DIARIO DE TRABAJO:

- Día 1: Lectura completa de la practica, empiezo a entender como funcionan los diferentes modos de popcount, sobre todo los explicados en clase.
- Día 2: Realizo las dos primeras versiones de popcount en C, no tuve complicación alguna ya que es simple código en C, el cual fue explicado en clase.
- Día 3: Realizo los tres siguientes popcount, siendo el popcount3 y el popcount4 iguales simplemente usando la instrucción CLT en este ultimo, es su diferencia que no me hizo perder mucho tiempo ya que lo entendí perfectamente y en popcount5 era escribir en C luego tampoco tuve problemas.
- Día 4: Realizo todo lo que faltaba de la practica, ya que era muy guiado no tuve problemas, pero no termino de entender como funcionan los popcount últimos en los que se utilizan instrucciones multimedia. En cuanto al popcount en el que se utiliza “naivi” los he entendido simplemente los pasé a 32 bits, fue explicado en clase perfectamente.
- Día 5: Me doy cuenta que el ultimo popcount, el 10, me produce un fallo con optimización 2, el profesor me ha aclarado que ha sido debido a que el registro usado para acumular puede que lo estuviera usando para otra cosa, luego tuve que decirle al compilador que lo usaba para compilar.
- Día 6: Ejecuto el script e introduzco tiempos en la hoja de calculo. Finalizo la practica.

CÓDIGO:

```
#include <stdio.h> // para printf()
#include <stdint.h>
#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 156
/* ------------------------------------------------------------------ */
#elif TEST==3
  #define SIZE 8
    unsigned lista[SIZE]={0x0 , 0x01020408, 0x35906a0c, 0x70b0d0e0, 0xffffffff, 0x12345678,
0x9abcdef0, 0xdeadbeef};
```

```
  #define RESULT 116
/* --------------------------------------------------------------- */
#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 * ( NBITS << NBITS-1 ) ) // pistas para deducir fórmula
/* --------------------------------------------------------------- */
#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++){
             x = array[i];
    for (j = 0; j < 8 * sizeof(unsigned); j++){
      result += x & 0x1;
      x >>= 1;
    }
  }

  return result;
}

int popcount2(unsigned* array, size_t len){
  long result = 0;
  size_t i=0;
  unsigned x;
  for (i = 0; i < len; i++){
            x = array[i];
   while (x){
     result += x & 0x1;
     x >>= 1;
   }
  }

  return result;
}

int popcount3(unsigned* array, size_t len){
  int result = 0;
  size_t i=0;
  unsigned x;
```

```
  for (i = 0; i < len; i++){
            x = array[i];
    asm("           \n  "
      "ini3:         \n\t" // seguir mientras que x!=0
      "shr %[x]      \n\t" // Desplaza todos los bits
      "adc $0x0,  %[r]\n\t" // Si se activa el bit de signo sumo carry+result
      "test %[x], %[x]\n\t" // Se hace para activar o no el ZF
      "jnz ini3"           // Salta si no es cero
      :[r] "+r" (result)
      :[x] "r" (x));
  }
  return result;
}

int popcount4(unsigned* array, size_t len){
  int result = 0;
  size_t i=0;
  unsigned x;
  for (i = 0; i < len; i++){
            x = array[i];
    asm("            \n  "
      "clc           \n\t"
      "ini4:         \n\t" // seguir mientras que x!=0
      "adc $0,  %[r]\n\t" // Si se activa el bit de signo sumo carry+result
      "shr %[x]      \n\t" // Desplaza todos los bits
      "jnz ini4    \n\t"           // Salta si no es cero
      "adc $0,  %[r]\n\t" // Si se activa el bit de signo sumo carry+result
      :[r] "+r" (result)
      :[x] "r" (x));
  }
  return result;
}

int popcount5(unsigned* array, size_t len) {
  int result = 0;
  size_t i,j;
  unsigned x, valor;

  for (i = 0; i < len; i++) {
    valor=0;
    x = array[i];
    for (j = 0; j < 8; j++) {
      valor += x & 0x01010101;
      x >>= 1;
    }
    valor += (valor >> 16);
    valor += (valor >> 8);
    valor&=0xFF;
```

```
    result += valor;
  }

  return result;
}

int popcount6(unsigned* array, size_t len){
  int result = 0;
  size_t i;
  unsigned x, valor;

  const unsigned long m1 = 0x5555555555555555; //binary: 0101...
  const unsigned long m2 = 0x3333333333333333; //binary: 00110011..
  const unsigned long m4 = 0x0f0f0f0f0f0f0f0f; //binary: 4 zeros, 4 ones ...
  const unsigned long m8 = 0x00ff00ff00ff00ff; //binary: 8 zeros, 8 ones ...
  const unsigned long m16 =0x0000ffff0000ffff; //binary: 16 zeros, 16 ones ...
  const unsigned long m32 =0x00000000ffffffff; //binary: 32 zeros, 32 ones

  for (i = 0; i < len; i++) {
    x=array[i];
    x = (x & m1 ) + ((x >> 1) & m1 ); //put count of each 2 bits into those 2 bits
    x = (x & m2 ) + ((x >> 2) & m2 ); //put count of each 4 bits into those 4 bits
    x = (x & m4 ) + ((x >> 4) & m4 ); //put count of each 8 bits into those 8 bits
    x = (x & m8 ) + ((x >> 8) & m8 ); //put count of each 16 bits into those 16 bits
    x = (x & m16) + ((x >> 16) & m16); //put count of each 32 bits into those 32 bits
    //x = (x & m32) + ((x >> 32) & m32); //put count of each 64 bits into those 64 bits
    result += x;
  }
  return result;
}

int popcount7(unsigned* array, size_t len){
  int result = 0;
  size_t i;
  unsigned long x1,x2, valor;

  const unsigned long m1 = 0x5555555555555555; //binary: 0101...
  const unsigned long m2 = 0x3333333333333333; //binary: 00110011..
  const unsigned long m4 = 0x0f0f0f0f0f0f0f0f; //binary: 4 zeros, 4 ones ...
  const unsigned long m8 = 0x00ff00ff00ff00ff; //binary: 8 zeros, 8 ones ...
  const unsigned long m16 =0x0000ffff0000ffff; //binary: 16 zeros, 16 ones ...
  const unsigned long m32 =0x00000000ffffffff; //binary: 32 zeros, 32 ones

  for (i = 0; i < len; i+=4) {
    x1=*(unsigned long*) &array[i];
    x2=*(unsigned long*) &array[i+2];
    x1 = (x1 & m1 ) + ((x1 >> 1) & m1 ); //put count of each 2 bits into those 2 bits
    x1 = (x1 & m2 ) + ((x1 >> 2) & m2 ); //put count of each 4 bits into those 4 bits
```

```
    x1 = (x1 & m4 ) + ((x1 >> 4) & m4 ); //put count of each 8 bits into those 8 bits
    x1 = (x1 & m8 ) + ((x1 >> 8) & m8 ); //put count of each 16 bits into those 16 bits
    x1 = (x1 & m16) + ((x1 >> 16) & m16); //put count of each 32 bits into those 32 bits
    x1 = (x1 & m32) + ((x1 >> 32) & m32); //put count of each 64 bits into those 64 bits
    x2 = (x2 & m1 ) + ((x2 >> 1) & m1 ); //put count of each 2 bits into those 2 bits
    x2 = (x2 & m2 ) + ((x2 >> 2) & m2 ); //put count of each 4 bits into those 4 bits
    x2 = (x2 & m4 ) + ((x2 >> 4) & m4 ); //put count of each 8 bits into those 8 bits
    x2 = (x2 & m8 ) + ((x2 >> 8) & m8 ); //put count of each 16 bits into those 16 bits
    x2 = (x2 & m16) + ((x2 >> 16) & m16); //put count of each 32 bits into those 32 bits
    x2 = (x2 & m32) + ((x2 >> 32) & m32); //put count of each 64 bits into those 64 bits
    result += x1+x2;
  }
  return result;
}

int popcount8 (unsigned* array, int len){
    int i, 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 multiplo de 4?\n");

    for(i=0; i<len; i+=4){
      asm("movdqu  %[x],   %%xmm0      \n\t"
          "movdqa  %%xmm0, %%xmm1      \n\t"
          "movdqu    %[m], %%xmm6      \n\t"
          "psrlw       $4, %%xmm1      \n\t"
          "pand    %%xmm6, %%xmm0      \n\t"
          "pand    %%xmm6, %%xmm1      \n\t"

          "movdqu    %[l], %%xmm2      \n\t"
          "movdqa  %%xmm2, %%xmm3      \n\t"
          "pshufb  %%xmm0, %%xmm2      \n\t"
          "pshufb  %%xmm1, %%xmm3      \n\t"

          "paddb   %%xmm2, %%xmm3      \n\t"
          "pxor    %%xmm0, %%xmm0      \n\t"
          "psadbw  %%xmm0, %%xmm3      \n\t"
          "movhlps %%xmm3, %%xmm0      \n\t"
          "paddd   %%xmm3, %%xmm0      \n\t"
          "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, int len){
    int i, val, result = 0;
    unsigned x1,x2;
    if( len & 0x1)
      printf("leer 64b y len impar?\n");
    for(i=0; i<len; i+=2){
      x1 = array[i];
      x2 = array[i+1];
      asm("popcnt %[x1], %[val] \n\t"
          "popcnt %[x2], %%edi  \n\t"
          "add    %%edi, %[val] \n\t"
          : [val] "=&r" (val)
          :  [x1]  "r"  (x1),
             [x2]  "r"  (x2)
          : "edi");
      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)
        : "cc","rdi");
    result += val;
  }
  return result;
}

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);
}
```

L-2.9

lscpu:

```
CPU(s):                  4
Nombre del modelo:       Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
Virtualización:          VT-x
Caché L3:                3072 K
```

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):	106428	102331	112542	104329	99570	103805	101966	103520	103512	100841	103282	103570
popcount2 (lenguaje C - while):	53887	67319	56633	55179	52117	52161	59186	52339	53624	50037	63926	56252
popcount3 (leng.ASM-body while 4i):	15920	17101	17280	15008	15515	14930	15928	16281	17100	16239	18586	16397
popcount4 (leng.ASM-body while 3i):	14533	14845	14192	16344	14807	13853	14316	14645	14776	14415	13751	14594
popcount5 (CS:APP2e 3.49-group 8b):	26477	27393	25358	25684	26969	25456	26877	31634	26852	25836	26074	26813
popcount6 (Wikipedia- naive - 32b):	10994	12771	10249	10377	10681	10344	10947	15144	10104	10558	10343	11152
popcount7 (Wikipedia- naive -128b):	6435	7813	6043	7813	6108	5995	6424	6566	6199	6383	6449	6629
popcount8 (asm SSE3 - pshufb 128b):	1162	1183	1065	1077	1066	1400	1107	1117	1122	1170	1088	1145
popcount9 (asm SSE4- popcount 32b):	2539	2440	2195	2271	2384	2268	2248	2334	2242	2381	2197	2298
popcount10(asm SSE4- popcount128b):	1169	1155	1215	1080	1255	1086	1160	1324	1075	1090	1059	1138

Optimización -Og	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	36433	29018	29287	26958	31630	27464	35305	30845	27399	26943	26846	29170
popcount2 (lenguaje C - while):	15257	10208	10493	10353	10585	11005	10592	11055	10828	10573	10195	10589
popcount3 (leng.ASM-body while 4i):	19835	15998	16052	16421	16342	18976	16032	17478	16327	19654	15900	16918
popcount4 (leng.ASM-body while 3i):	15566	12378	12514	12367	12327	14281	12226	16874	12559	12878	15203	13361
popcount5 (CS:APP2e 3.49-group 8b):	7086	7305	6724	6763	6937	8820	6760	6692	6685	7983	6737	7254
popcount6 (Wikipedia- naive - 32b):	3490	3700	3391	3428	3668	4025	3650	3595	3446	3618	3747	3653
popcount7 (Wikipedia- naive -128b):	2009	2156	2169	2050	2047	2227	2025	1984	1999	2016	1991	2085
popcount8 (asm SSE3 - pshufb 128b):	548	594	529	620	545	649	539	581	523	539	517	570
popcount9 (asm SSE4- popcount 32b):	651	649	603	605	612	670	613	614	635	611	678	633
popcount10(asm SSE4- popcount128b):	424	441	441	412	414	521	415	419	408	414	410	436

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	25668	20012	19196	18378	16680	16493	17669	16601	17501	17033	18151	17771
popcount2 (lenguaje C - while):	25524	23610	16678	16199	17597	17587	16442	16004	16876	18889	17288	17717
popcount3 (leng.ASM-body while 4i):	13332	13264	15340	13090	13665	13859	13075	13095	13307	13259	13931	13589
popcount4 (leng.ASM-body while 3i):	14811	13187	15849	12707	15602	13621	14677	15780	15475	13767	16062	14673
popcount5 (CS:APP2e 3.49-group 8b):	7570	9090	7124	7339	7255	7098	7701	7088	7823	7299	7721	7578
popcount6 (Wikipedia- naive - 32b):	3591	4054	3407	3502	3426	4162	3684	3437	3689	3506	3685	3678
popcount7 (Wikipedia- naive -128b):	1897	1970	1911	1852	2062	2617	1825	1904	2060	2006	1977	2028
popcount8 (asm SSE3 - pshufb 128b):	524	554	614	527	537	1849	711	620	587	559	566	740
popcount9 (asm SSE4- popcount 32b):	599	612	678	661	604	703	675	673	703	602	607	638
popcount10(asm SSE4- popcount128b):	404	453	420	415	407	450	455	416	456	411	405	423

Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	media
popcount1 (lenguaje C - for):	19070	16628	17209	17090	16852	16641	16614	17258	16623	17156	17660	16973
popcount2 (lenguaje C - while):	10106	10077	11959	10276	10074	10108	11763	11753	10260	15957	10283	11251
popcount3 (leng.ASM-body while 4i):	12878	13024	12966	13514	13749	13501	13454	13629	12718	13843	12966	13336
popcount4 (leng.ASM-body while 3i):	12234	14219	12289	13617	12048	13383	13211	12366	14178	13561	12433	13131
popcount5 (CS:APP2e 3.49-group 8b):	5521	6954	5553	5733	5653	5865	5931	5788	6243	5497	5665	5856
popcount6 (Wikipedia- naive - 32b):	3030	3063	3168	3057	3007	3038	3230	3124	3223	3046	3218	3103
popcount7 (Wikipedia- naive -128b):	1774	1882	1768	1734	1739	1762	1862	1727	2081	1982	1772	1813
popcount8 (asm SSE3 - pshufb 128b):	586	579	507	501	509	564	566	526	623	554	513	532
popcount9 (asm SSE4- popcount 32b):	807	836	788	820	834	857	898	842	1063	820	830	826
popcount10(asm SSE4- popcount128b):	417	423	404	412	410	440	526	447	917	405	409	415

POPCOUNT:	-O0	-Og	-O1	-O2
pcnt1	103570	29170	**17771**	16973
pcnt2	56252	**10589**	17717	11251
pcnt3	16397	16918	**13589**	13336
pcnt4	14594	13361	**14673**	13131
pcnt5	26813	7254	7578	**5856**
pcnt6	11152	3653	3678	**3103**
pcnt7	6629	2085	2028	**1813**
pcnt8	1145	570	740	**532**
pcnt9	2298	633	638	**826**
pcnt10	1138	**436**	**423**	**415**

Ganancias:	-O0	-Og	-O1	-O2
pcnt1			1,00	
pcnt2		**1,68**		
pcnt3			1,31	
pcnt4			1,21	
pcnt5				**3,03**
pcnt6				**5,73**
pcnt7				**9,80**
pcnt8				**33,38**
pcnt9				21,50
pcnt10		**40,75**	42,01	42,85

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