CARLOS MORALES AGUILERA

ESTRUCTURA DE COMPUTADORES

PRÁCTICA 3

Algunas preguntas de autocomprobación:

Sesión de depuracion suma_01_S_cdecl

1. Se puede realizar un volcado de la pila, usando Data->Memory->Examine 8 hex words(4B) $esp, en donde se ha escogido 8 por tener margen de sobra(en línea de comandos gdb sería x/8xw $esp). Comprobar que coindice el volcado así obtenido con la figura 3.

Sí coincide porque a partir de $8, en %esp es donde se empiezan a guardar los argumentos de la función

El programa principal no tiene marco de pila(EBP=0) pero el S.O. le deja algo anotado en la pila. Si el 1 indicara int argc=1, y el segundo argumento fuera un array de char argv[]... ¿dónde se volcaría su valor?¿Qué podría ser ese argumento?

Según la convención cdecl, primero se le pasa el vector y después el número de argumentos(orden inverso).

1<qué>bytes<argv[0]>. Primero miramos en la pila de dirección con 8 hex words(4B) $esp la dirección de memoria de argc=1 → 0xffffd020 y la de argv[0] → 0xffffd024 que contiene la dirección 0xffff020e. Eso es un puntero al nombre del programa. Si en data → examine → memory ponemos : 1 string bytes from 0xffffd20e y print, nos da el nombre y ruta en la que se encuentra el ejecutable.

2. El volcado de la pila es útil para ir viendo la pila durante la ejecución del programa, conforme va cambiando ESP. Tras llamar a suma, se puede realizar un volcado de memoria para comprobar que el argumento #2 es nuestra lista de 9 enteros, usando Data → Memory → Examine <cuántos> hex words(4B) <qué>. ¿De dónde sacamos <cuántos> y <qué>?

Con 9 hex words 4B $esp, obtenemos las direcciones de los valores, y después: 1 decimal bytes <direccion>

3. ¿Por qué la función suma preserva ahora %ebx y no hace lo mismo con %edx?

Porque %ebx es salvainvocado, y la función debe guardarlo en pila y restaurarlo antes de retornar si necesitara modificar su contenido.

4.¿Qué modos de direccionamiento usa la instrucción add (%ebx, %edx, 4), %eax?
¿Cómo se llama a cada componente del primer nodo?.El último componente se llama escala, ¿qué sucedería si lo eliminásemos?

Modo de direccionamiento indexado
(%base, %indice, escala), destino
Se pondría por defecto 1

7. La instrucción **jne** en el programa original se podría cambiar por alguno de entre otros tres saltos concidionales (uno de ellos es sencillamente mnemotécnico para el mismo código de operación) y el programa seguiría funcionando igual. ¿Cuáles son esos 3 mnemotécnicos? ¿Qué tendría que suceder para que se notaran diferencias con el original?

Instrucciones js y jl (sin signo)

8. Según la figura si hubiésemos necesitado añadir una variable loca .int (entero de 4B), a la función suma, hubiéramos restado 4 a ESP....¿cuándo?
A la salida deberíamos sumarle 4 a ESP... ¿cuándo?

SUMARLE 4 A ESP: Cuando se terminara de ejecutar la función, para sacarla de la pila.
RESTARLE 4 A ESP:Dentro de la función suma, antes o después del bucle

9. Si hubiésemos reservado sitio para 3 variables locales .int (enteros de 4B) ¿qué dos instrucciones cambiarían respecto a la pregunta anterior y en qué cambiarían?
¿Cómo se direccionaría la segunda variable local respecto al marco de pila? Por ejemplo, cómo sería la instrucción ensamblador para poner esa variable a 0?

Cambiar el valor que se resta (subbl) y se le suma(addl) a ESP. En vez de 4, sería un 12(4 por variable).

Redireccionar: 4(%esp)
movl $0, 4(%esp) , para inicializarla a 0

4.1 Calcular la suma de bits de una lista de enteros sin signo.

Código del programa(con la implementación de las 7 versiones del popcount).

Nota: ahora mismo el programa está programado de forma que lo que imprime no son los valores de la función, sino los tiempos que tardan las funciones en realizar el cálculo. (Para la lista que he utilizado, el cálculo será 4).

Si lo que se quisiera imprimir fuera los resultados del popcount, bastaría con comentar la línea 214 y descomentar la línea 213, y cambiar el valor de test a 1, (línea 7).

Código del programa:

```
#####################################################################
//  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 ()
#else

/* ---------------------------------------------------------------------------*/
   #define SIZE 4
   unsigned lista[SIZE]={0x80000000, 0x00100000, 0x00000800,0x000000001};

   //#define RESULT 4

/* ---------------------------------------------------------------------------*/
#endif

unsigned resultado=0;

unsigned popcount1(unsigned* array, unsigned len)
{
   unsigned  i,j,   x, res=0;

   for (i=0; i<len; i++){                    // Recorro el array con un bucle for
      x = array[i];
      res+=x;
         for ( j=0; j<8*sizeof(int) ; j++){  // Recorro los bits de cada elemento del array
           unsigned mask=1 << j;             // Aplico la mascara y desplazo a la derecha
           res+=(x & mask) !=0;              // Acumulo el resultado
        }
         }

   return res;
}

unsigned popcount2 (unsigned *array, int len) {
```

```
    unsigned  i, res=0;

    for (i=0; i<len; i++){                  // Recorro el array con un bucle for
        unsigned x = array[i];
        do {
            res += x & 0x1;                 // Aplico la mascara y sumo los bits
            x >>= 1;                        // Desplazo a la derecha
        } while (x);
    }

    return res;
}

unsigned popcount3 (unsigned* array, int len)
{
    unsigned  i,   res=0, x;
    for (i=0; i<len; i++) {                 // Recorro el array con un bucle for

        x = array[i];
        //res += x;

        asm("\n"
    "ini3:          \n\t"


            "shr %[x] \n\t"                 // Desplazamiento a la derecha

            "adc $0, %[r]   \n"             // Acumulacion

            "test %[x], %[x]    \n"         // Comprobacion de fin de bucle

            "jne ini3  \n"

            : [r] "+r" (res)                // Guardar res en un registro
            : [i]  "r" (i),                 // Guardar el indice en un registro
            [x] "r" (x)                     // Guardar x en un registro
        );
    }

    return res;
}


unsigned popcount4 (unsigned *array, int len) {

    int j,i;
    unsigned x,val;
    unsigned res = 0;
```

```
    for ( j = 0; j < len; j++) {                // Recorro el array con un bucle for
        val = 0;
        x = array[j];
        for ( i = 0; i < 8; i++) {

            val += x & 0x01010101;              // Acumulo los bits y aplico la mascara
            x >>= 1;                            // Desplazo a la derecha
        }
        val += (val >> 16);
        val += (val >> 8);
        res += val & 0xFF;
    }



    return res;
}


unsigned popcount5 (unsigned *array, int len) {

    unsigned i;
    unsigned val, result=0;
    unsigned SSE_mask[] = {0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f};
    unsigned SSE_LUTb[] = {0x02010100, 0x03020201, 0x03020201, 0x04030302};

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

    for (i=0; i<len; i+=4){                     // Recorro el array con un bucle for

    asm("movdqu %[x], %%xmm0 \n\t"              // Completamos en ensamblador con los
        "movdqa %%xmm0, %%xmm1 \n\t"                    // registros correspondientes
        "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;                          // Acumulo los bits en el resultado
    }

    return result;
}

unsigned popcount6(unsigned *array, int len){

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

   for(i=0; i<len;i++){                                 // Recorro el array con un bucle for
      x=array[i];
      asm("popcnt %[x], %[val]"
         :[val] "=r" (val)
         :[x] "r" (x)
      );
      result += val;                            // Acumulo en la variable resultado
   }

   return result;
}

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

   if(len & 0x1)
      printf("leer 64b y len impar?\n");

   for(i=0; i<len;i+=2){                        // Recorro el array con un bucle for
      x1=array[i]; x2=array[i+1];

      asm("popcnt %[x1], %[val] \n\t"
         "popcnt %[x2], %%edi \n\t"
         "add  %%edi, %[val] \n\t"
          // Guardo los datos en registros
```

```
        : [val]"=&r" (val)
        : [x1]   "r" (x1),
          [x2]  "r" (x2)
        : "edi");

     result +=val;
   }
  return result;
}



void crono(unsigned (*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);
    // printf("resultado = %d\n", resultado);
    printf("%ld\n", tv_usecs);
}

int main()
{
    #if ! TEST
        unsigned i;

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

    crono(popcount1, "popcount1 (lenguaje C -       for) ");
    crono(popcount2, "popcount2 (lenguaje C -       for) ");
    crono(popcount3, "popcount3 (lenguaje C -       for) ");
    crono(popcount4, "popcount4 (lenguaje C -       for) ");
    crono(popcount5, "popcount5 (lenguaje C -       for) ");
    crono(popcount6, "popcount6 (lenguaje C -       for) ");
    crono(popcount7, "popcount7 (lenguaje C -       for) ");



    exit(0);
}
######################################################################
```

Una vez ejecutadas las funciones para diferentes optimizaciones(O0, O1, O2), los resultados obtenidos por la terminal los paso a la hoja de cálculo, para poder obtener una gráfica que permita visualizar y comparar los resultados obtenidos.

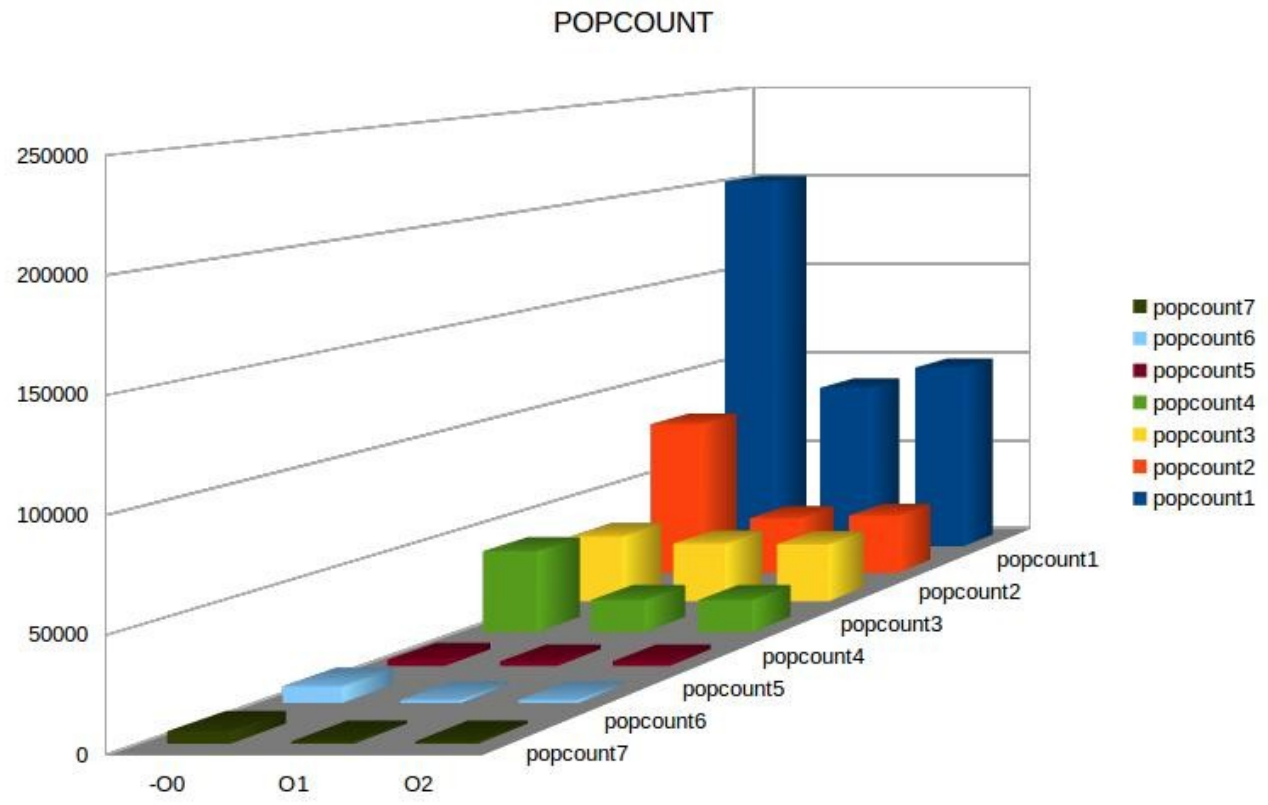
Resultados obtenidos:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Optimizacion -O0		0	1	2	3	4	5	6	7	8	9	10
2	popcount1		201256	201120	201060	201130	201793	201169	201035	201181	201171	201091	201088
3	popcount2		78783	78842	78905	78784	78952	78927	78780	78801	78790	78897	78876
4	popcount3		33114	33183	33105	33134	33170	33152	33132	33147	33257	33152	33117
5	popcount4		39549	39531	39757	39511	39565	39663	39602	39461	39481	39601	39541
6	popcount5		3109	3135	3143	3163	3109	3085	3110	3121	3085	3107	3108
7	popcount6		8891	7523	7587	7578	7565	7591	7561	7603	7542	7568	7541
8	popcount7		5523	5562	5500	5557	5497	5542	5602	5496	5577	5521	5654
9													
10													
11	Optimizacion -O1		0	1	2	3	4	5	6	7	8	9	10
12	popcount1		88983	87584	87538	87628	87594	87531	87534	87339	87345	87345	87398
13	popcount2		29063	29088	29087	29045	29073	29063	29030	28994	28990	28991	29035
14	popcount3		29456	29437	29468	29470	29455	29514	29536	29490	29439	29423	29438
15	popcount4		15920	15830	15853	15917	15863	15828	15872	15826	15813	15843	15828
16	popcount5		2341	2314	2280	2303	2277	2310	2277	2310	2311	2282	2286
17	popcount6		1455	1457	1456	1454	1488	1473	1519	1480	1476	1526	1480
18	popcount7		1341	1308	1297	1330	1275	1278	1315	1288	1316	1316	1316
19													
20													
21	Optimizacion -O2		0	1	2	3	4	5	6	7	8	9	10
22	popcount1		100712	98423	98459	98488	98379	98535	98599	98455	98490	98393	98434
23	popcount2		30419	30417	30351	30339	30392	30315	30327	30282	30296	30344	30386
24	popcount3		29056	29076	29070	29053	29087	29094	29145	29118	29073	29122	29142
25	popcount4		15834	15867	15884	15844	15814	15833	15841	15884	15843	15858	15821
26	popcount5		2298	2299	2309	2304	2303	2334	2288	2273	2348	2348	2299
27	popcount6		1449	1451	1499	1453	1450	1452	1448	1489	1450	1448	1496
28	popcount7		1316	1317	1314	1315	1314	1317	1334	1418	1341	1335	1367

Hago la media del tiempo que tarda cada función popcount para cada uno de los niveles de optimización, y a partir de de estos datos, obtengo la gráfica.

En forma gráfica:

30				
31	POPCOUNT	-O0	O1	O2
32	popcount1	201190,3636	87619,90909	98669,72727
33	popcount2	78848,81818	29041,72727	30351,63636
34	popcount3	33151,18182	29466	29094,18182
35	popcount4	39569,27273	15853,90909	15847,54545
36	popcount5	3115,909091	2299,181818	2309,363636
37	popcount6	7686,363636	1478,545455	1462,272727
38	popcount7	5548,272727	1307,272727	1335,272727
39				



Como podemos ver, a medida que hemos ido desarrollando versiones , cada vez hemos obtenido un resultado mejor en general, y de forma paralela, a medida que aumentamos la optimización, el resultado también mejora.

4.2 Calcular la suma de paridades de una lista de enteros sin signo

Código del programa

En este caso, el programa de este pdf está programado de forma que la salida del programa sí son los valores obtenidos al hacer los cálculos de paridad. Si quisieramos obtener los tiempos con los que posteriormente realizo la gráfica, sería al contrario que en el ejemplo de popcount; la línea 7, el test se pondría a 0 y comentaría la línea 151, descomentando la 152.

Código del programa:

```
#########################################################################
//  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    1
#define COPY_PASTE_CALC 1

#if ! TEST
   #define NBITS 20
   #define SIZE (1<<NBITS)
   unsigned lista[SIZE];
   //#define RESULT ()
#else

/* ----------------------------------------------------------------------------*/
   #define SIZE 4
   unsigned lista[SIZE]={0x80000000, 0x00100000, 0x00000800,0x000000001};

   //#define RESULT 4

/* ----------------------------------------------------------------------------*/
#endif

unsigned resultado=0;

unsigned paridad1(unsigned* array, unsigned len)
{
   unsigned i, j, x, res=0;
   unsigned paridad;

   for (i=0; i<len; i++){                          // Recorro el array con un bucle for
      x = array[i];
      paridad = 0;
          for ( j=0; j<8*sizeof(int) ; j++){       // Recorro cada bit del array
           unsigned mask = (1 << j);                        // Aplico la mascara
           paridad ^= ((x & mask) != 0);           // Obtengo la paridad
      }

      res += paridad;                              // Acumulo el valor de la paridad
         }

   return res;
}

unsigned paridad2 (unsigned *array, unsigned len) {

   unsigned  i, x, res=0;
   unsigned paridad;

   for (i=0; i<len; i++){                          // Recorro el array con un bucle for
      paridad=0;
      x = array[i];
```

```
      do {
         paridad ^= x & 0x1;                    // Obtengo el bit de paridad
         x >>= 1;                               // Desplazo hacia la derecha
         res+=paridad;                          // Acumulo el valor de la paridad
      } while (x);

   }

   return res;
}

unsigned paridad3 (unsigned* array, int len)
{
   unsigned i, x, res=0, paridad;

   for (i=0; i<len; i++){                       // Recorro el array con un bucle for
      paridad=0;
      x=array[i];

      do {
         paridad ^= x;                          // Obtengo el bit de paridad
         x >>= 1;                               // Desplazo hacia la derecha
      } while (x);

        res+=paridad & 0x1;                     // Acumulo en el resultado aplicando
la máscara
   }

  return res;
}


unsigned paridad4 (unsigned *array, unsigned len) {

   unsigned i, x, res=0, paridad;

   for (i=0; i<len; i++)                        // Recorro el array con un bucle for
   {
        paridad=0;
        x=array[i];

   asm("\n"
      "ini3:                    \n\t"
      "xor %[x], %[v]               \n"         // Calculo del bit de paridad
      "shr %[x]                 \n\t"           // Desplazo hacia la derecha
      "test %[x], %[x]          \n"             // Comprobacion de fin de bucle
      "jne ini3          \n"
```

```
        // Guardo las variables en los registros
      : [v] "+r" (paridad)
      : [i] "r" (i),
        [x] "r" (x)
   );

      res += paridad & 1;                    // Acumulo  el valor del bit de paridad con las mascara
   }

   return res;
}

unsigned paridad5 (unsigned *array, unsigned len) {
        int i, j, res=0;
        unsigned paridad;

        for(i=0; i<len;i++){                 // Recorro el array con un bucle for
           paridad=array[i];
           for(j=16; j>=1; j/=2)
                paridad ^= paridad >> j;     // Aplico desplazamiento y guardo el valor del bit

           res+=(paridad & 0x01);            // Acumulo el bit de paridad aplicandole la mascara
        }

        return res;

}

unsigned paridad6(unsigned *array, int len){

  int i, j, res=0;
  unsigned x;

  for(i=0; i<len;i++){                                  // Recorro el array con un bucle for
    x=array[i];
    asm("mov %[x], %%edx       \n"
      "shr $16, %%edx          \n"                      // Desplazamiento a la derecha
      "xor %[x], %%edx     \n"
      "xor %%dh, %%dl          \n"
      "setpo %%dl          \n"

      "movzbl %%dl, %[x] \n"                           // Extender los bit a long

        // Guardo las variables en los registros
      : [x] "+r" (x)
```

```
        :
        : "edx"
    );
      res += x;                                    // Acumulo en resultado
   }

  return res;
}



void crono(unsigned (*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);
   printf("resultado = %d\n", resultado);
   //printf("%u\n", resultado);
}

int main()
{
   #if ! TEST
      unsigned i;

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

   crono(paridad1, "paridad1 (lenguaje C -      for) ");
   crono(paridad2, "paridad2 (lenguaje C -      for) ");
   crono(paridad3, "paridad3 (lenguaje C -      for) ");
   crono(paridad4, "paridad4 (lenguaje C -      for) ");
   crono(paridad5, "paridad5 (lenguaje C -      for) ");
   crono(paridad6, "paridad6 (lenguaje C -      for) ");



   exit(0);
}
```

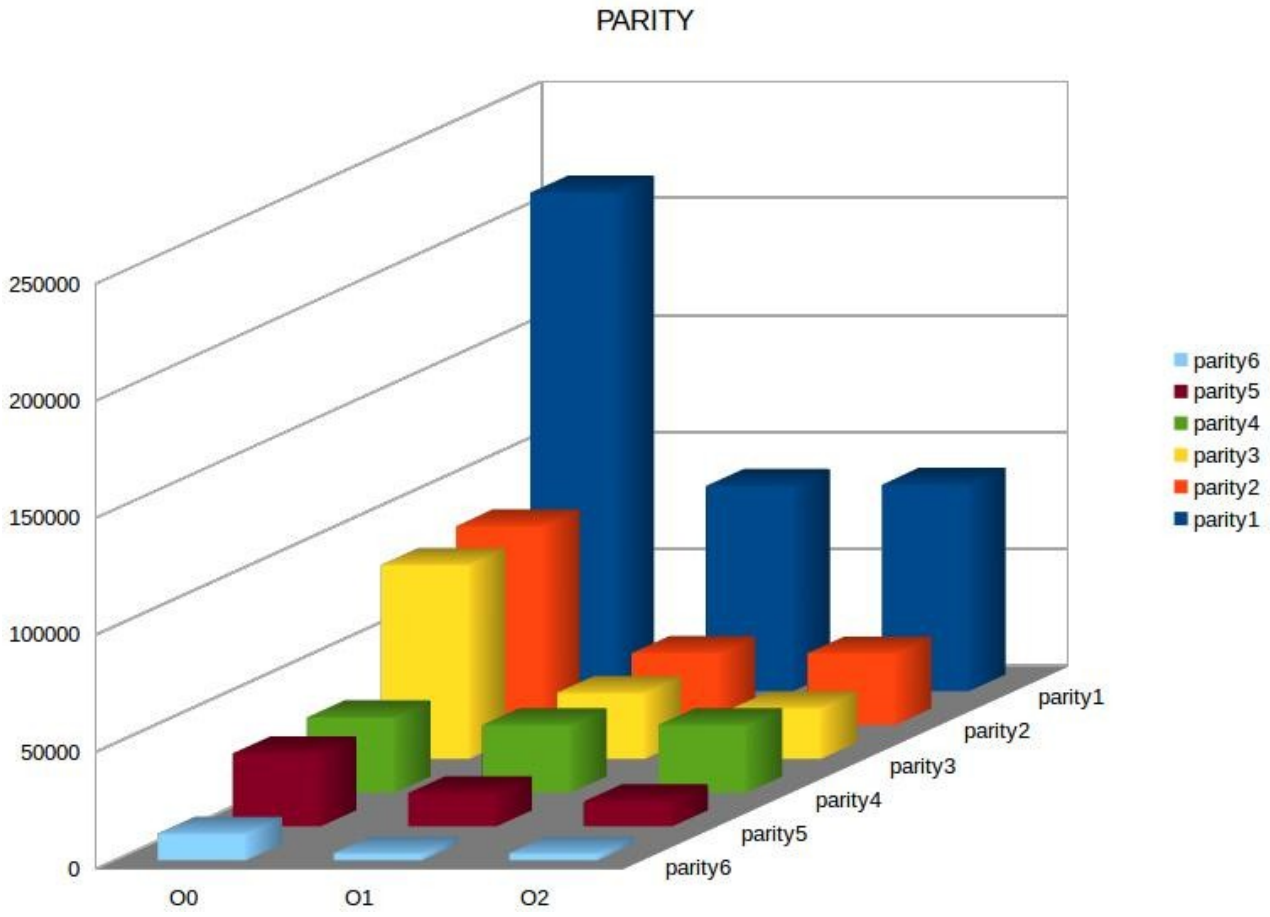
Los resultados obtenidos en la terminal, los paso a la hoja de cálculo, y a partir de ahí obtengo la gráfica, (exactamente igual que con popcount)

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Optimizacion -O0		0	1	2	3	4	5	6	7	8	9	10
2	parity1		223241	216642	210078	210101	213690	210130	211791	213717	215486	210027	210538
3	parity2		84439	84512	87135	84516	84553	87980	84562	84529	84816	84522	84688
4	parity3		82560	82308	85874	84992	82343	82471	82375	82460	82407	83753	82234
5	parity4		31329	31331	31233	32472	35248	31282	31256	31189	31232	34709	34773
6	parity5		31416	31488	31333	31390	31543	31462	31511	31530	31436	31356	31516
7	parity6		11673	11635	11640	11640	11632	11634	11721	11691	11674	11636	11641
8													
9	Optimizacion -O1		0	1	2	3	4	5	6	7	8	9	10
10	parity1		92580	87467	87460	87520	87454	87427	87474	87462	87554	87411	87473
11	parity2		30942	30957	30894	30911	30928	31065	31014	30877	30864	30982	30913
12	parity3		28348	28326	28397	28957	28288	28292	28553	28303	28335	28304	28383
13	parity4		29143	29097	29162	29201	29169	29124	29127	29164	29101	29171	29152
14	parity5		14606	14644	14643	14417	14448	14714	14628	14423	14436	14468	14448
15	parity6		3171	3169	3150	3169	3196	3179	3172	3149	3167	3197	3168
16													
17													
18	Optimizacion -O2		0	1	2	3	4	5	6	7	8	9	10
19	parity1		88235	88039	94578	88201	87669	87705	87656	87650	87674	87678	87716
20	parity2		30938	31069	30803	30833	30844	30801	30818	30825	30820	30807	30834
21	parity3		21796	22802	21821	21798	21775	21808	21825	21796	21800	21796	21783
22	parity4		29228	29153	29091	29087	29142	29065	29113	29195	29065	29084	29163
23	parity5		10754	10781	10750	10757	10753	10745	10748	10761	13632	10806	10873
24	parity6		3169	3219	3201	3149	3173	3149	3168	3150	3255	3151	3151
25													

Hago la media del tiempo que tarda cada función parity para cada uno de los niveles de optimización, y a partir de de estos datos, obtengo la gráfica.

27				
28	PARITY	O0	O1	O2
29	parity1	213221,9091	87934,72727	88436,45455
30	parity2	85113,81818	30940,63636	30853,81818
31	parity3	83070,63636	28407,81818	21890,90909
32	parity4	32368,54545	29146,45455	29126
33	parity5	31452,81818	14534,09091	11032,72727
34	parity6	11656,09091	3171,545455	3175,909091
35				

La gráfica en este caso es :



Podemos observar que como en el caso anterior, a medida que vamos realizando versiones, vamos a ir obteniendo mejores resultados, y paralelamente, a medida que la optimización es mayor, el resultado también mejorará.