Práctica 3

Alejandro Poyatos López

2ºB(2)

En la primera clase de prácticas correspondiente a esta sesión estuve haciendo los ejercicios de las sumas correspondientes al tutorial, durante el resto de las clases he estado trabajando con el código de parity.c y popcount.c, y finalmente los he compilado con distintos ordenes de optimización para recopilar los datos que plasmaré en la gráfica de esta práctica.

Codigo parity.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define WSIZE 8*sizeof(unsigned)
#define SIZE 1<<20

unsigned lista[SIZE];
int resultado = 0;



int paridad1(int* array, int len){

        int resultado = 0;
        int x;
```

```
	int i = 0;
	unsigned val = 0;

	for (i = 0; i < len; i++){

		x = array[i];
		do{
			val ^= (x & 0x1);
		}while(x >>= 1);

		resultado += val;
		val = 0;
	}

	return resultado;
}



int paridad2(int* array, int len){

	int resultado = 0;
	int i;
	int j;
	unsigned x;
	unsigned val = 0;

	for(i = 0; i < len; i++){

		x = array[i];
		for(j = 0; j < WSIZE; j++){
			unsigned mask = 0x1 << j;
			val ^= (x & mask) != 0;
```

```
                }
                resultado += val;
                val = 0;
        }

        return resultado;
}



int paridad3(int* array, int len){

        int i;
        int x;
        int resultado = 0;
        int val = 0;

        for (i = 0; i < len; i++){

                x = array[i];
                while(x){
                        val ^= x;
                        x >>= 1;
                }
                resultado += (val & 0x1);
        }
        return resultado;
}



int paridad4(int* array, int len){
```

```
	unsigned result = 0, i, val, x;
	for (i = 0; i < len; i++){
		x = array[i];
		val = 0;
		asm("\n"

		"bucle:                                         \n\t"

			"xor %[x], %[val]           \n\t"     //val ^= x

			"shr $1, %[x]                       \n\t"

			"jnz bucle                          \n\t"

			: [val] "+r" (val)

			: [x] "r" (x)
		);
		result += (val & 1);
	}
	return result;
}



int paridad5(unsigned* array, int len){
	int i, j, x;
	int result = 0;
	for(i = 0; i < len; i++){
		x = array[i];
		for(j = 16; j >= 1; j = j / 2){
			x ^= x >> j;
		}
```

```
                result += (x & 0x1);
        }
        return result;
}



int paridad6(unsigned* array, int len){

        int i;
        unsigned x;
        int result = 0;

        for (i = 0; i < len; i++){
                x=array[i];

                asm("\n"

                        "mov %[x], %%edx        \n\t" //sacar copia para XOR.  Controlar el registro…

                        "shr $16, %[x]          \n\t"   //x >>= 1

                        "xor %[x], %%edx\n\t"

                        "xor %%dh, %%dl         \n\t"

                        "setpo %%dl             \n\t"

                        "movzx %%dl, %[x]       \n\t"   //devolver en 32 bits

                        : [x] "+r" (x)                  //Entra valor elemento, sale paridad

                        :
```

```
                                : "edx"

                                );

                        result += x;
                }
                return result;
}

void crono(int (*func)(), char* msg){                                   //Funcion usada para medir los
tiempos

                struct timeval tv1,tv2;

                long          tv_usecs;

                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);

                printf("resultado = %d\t", resultado);

                printf("%s:%9ld us\n", msg, tv_usecs);
```

```
}



int main(){

        int i;

        for (i = 0; i < SIZE; i++)

         lista[i] = i;



        crono(paridad1, "paridad1");

        crono(paridad2, "paridad2");

        crono(paridad3, "paridad3");

        crono(paridad4, "paridad4");

        crono(paridad5, "paridad5");

        crono(paridad6, "paridad6");


        exit(0);

}
```

Código popcount.c:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define WSIZE 8*sizeof(unsigned)
#define SIZE 1<<20
unsigned lista[SIZE];
int resultado = 0;
int popcount1(int* array, int len){
	int resultado = 0;
	int i = 0;
	int x;

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


int popcount2(int* array, int len){
	int resultado = 0;

	int i, j;
	unsigned x;

	for(i = 0; i < len; i++){
		x = array[i];
```

```
                for(j = 0; j < WSIZE; j++){
                        unsigned mask = 1 << j;
                        if((x & mask)){
                                resultado++;
                        }
                }
        }
        return resultado;
}


int popcount3(int* array, int len){
        int i, x;
        int resultado = 0;

        for (i = 0; i < len; i++){
                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"   //Salto si != 0

                        : [r] "+r" (resultado)      // e/s:  añadir lo acumulado hasta el momento

                        : [x] "r" (x)                   //entrada: valor elemento
```

```
                );
        }

        return resultado;
}



int popcount4(int* array, int len){
        int result = 0, val, i, j;
        unsigned x;

        for (i = 0; i < len; i++){
                val = 0;
                x = array[i];

                for (j = 0;j < 8; j++){
                        val += x & 0x1010101;
                        x >>= 1;
                }

                val += (val >> 16);
                val += (val >> 8);
                result += (val & 0xFF);
        }
        return result;
}



int popcount5(unsigned* array, int len){
        int i, val=0, result=0;
```

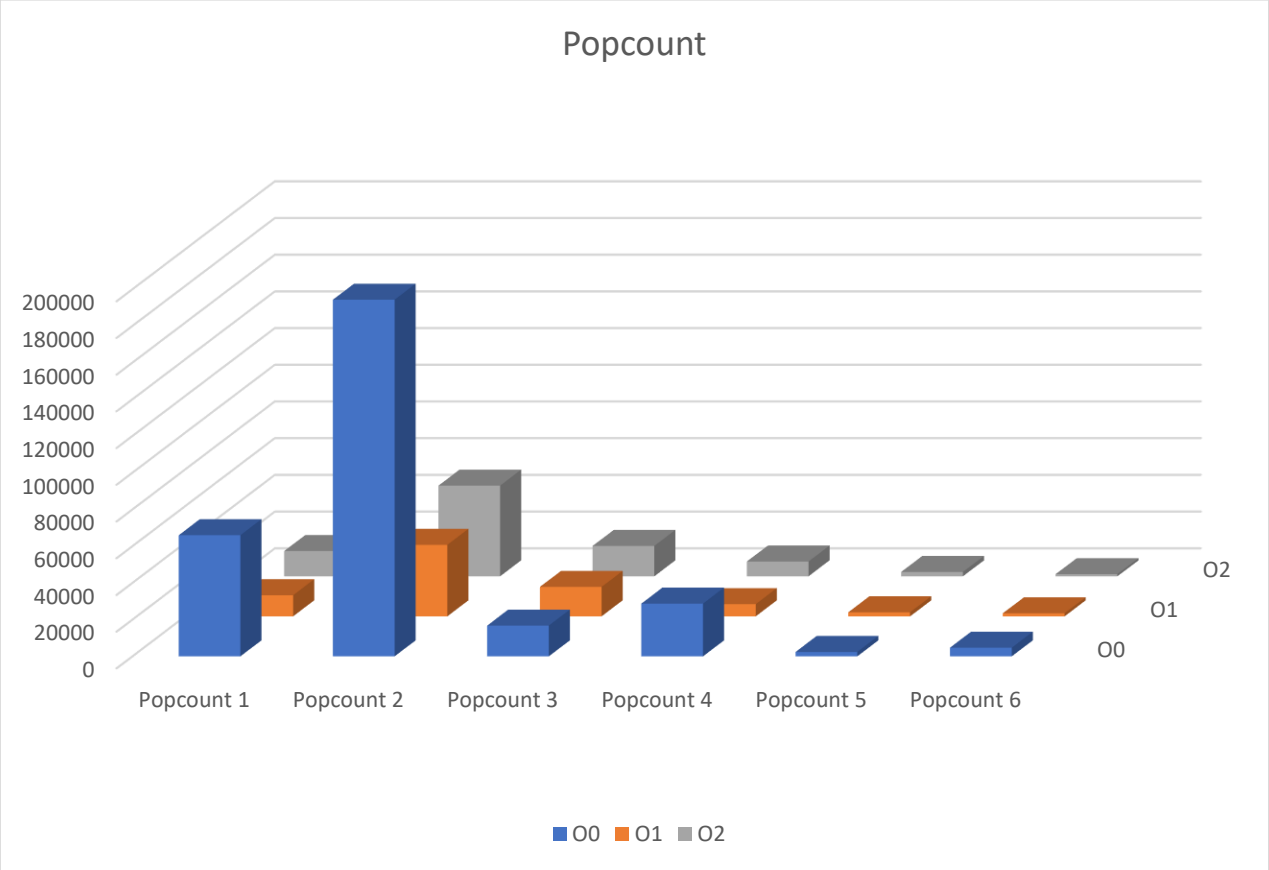
```
int SSE_mask[] = {0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f};
int SSE_LUTb[] = {0x02010100, 0x03020201, 0x03020201, 0x04030302};

if (len & 0x3){
        printf("Leyendo 128 b, pero ¿len no es 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 popcount6(unsigned* array, int len){
        int i;
```

```
        unsigned x;
        int val, result = 0;

        for (i = 0; i < len; i++){
                x = array[i];

                asm(

                        "popcnt %[x], %[val]"

                        :       [val] "=r" (val)

                        :       [x] "r" (x)

                );

                result+=val;
        }

        return result;
}

void crono(int (*func)(), char* msg){

   struct timeval tv1,tv2;

   long         tv_usecs;

   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);

    printf("resultado = %d\t", resultado);

    printf("%s:%9ld us\n", msg, tv_usecs);

}



int main(){

            for (int i = 0; i < SIZE; i++)
                        lista[i] = i;



            crono(popcount1, "popcount1");

            crono(popcount2, "popcount2");

            crono(popcount3, "popcount3");

            crono(popcount4, "popcount4");

            crono(popcount5, "popcount5");

            crono(popcount6, "popcount6");

            exit(0);
}
```

En cuanto las gráficas se han realizado en Excel y los programas se han ejecutado en la terminal de mi Ubuntu 16.04 ejecutándose sobre una máquina virtual, en cuanto a las prestaciones del equipo se ha realizado sobre un Intel Core i7 7700HQ:

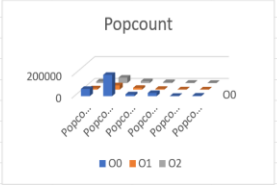


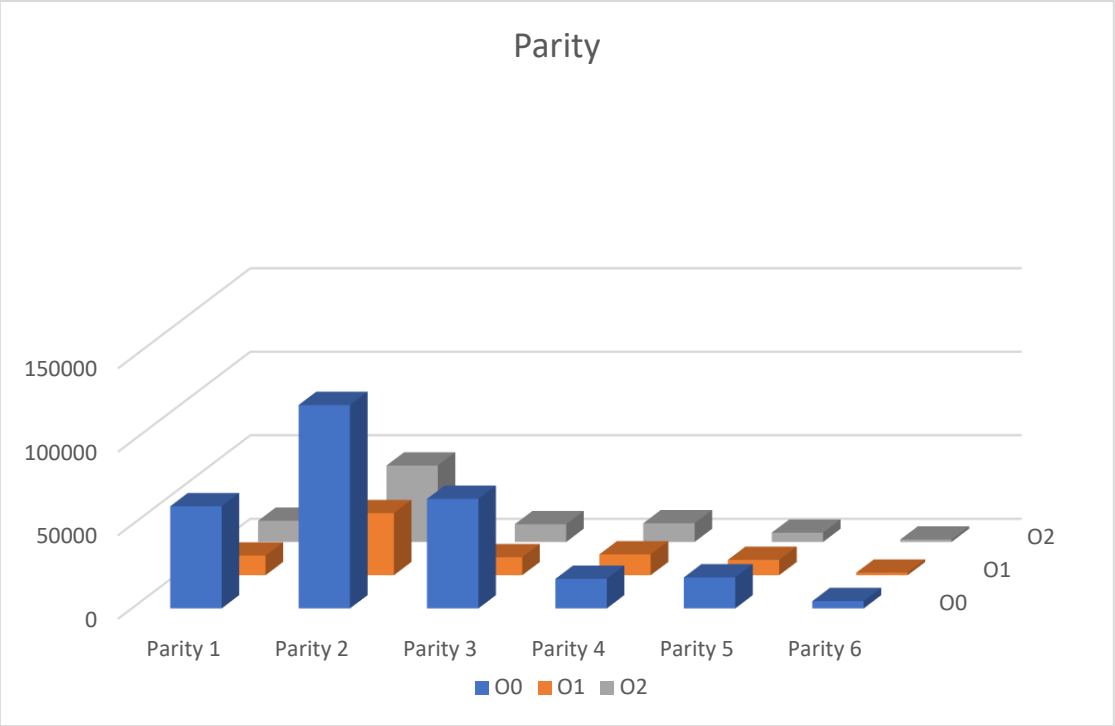
Con optimizacion -O0	0	1	2	3	4	5	6	7	8	9	10	Media
Popcount 1	82189	72082	72619	74184	55385	68202	58179	56797	68682	54772	62423	65955,8182
Popcount 2	216521	201560	208790	176030	178834	215353	165869	166286	206317	201392	200448	194309,091
Popcount 3	15886	20908	15337	21102	11804	21437	13080	14252	17308	15998	16865	16725,1818
Popcount 4	31403	26461	30673	30115	26963	35406	25581	25844	27008	27719	28735	28718,9091
Popcount 5	1464	1843	2343	1619	2112	1311	2753	3917	2681	4379	1643	2369,54545
Popcount 6	3945	3540	4992	3898	4757	4348	5745	6934	5202	4336	3881	4688,90909

Con optimizacion -O1	0	1	2	3	4	5	6	7	8	9	10	Media
Popcount 1	10033	9434	12402	14876	10945	12656	11453	9749	12001	9799	12930	11479,8182
Popcount 2	39463	37071	34756	43841	40537	42196	41137	33662	40771	36262	38319	38910,4545
Popcount 3	15183	15889	13956	16515	14710	17885	17872	13433	17451	15952	17296	16012,9091
Popcount 4	6703	6102	5750	7396	5898	7900	6119	7149	6387	5979	7596	6634,45455
Popcount 5	1563	1559	1319	3648	2507	2065	4173	1700	2778	1500	1644	2223,27273
Popcount 6	1430	821	1339	1041	2506	663	2207	1155	3889	1114	1954	1647,18182

Con optimizacion -O2	0	1	2	3	4	5	6	7	8	9	10	Media
Popcount 1	15835	14671	14132	9552	9539	16578	15170	17266	12368	16321	10008	13767,2727
Popcount 2	48280	45376	46908	47347	40993	52948	54816	60562	47003	56800	42016	49368,0909
Popcount 3	17526	16868	14756	16531	15006	17708	17625	18392	14981	17033	15649	16552,2727
Popcount 4	5845	8874	9513	7251	7499	8442	8650	9345	7781	8154	7193	8049,72727
Popcount 5	1554	2602	3084	1824	1455	2634	1553	4003	3894	1824	1774	2381,90909
Popcount 6	1135	645	694	2022	850	1056	1162	1110	2229	1155	777	1166,81818

POPCOUNT	O0	O1	O2
Popcount 1	65955,8182	11479,8182	13767,2727
Popcount 2	194309,091	38910,4545	49368,0909
Popcount 3	16725,1818	16012,9091	16552,2727
Popcount 4	28718,9091	6634,45455	8049,72727
Popcount 5	2369,54545	2223,27273	2381,90909
Popcount 6	4688,90909	1647,18182	1166,81818





Con optimizacion -O0	0	1	2	3	4	5	6	7	8	9	10	Media
Parity 1	78394	65543	63728	58730	59601	55145	63497	60719	53645	62674	51387	61187,545
Parity 2	133730	134865	132709	125075	121334	125034	109004	136427	97001	122620	102107	121809,63
Parity 3	82038	70015	69974	54167	64152	56289	71318	76349	55284	71029	50926	65594,636
Parity 4	19376	14991	16111	13300	22335	17375	19925	19022	17718	18003	16635	17708,272
Parity 5	17931	18132	23389	17492	17761	15845	20421	19863	16943	17772	18833	18580,181
Parity 6	2918	4957	4937	3473	4449	3113	5250	4408	5313	5221	4020	436

Con optimizacion -O1	0	1	2	3	4	5	6	7	8	9	10	Media
Parity 1	14451	15300	10015	9799	10772	10185	15445	11865	9892	11255	10070	11731,727
Parity 2	46237	37062	29505	35545	38476	34459	37716	37830	38447	38051	34949	37116,090
Parity 3	11052	12399	8892	7923	11326	10751	9641	8853	10478	12584	14149	10731,636
Parity 4	15489	12556	15284	7648	12346	10751	11373	9559	14888	13820	13431	12467,727
Parity 5	10546	7927	8277	11661	10386	12785	7475	8452	7912	8120	7076	914
Parity 6	1839	1011	1257	2346	1067	2464	1052	2395	1031	1184	1059	1518,6363

Con optimizacion -O2	0	1	2	3	4	5	6	7	8	9	10	Media
Parity 1	9618	11921	12536	15533	13805	13264	10827	15400	9797	12726	14048	12679,545
Parity 2	43972	46897	51175	48974	46863	53851	49834	43044	38661	38915	41465	45786,454
Parity 3	12499	11157	7851	9398	13407	12127	8542	9936	10530	12296	9233	10634,181
Parity 4	11555	11739	10026	8206	11816	14578	9639	10191	11774	11458	12568	11231,818
Parity 5	4787	5487	4263	4221	6192	6901	6461	5404	7345	4998	5301	5578,1818
Parity 6	1638	1289	859	914	1843	1471	1699	976	2232	1029	1781	1430,0909

PARITY	O0	O1	O2
Parity 1	61187,5455	11731,7273	12679,5455
Parity 2	121809,636	37116,0909	45786,4545
Parity 3	65594,6364	10731,6364	10634,1818
Parity 4	17708,2727	12467,7273	11231,8182
Parity 5	18580,1818	9147	5578,18182
Parity 6	4369	1518,63636	1430,09091