Adrián Jesús Peña Rodríguez

Diario de trabajo – Práctica 3

Sesión 1.- Primero he realizado una investigación en Internet acerca de los conceptos de la práctica, posteriormente he comprendido que se me pedía y empece a realizar la sección de paridad.
Complete en este día las versiones 1 y 2 de paridad.

Sesión 2.- En este sesión pase a realizar las cinco primeras versiones de PopCount y la investigación de los nuevos conceptos aportados en la sesión. También investigue posibles mejoras de estos métodos, en concreto del quinto.

Trabajo en casa.- Hice en casa las versiones 6 y 7 de PopCount las cuales consistían en entender el código propuesto en el PDF de la práctica.

Sesión 3.- Completo la sección paridad excepto la versión 6 la cual la hice en la biblioteca esa misma tarde.

Trabajo en casa.- Realización del estudio en gráficas de Excel. Y algunos ejercicios de del apéndice 2.

Cuestión 6 – parity.c

La propiedad del XOR en la que se basa es que afecta a la bandera PF (parity flag), la cual indica si en un número en binario hay un número de unos par o no, puesto que lo hace de los 8 bits inferiores.

Cuestión 1 – popcount.c

Podría llegar a ser 32*N donde N corresponde al número de elementos de la lista, si acumulamos la suma en int puede llegara 2147483647, si es en unsigned int puede ser 4294967295

Código popcount:

```
//  según la versión de gcc y opciones de optimización usadas, tal vez haga falta
//  usar gcc –fno-omit-frame-pointer si gcc quitara el marco pila (%ebp)

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

#define TEST  0
#define COPY_PASTE_CALC 1

#if ! TEST
  #define NBITS 20
  #define SIZE (1<<NBITS)
  unsigned lista[SIZE];
  #define RESULT ("N*(N+1)/2 = %d\n", (SIZE-1)*(SIZE/2))
#else
  #define SIZE 4
  unsigned lista[SIZE] = {0x80000000, 0x00100000, 0x00000800, 0x00000001};
  #define RESULT 4

#endif

int pcount_for(int* array, int len)
{
  int result = 0;
  for(int j = 0; j < len; j++){
```

```
    unsigned x = array[j];
    for (int i = 0; i < 8*sizeof(int); i++) {
      unsigned mask = 1 << i;
      result += (x & mask) != 0;
    }
  }
  return result;
}

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

int pcount_asm(int* array, int len) {
  int result = 0;
  for(int i = 0; i < len; i++){
    unsigned x = array[i];
    asm("\n"
    "ini3:              \n\t"
        "shr $1, %[x]   \n\t"        // Desplaza 1 hacia la derecha, esta dentro de un bucle ini3
        "adc $0, %[r]   \n\t"        // Sumamos si hay acarreo en este caso por la derecha
        "cmp $0, %[x]   \n\t"         // Activamos o no el ZF para saltar o no
        "ja ini3        \n\t"
        :[r]"+r" (result)            // Indica que r es de lectura-escritura desde result, r referencia a result en c
        :[x]"r" (x)     );           // x referencia a x en c y es de lectura solo
  }
  return result;
}

int pcount_4(int* array, int len) {
  int result = 0;
  int val;
  int x;

  for(int j = 0; j < len; j++){
    val = 0;
    x = array[j];
    for(int i = 0; i < 8; i++){
      val += x & 0x01010101L;
      x >>= 1;
    }
    val += (val >> 16);
```

```
    val += (val >> 8);
    val = val & 0xFF;
    result += val;
  }
  return result;
}

// Version SSSE3 (pshufb) web http://wm.ite.pl/articles/sse-popcount.html
int popcount5(unsigned* array, int len){
  int i,
      val,
      result = 0;
  int SSE_mask[] = {0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f};
  int SSE_LUTb[] = {0x02010100, 0x03020201, 0x03020201, 0x04030302};

  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"
        "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 popcount6(int* lista, int longitud){
  unsigned x;
  int val, result = 0;
  for(int i = 0; i < longitud; i++){
```

```
    x = lista[i];
    asm("popcnt %[x], %[x]  \n\t"
      :[val] "=r" (val)
      :[x] "r" (x)
    );
    result += val;
  }
  return result;
}

int popcount7(int* lista, int longitud){
  unsigned x1, x2;
  int val, result = 0;
  if(longitud & 0x1)
    printf("Leer 6b y longitud impar?\n");
  for(int i = 0; i < longitud; i+=2){
    x1 = lista[i];
    x2 = lista[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;
}

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

    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 COPY_PASTE_CALC
      printf("%ld\n", tv_usecs);
    #else
      printf("resultado = %d\t", resultado);
      printf("%s:%9ld us\n", msg, tv_usecs);
    #endif
}
```

Adrián Jesús Peña Rodríguez

```
int main()
{
  #if ! TEST
    for(int i = 0; i < SIZE; i++)
      lista[i] = i;
  #endif

  crono(pcount_for, "popcount1 (lenguaje C -)");
  crono(pcount_while, "popcount2 (lenguaje C -)");
  crono(pcount_asm, "popcount3 (lenguaje C - ASM)");
  crono(pcount_4, "popcount4 (lenguaje C -)");
  crono(popcount5, "popcount5 (lenguaje C - ASM)");
  crono(popcount6, "popcount6 (lenguaje C - ASM)");
  crono(popcount7, "popcount7 (lenguaje C - ASM)");

  #if ! COPY_PASTE_CALC
    printf("Calculado = %d\n", RESULT);
  #endif
    exit(0);
}
```

Código paridad:

```
//  según la versión de gcc y opciones de optimización usadas, tal vez haga falta
//  usar gcc –fno-omit-frame-pointer si gcc quitara el marco pila (%ebp)

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

#define TEST  0
#define COPY_PASTE_CALC 1

#if ! TEST
  #define NBITS 20
  #define SIZE (1<<NBITS)
  unsigned lista[SIZE];
  #define RESULT ("N*(N+1)/2 = %d\n", (SIZE-1)*(SIZE/2))
#else
  #define SIZE 4
  unsigned lista[SIZE] = {0x80000000, 0x00100000, 0x00000800, 0x00000001};
  #define RESULT 4

#endif

int paridad_for(int* lista, int longitud){
  int aux, bit, result = 0;

  for(int i = 0; i < longitud; i++){
    aux = lista[i];
```

```
    bit = 0;
    for(int j = 0; j < 32; j++){
      bit = bit^( aux & 0x1);
      aux >>= 1;
    }
    result += bit;
  }
  return result;
}

int paridad_while(int* lista, int longitud){
  int result = 0, bit;
  for(int i = 0; i < longitud; i++){
    unsigned int x = lista[i];
    bit = 0;
    while(x){
      bit = bit^(x & 0x1);
      x >>= 1;
    }
    result += bit;
  }
  return result;
}

int parity3(int* lista, int longitud){
  int val = 0, result = 0, x;

  for(int i = 0; i < longitud; i++){
    val = 0;
    x = lista[i];
    while(x){
      val ^= x;
      x >>= 1;
    }
    result += val & 0x1;
  }
  return result;
}

int parity4(int* lista, int longitud){
  int x, val, result = 0;
  for (int i = 0; i < longitud; i++) {
    x = lista[i];
    val = 0;
    asm("\n"
    "ini3:              \n\t"   // Seguir mientras que x != 0
        "xor %[x], %[v] \n\t"   // Realmente sólo nos interea LSB
        "shr $1,   %[x] \n\t"
        "cmp $0,   %[x] \n\t"
        "ja  ini3       \n\t"
```

```
        :[v]"+r" (val)        // e/s: entrada 0, salidad paridad elemento
        :[x]"r"  (x)          // entrada: valor elemento
    );
    result += val & 0x1;
  }
  return result;
}

int parity5(int* lista, int longitud){
  int result = 0;
  int x;

  for(int j = 0; j < longitud; j++){
    x = lista[j];
    for(int i = 16; i > 1; i /= 2){
      x ^= x >> i;
    }
    result += x & 0x1;
  }
  return result;
}

int parity6(int* lista, int longitud){
  int x, result = 0;
  for (int i = 0; i < longitud; i++) {
    x = lista[i];;
    asm(
        "mov  %[x], %%edx        \n\t"   // sacar copia para XOR. Controlar el registro...
        "shr  $16, %[x]          \n\t"
        "xor  %[x], %%edx        \n\t"
        "xor  %%dh, %%dl         \n\t"
        "setpo %%dl              \n\t"
        "movzx  %%dl, %[x]       \n\t"
        :[x]"+r" (x)            // e/s: entrada valor elemento, salidad paridad
        :
        :"edx"                  // Clobber
    );
    result += x;
  }
  return result;
}

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

  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 COPY_PASTE_CALC
    printf("%ld\n", tv_usecs);
  #else
    printf("resultado = %d\t", resultado);
    printf("%s:%9ld us\n", msg, tv_usecs);
  #endif
}

int main()
{
    #if ! TEST
      for(int i = 0; i < SIZE; i++)
        lista[i] = i;
    #endif
    crono(paridad_for, "paridad con for (lenguaje C)");
    crono(paridad_while, "paridad con while (lenguaje C)");
    crono(parity3, "paridad 3 (lenguaje C)");
    crono(parity4, "paridad 4 (lenguaje C - ASM)");
    crono(parity5, "paridad 5 (lenguaje C )");
    crono(parity6, "paridad 6 (lenguaje C - ASM)");

    #if ! COPY_PASTE_CALC
      printf("Calculado = %d\n", RESULT);
    #endif

    exit(0);
}
```

Adrián Jesús Peña Rodríguez

Optimización O0	0	1	2	3	4	5	6	7	8	9	10	Media (1 a 10)
Popcount1	141644	111161	111404	158515	111396	111109	131672	156235	144627	111307	111209	124500,930843525
Popcount2	62684	55834	55862	63113	55894	62526	62612	55997	60830	56315	55867	58402,0306235086
Popcount3	19690	16848	16858	16930	16875	16860	16865	16874	16927	16996	16862	16889,4415759457
Popcount4	28913	28833	28737	29626	29658	28730	28780	28793	28807	29637	28768	29034,2185443173
Popcount5	1253	1229	1237	1235	1235	1240	1250	1236	1249	1248	1255	1241,3739901892
Popcount6	3717	3702	3708	3751	3697	3694	3701	3800	3688	3750	3703	3719,2452065885
Popcount7	2482	2486	2478	2504	2478	2481	2487	2472	2488	2495	2480	2484,8842739183

Optimización O1	0	1	2	3	4	5	6	7	8	9	10	
Popcount1	70973	78665	45256	44758	44463	44676	44515	76367	71053	77004	78502	58408,8958225636
Popcount2	18741	25162	16027	16278	16026	16222	16169	21916	20444	21846	22205	18958,1563511497
Popcount3	25681	27897	25804	24402	24000	23994	23992	28992	27445	29114	29464	26419,1311135409
Popcount4	10626	11364	10820	10684	10587	10572	10620	11808	11231	12170	11972	11167,6013041605
Popcount5	607	632	619	618	648	642	604	709	635	692	657	644,857061513
Popcount6	866	921	864	892	864	866	867	980	916	958	939	905,8038473106
Popcount7	1321	1384	1334	1321	1338	1291	1302	1432	1364	1458	1424	1363,7077162872

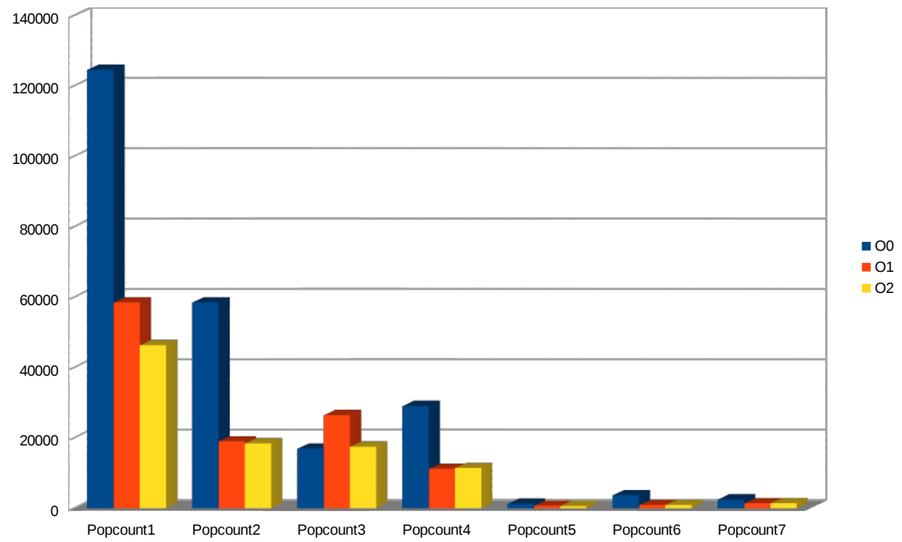
Optimización O2	0	1	2	3	4	5	6	7	8	9	10	
Popcount1	72744	38512	42752	65778	67084	67274	46698	37544	37494	37598	37498	46302,8029625144
Popcount2	21695	16419	16552	23007	23126	23348	18193	16419	16428	16415	16572	18424,0833934696
Popcount3	19331	17937	15994	20400	20457	20495	16634	16001	16000	16013	15992	17488,2608506087
Popcount4	12240	10756	10636	12726	12865	12856	10708	10688	10738	10799	11382	11378,2822454481
Popcount5	624	605	566	640	643	728	599	609	598	577	591	614,1299427327
Popcount6	954	876	877	1024	993	998	878	879	872	896	880	915,5190813973
Popcount7	1488	1365	1376	1522	1522	1552	1343	1357	1311	1323	1363	1400,8080728219

Nota: vemos que el popcount 4 es muy ineficaz sin optimización, esto se debe a que todo es con acceso a memoria y al no tener optimización el uso de variables nos provoca que sea más ineficaz que popcount3 por ejemplo

Comando usado para la medición: for((i=0;i<11;i++));do echo $i; ./popcount ;done |pr -11 -l 20 -

Procesador usado = Intel Core I3-3210M

Optimizaciones	O0	O1	O2
Popcount1	124500,930843525	58408,895823	46302,802963
Popcount2	58402,0306235086	18958,156351	18424,083393
Popcount3	16889,4415759457	26419,131114	17488,260851
Popcount4	29034,2185443173	11167,601304	11378,282245
Popcount5	1241,3739901892	644,85706151	614,12994273
Popcount6	3719,2452065885	905,80384731	915,5190814
Popcount7	2484,8842739183	1363,7077163	1400,8080728



Adrián Jesús Peña Rodríguez

Optimización O0	0	1	2	3	4	5	6	7	8	9	10	**Media (1 a 10)**
Paridad1	93877	88067	81280	86531	85554	86019	81381	86115	80557	85371	82418	84291,5593503584
Paridad2	44971	44844	43246	44536	44880	44601	40842	45416	42266	44419	42738	43756,7335475732
Paridad3	40739	43270	45363	43494	42844	45257	43341	47896	40583	44728	40562	43683,1044148717
Paridad4	10468	11382	13723	11835	12254	11453	10779	11499	13148	11633	13436	12078,397018295
Paridad5	10352	10182	9907	9983	9744	10534	9247	10148	9734	12253	9721	10118,1076355597
Paridad6	4045	2837	2726	2846	3060	2732	2842	2928	3054	2934	3289	2920,3912370439

Optimización O1	0	1	2	3	4	5	6	7	8	9	10	
Paridad1	28054	22059	22504	23632	22904	21626	20149	21684	21867	22127	22952	22131,7559376683
Paridad2	14847	13896	13402	17153	13226	13424	13026	12568	13116	13431	13049	13580,8646281764
Paridad3	8804	8430	8109	9032	11453	11739	11417	8209	11588	7749	7729	9402,3169674379
Paridad4	7475	7307	7498	7481	7751	7215	7065	7745	7017	9039	7080	7500,111121508
Paridad5	5082	4328	4285	4997	4299	4448	4216	4297	4343	4289	4349	4380,3426981541
Paridad6	995	1048	914	1017	869	1030	877	964	867	970	895	942,7956991248

Optimización O2	0	1	2	3	4	5	6	7	8	9	10	
Paridad1	30528	23439	25272	22473	23210	23656	24486	23797	23424	25539	23418	23854,5429562034
Paridad2	10235	8263	8600	10398	8525	8545	9086	8830	9575	9209	10672	9138,8657588627
Paridad3	8497	7473	7738	8946	8001	7875	7767	8032	8114	7948	7728	7953,8691384744
Paridad4	8214	7725	7853	10604	8254	8286	7814	8124	8300	8166	8427	8323,0345636518
Paridad5	4600	3301	3664	4071	3556	3812	3426	3532	3734	3457	3618	3611,2429293022
Paridad6	1057	899	895	908	929	899	852	832	954	894	970	902,3321946743

Procesador usado = Intel Core I3-3210M

Comando usado para la medición: for((i=0;i<11;i++));do echo $i; ./paridad ;done |pr -11 -l 20 -w

Nota: podemos apreciar que en O1 y en O2 la paridad 1 y 4 presentan una anomalía, posiblemente se deba a algún pico en el rendimiento del ordenador en ese momento ya que el margen es muy pequeño

Optimizaciones	O0	O1	O2
Paridad1	84291,55935	22131,755938	23854,5429562034
Paridad2	43756,733548	13580,864628	9138,8657588627
Paridad3	43683,104415	9402,3169674	7953,8691384743
Paridad4	12078,397018	7500,1111215	8323,0345636518
Paridad5	10118,107636	4380,3426982	3611,2429293022
Paridad6	2920,391237	942,79569912	902,3321946743