //put count of each 16 bits
into
  293:                 x = (x & m16_64) + ((x >> 16) & m16_64); //put count of each 32 bits
 into
  294:                 x = (x & m32_64) + ((x >> 32) & m32_64); //put count of each 64 bits
 into
  295:
  296:                 y = (y & m1_64 ) + ((y >> 1) & m1_64 ); //put count of each 2 bits i
nto
  297:                 y = (y & m2_64 ) + ((y >> 2) & m2_64 ); //put count of each 4 bits i
nto
  298:                 y = (y & m4_64 ) + ((y >> 4) & m4_64 ); //put count of each 8 bits i
nto
  299:                 y = (y & m8_64 ) + ((y >> 8) & m8_64 ); //put count of each 16 bits
into
  300:                 y = (y & m16_64) + ((y >> 16) & m16_64); //put count of each 32 bits
 into
  301:                 y = (y & m32_64) + ((y >> 32) & m32_64); //put count of each 64 bits
 into
```

popcount.c Sun Nov 04 20:42:28 2018 6

```
  302:
  303:                  resultado += x+y;
  304:          }
  305:          return resultado;
  306: }
  307:
  308: /* POPCOUNT8:
  309:  * EN ESTA VERSIÃ\223N HACEMOS USO DE INSTRUCCIONES
  310:  * MULTIMEDIA SSSE3 Y EN PARTICULAR, LA INTERANTE ES
  311:  * PSHUFB, UNA INSTRUCCIÃ\223N SIMD (SINGLE INSTRUCTION MULTIPLE DATA)
  312:  * QUE NOS PERMITE BARAJAR EL SEGUNDO PARÃ\201METRO PASADO
  313:  * SEGUN LAS RESTRICCIONES IMPUESTAS CON EL PRIMERO.
  314:  * ASM CARGA 4 ENTEROS EN UN REGISTRO XMM Y SACA COPIA EN OTRO XMM.
  315:  * CARGA UNA MÃ\201SCARA PARA QUEDARSE CON NIBBLES INFERIORES. DESPLAZA
  316:  * 4B UNA DE LAS COPIAS, DE MANERA QUE AL APLICAR LA MÃ\201SCARA A AMBAS
  317:  * COPIAS, RESULTEN SEPARADOS LOS NIBBLES INFERIORES Y SUPERIORES EN
  318:  * SU CORRESPONDIENTE REGISTRO XMM. SE CARGAN ENTONCES DOS COPIAS DE
  319:  * LA LUT Y SE BARAJAN USANDO COMO Ã\215NDICES LOS NIBBLES, OBTENIENDO
  320:  * LOS POPCOUNT RESPECTIVOS
  321:  */
  322: int popcount8(unsigned* v, size_t len){
  323:          size_t i;
  324:          int val;
  325:          int SSE_mask[] = {0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f};
  326:          int SSE_LUTb[] = {0x02010100, 0x03020201, 0x03020201, 0x04030302};
  327:                                          //      3 2 1 0         7 6 5 4     11109 8
    15141312
  328:
  329:          if (len & 0x3) printf("leyendo 128b pero len no mÃºltiplo de 4\n");
  330:          for (i=0; i<len; i+=4)
  331:          {
  332:                  asm("movdqu     %[x], %%xmm0    \n\t"
  333:                          "movdqa %%xmm0, %%xmm1  \n\t" //; x: two copies xmm0-1
  334:                          "movdqu %[m], %%xmm6    \n\t" //: mask: xmm6
  335:                          "psrlw  $4 , %%xmm1     \n\t"
  336:                          "pand   %%xmm6, %%xmm0  \n\t" //; xmm0 â\200\223 lower nibbl
es
  337:                          "pand   %%xmm6, %%xmm1  \n\t" //; xmm1 â\200\223 higher nibb
les
  338:
  339:                          "movdqu %[l], %%xmm2    \n\t" //; since instruction pshufb m
odifies LUT
  340:                          "movdqa %%xmm2, %%xmm3  \n\t" //; we need 2 copies
  341:                          "pshufb %%xmm0, %%xmm2  \n\t" //; xmm2 = vector of popcount
lower nibbles
  342:                          "pshufb %%xmm1, %%xmm3  \n\t" //; xmm3 = vector of popcount
upper nibbles
  343:
  344:                          "paddb %%xmm2, %%xmm3   \n\t" //; xmm3 â\200\223 vector of p
opcount for bytes
  345:                          "pxor %%xmm0, %%xmm0    \n\t" //; xmm0 = 0,0,0,0
  346:                          "psadbw %%xmm0, %%xmm3  \n\t" //; xmm3 = [pcnt bytes0..7|pcn
t bytes8..15]
  347:                          "movhlps %%xmm3, %%xmm0 \n\t" //; xmm0 = [             0
       |pcnt bytes0..7 ]
  348:                          "paddd %%xmm3, %%xmm0   \n\t" //; xmm0 = [      not needed
|pcnt bytes0..15]
  349:                          "movd %%xmm0, %[val]            "
  350:                          : [val]"=r" (val)
  351:                          : [x] "m" (v[i]),
  352:                          [m] "m" (SSE_mask[0]),
  353:                          [l] "m" (SSE_LUTb[0])
  354:                  );
  355:
  356:                  resultado += val;
  357:          }
  358:          return resultado;
  359: }
```

popcount.c Sun Nov 04 20:42:28 2018 7

```
360:
361: /* POPCOUNT9:
362:  * EN ESTA VERSIÃ\223N HACEMOS USO DE UNA INSTRUCCIÃ\223N SSE4
363:  * 'popcnt' QUE CALCULA EL PESO HAMMING DEL PRIMER PARÃ\201METRO
364:  * Y LO ALMACENA EN EL SEGUNDO.
365:  */
366: int popcount9(unsigned* v, size_t len){
367:         size_t i;
368:         int val;
369:
370:         for (i=0; i<len; ++i)
371:         {
372:                 asm("popcnt     %[x], %[val]"
373:                         : [val]"=r" (val)
374:                         : [x] "m" (v[i])
375:                 );
376:
377:                 resultado += val;
378:         }
379:         return resultado;
380: }
381:
382: /* POPCOUNT9:
383:  * ESTA VERSIÃ\223N ES IGUAL QUE LA ANTERIOR PERO APLICANDO LA FILOSOFÃ\215A
384:  * DE LA VERSIÃ\223N 7, DESENROLLAR EL BUCLE TRABAJANDO CON 128 BITS
385:  * EN UNA SOLA ITERACIÃ\223N.
386:  */
387: int popcount10(unsigned* v, size_t len){
388:         size_t i;
389:         unsigned long x, y;
390:         long val;
391:
392:         if (len & 0x3) printf("leyendo 128b pero len no mÃºltiplo de 4\n");
393:
394:         for (i=0; i<len; i+=4)
395:         {
396:                 x = *(unsigned long*) &v[i];
397:                 y = *(unsigned long*) &v[i+2];
398:                 asm("popcnt     %[y], %[val]    \n\t"
399:                         "popcnt %[x], %[y]              \n\t"
400:                         "add    %[y], %[val]    \n\t"
401:                         :       [val]"+r" (val),
402:                                 [y]"+r" (y)
403:                         :       [x] "m" (x)
404:                 );
405:
406:                 resultado += val;
407:         }
408:         return resultado;
409: }
410:
411: ///////   MAIN   //////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////////////
412:
413: int main()
414: {
415: // SI ESTAMOS EJECUTANDO TEST0 O TEST4 INICIALIZAMOS LA LISTA
416: // DE SIZE ENTEROS DE MANERA QUE lista[0] == 0 y lista[SIZE] == SIZE-1
417: #if TEST==0 || TEST==4
418:         unsigned i;
419:         for (i=0; i<SIZE; i++)
420:                 lista[i]=i;
421: #endif
422:
423:         // EJECUTAMOS TODAS LAS VERSIONES DE popcount DENTRO DE crono PARA
424:         // CALCULAR SU TIEMPO DE EJECUCIÃ\223N. AL FINAL DE CADA crono IGUALAMOS
425:         // resultado A 0 YA QUE LA VARIABLE ES GLOBAL.
426:         crono(popcount1 ,"popcount1 (lenguaje C -       for)"); resultado = 0;
```

popcount.c Sun Nov 04 20:42:28 2018 8

```
427:          crono(popcount2 ,"popcount2 (lenguaje C -       while)"); resultado = 0;
428:          crono(popcount3 ,"popcount3 (leng.ASM-body while 4i)"); resultado = 0;
429:          crono(popcount4 ,"popcount4 (leng.ASM-body while 3i)"); resultado = 0;
430:          crono(popcount5 ,"popcount5 (CS:APP2e 3.49-group 8b)"); resultado = 0;
431:          crono(popcount6 ,"popcount6 (Wikipedia- naive - 32b)"); resultado = 0;
432:          crono(popcount7 ,"popcount7 (Wikipedia- naive -128b)"); resultado = 0;
433:          crono(popcount8 ,"popcount8 (asm SSE3 - pshufb 128b)"); resultado = 0;
434:          crono(popcount9 ,"popcount9 (asm SSE4- popcount 32b)"); resultado = 0;
435:          crono(popcount10,"popcount10(asm SSE4- popcount128b)");
436:
437:          printf("\n");
438:
439: #if TEST != 0
440:          printf("calculado = %d\n", resultado);
441: #endif
442:          exit(0);
443: }
```

lscpu:	CPU(s): 8 Nombre del modelo: Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz Virtualización: VT-x Caché L3: 8192K

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):	62764	67417	65081	67200	62962	61999	62884	62140	64915	63928	63745	64227
popcount2 (lenguaje C - while):	33859	33775	33530	33680	33621	33512	33533	33778	35592	33546	33511	33808
popcount3 (leng.ASM-body while 4i):	9959	10096	9957	10000	10051	9947	11466	9956	9950	9939	9942	10130
popcount4 (leng.ASM-body while 3i):	9181	9301	9175	9185	9280	9231	9189	9234	9193	9189	9179	9216
popcount5 (CS:APP2e 3.49-group 8b):	16601	16766	16677	16668	16733	16629	16621	16599	16598	16673	16615	16658
popcount6 (Wikipedia- naive - 32b):	7053	7144	7057	7066	7123	7025	7044	7126	7165	7045	7042	7084
popcount7 (Wikipedia- naive -128b):	3442	3565	3441	3439	3539	3439	3445	3438	3445	4076	3442	3527
popcount8 (asm SSE3 - pshufb 128b):	693	766	691	693	746	694	690	695	701	1263	691	763
popcount9 (asm SSE4- popcount 32b):	2031	2093	2031	2032	2076	2033	2032	2028	2039	3418	2035	2182
popcount10(asm SSE4- popcount128b):	744	837	743	742	819	741	742	742	742	1071	742	792

Optimización -Og	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	54754	51472	51237	52152	53204	52349	52024	51958	51363	51440	51798	51900
popcount2 (lenguaje C - while):	30195	30081	30913	30178	30143	30263	30199	30183	30211	30246	30293	30271
popcount3 (leng.ASM-body while 4i):	8858	8995	8865	8836	8839	8845	8863	8841	8837	9397	8850	8917
popcount4 (leng.ASM-body while 3i):	8654	8744	8660	8653	8707	8651	8659	8647	8657	8704	8651	8673
popcount5 (CS:APP2e 3.49-group 8b):	5257	5253	5259	5250	5248	5248	6196	5247	5248	5427	5248	5362
popcount6 (Wikipedia- naive - 32b):	2263	2276	2281	2287	2265	2267	2260	2263	2264	2292	2262	2272
popcount7 (Wikipedia- naive -128b):	1312	1364	1312	1311	1316	1312	1316	1311	1312	1312	1312	1318
popcount8 (asm SSE3 - pshufb 128b):	398	397	396	399	396	397	407	396	398	399	398	398
popcount9 (asm SSE4- popcount 32b):	1589	1589	1591	1590	1589	1591	1620	1588	1589	1594	1593	1593
popcount10(asm SSE4- popcount128b):	397	401	395	395	396	395	410	395	394	400	397	398

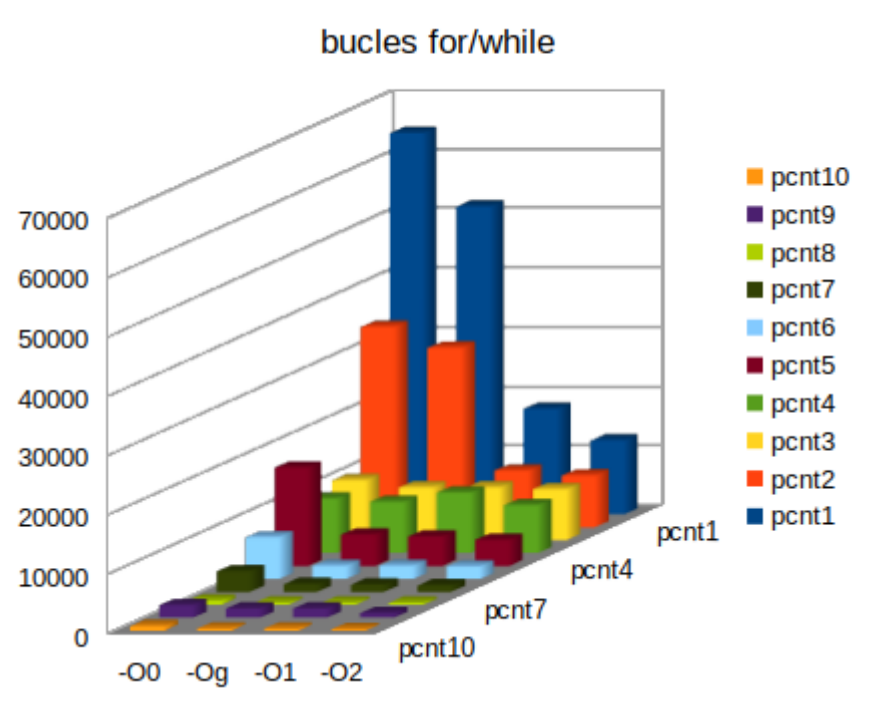
JOSÉ JAVIER ALONSO RAMOS 2º A1

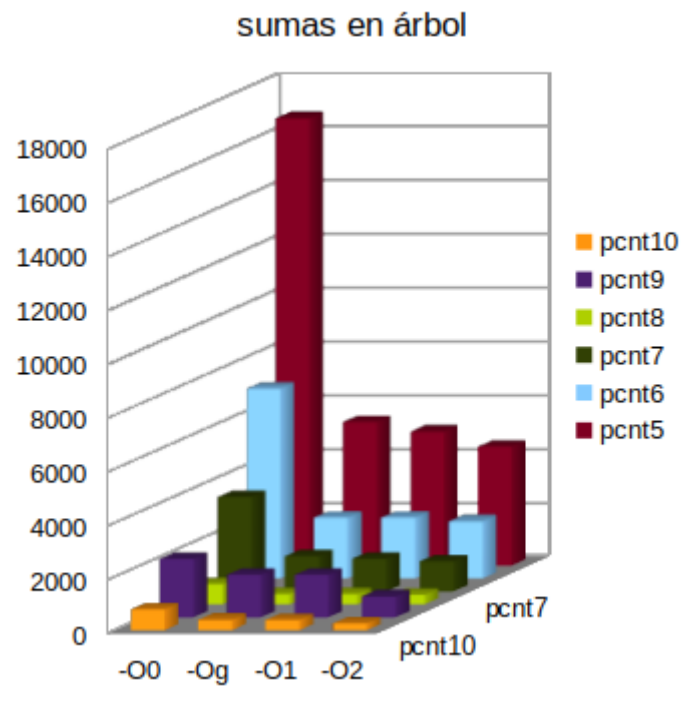
Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	20599	17213	17088	17084	18881	17112	18293	18331	18083	18320	18313	17872
popcount2 (lenguaje C - while):	8580	9974	7933	10074	10018	9156	10198	10166	9975	10001	8600	9610
popcount3 (leng.ASM-body while 4i):	8997	8950	8878	8875	8882	8880	8898	8879	8870	8900	8878	8889
popcount4 (leng.ASM-body while 3i):	10405	10248	10254	10237	10262	10256	10236	10240	10236	10236	10234	10244
popcount5 (CS:APP2e 3.49-group 8b):	4979	5251	4956	4883	5107	4972	4842	5248	4833	4928	5040	5006
popcount6 (Wikipedia- naive - 32b):	2261	2284	2259	2282	2303	2273	2323	2261	2261	2261	2259	2277
popcount7 (Wikipedia- naive -128b):	1215	1215	1214	1214	1225	1213	1304	1213	1213	1213	1216	1224
popcount8 (asm SSE3 - pshufb 128b):	403	396	398	400	395	394	402	395	394	396	394	396
popcount9 (asm SSE4- popcount 32b):	1590	1593	1590	1593	1591	1585	1586	1589	1583	1588	1588	1589
popcount10(asm SSE4- popcount128b):	399	397	395	397	395	397	406	395	395	396	395	397

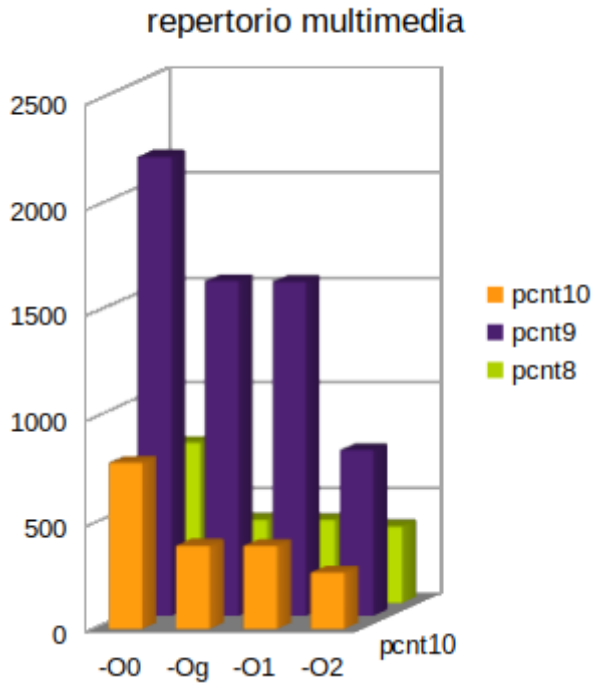
Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	14843	12535	11652	11886	13011	14964	11739	12950	12981	11658	11619	12500
popcount2 (lenguaje C - while):	9328	8578	8504	8509	9074	8443	8381	8307	9912	9275	8503	8749
popcount3 (leng.ASM-body while 4i):	8922	8641	8638	8629	8650	8651	8619	8654	8642	8622	8640	8639
popcount4 (leng.ASM-body while 3i):	8239	8111	8117	8120	8131	8114	8122	8157	8107	8127	8111	8122
popcount5 (CS:APP2e 3.49-group 8b):	4031	4461	4489	4638	3934	4724	4725	4093	4343	4302	4724	4443
popcount6 (Wikipedia- naive - 32b):	2285	2197	2120	2120	2125	2120	2120	2194	2120	2120	2124	2136
popcount7 (Wikipedia- naive -128b):	1114	1147	1117	1117	1114	1125	1130	1201	1120	1123	1122	1132
popcount8 (asm SSE3 - pshufb 128b):	362	380	362	362	362	362	362	391	363	362	362	367
popcount9 (asm SSE4- popcount 32b):	803	789	789	790	790	790	789	798	789	790	790	790
popcount10(asm SSE4- popcount128b):	271	269	269	269	270	268	269	285	270	269	268	271

POPCOUNT:	-O0	-Og	-O1	-O2
pcnt1	64227	51900	**17872**	12500
pcnt2	33808	**30271**	9610	8749
pcnt3	10130	8917	**8889**	8639
pcnt4	9216	8673	**10244**	8122
pcnt5	16658	5362	5006	**4443**
pcnt6	7084	2272	2277	**2136**
pcnt7	3527	1318	1224	**1132**
pcnt8	763	398	396	**367**
pcnt9	2182	1593	1589	**790**
pcnt10	792	**398**	**397**	**271**

Ganancias:	-O0	-Og	-O1	-O2
pcnt1			1,00	
pcnt2		**0,59**		
pcnt3			2,01	
pcnt4			1,74	
pcnt5				**4,02**
pcnt6				**8,37**
pcnt7				**15,79**
pcnt8				**48,72**
pcnt9				22,61
pcnt10		**44,93**	45,04	66,05







Además de incluir el código y los cálculos de tiempo en el pdf también se añadirán al archivo comprimido de la entrega.

- El diario de trabajo ha sido el siguiente:

Durante las horas de prácticas semanales he leído el Tutorial 3a de la práctica y realizado sus actividades.
Como trabajo autónomo he estudiado el ensamblador en línea, así como el uso de máscaras de bits para realizar el popcount para lograr comprender los ejercicios y poder realizarlos.
Los ejercicios de popcount descritos en Guión 3b los he realizado en el fin de semana siendo distribuidos los cinco primeros el sábado y los cinco últimos el domingo. El estudio de los tiempos de las funciones, comentar el código y la realización de esta memoria también se ha llevado a cabo el domingo.