Estructura de computadores
Práctica 3
Documentación

Alumno: Antonio David López Machado **Grupo:** C-2

Índice

1. Planificación

Semana 31/10 - 6/11	-Realización del tutorial de la práctica -Pruebas realizadas con las diferentes herramientas que usaremos.
Semana 7/11 - 13/11	-Comprensión del primer ejercicio a realizar -Realización del ejercicio 1 -Comprensión del segundo ejercicio a realizar -Realización del ejercicio 2 -Comprensión del tercer ejercicio a realizar -Realización del ejercicio 3
Semana 14/11 - 20/11	-Mejora en los comentarios de todos los ejercicios de la práctica -Realización de preguntas opcionales -Desarrollo de la planificación
Semana 21/11 - 27/11	-Realización de preguntas opcionales -Desarrollo de la planificación -Finalización de la práctica -Entrega de la práctica

2. Código de los ejercicios

2.1 Ejercicio 1

```
////////////////////////////////////////////////////////////////
/*
*   Antonio David López Machado - Curso 2016/2017
*   Práctica 3 - Ejercicio 1
*/
////////////////////////////////////////////////////////////////

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

#define SIZE (1<<20)        // tamaño suficiente para tiempo apreciable
#define WSIZE 8*sizeof(unsigned)
unsigned lista[SIZE];
//unsigned lista[SIZE]={0x80000000,0x00100000,0x00000800,0x00000001};
//unsigned
lista[SIZE]={0x7fffffff,0xffefffff,0xfffff7ff,0xfffffffe,0x10000024,0x00356700,0x8900ac00,0x00bd00ef};
//unsigned
lista[SIZE]={0x0,0x10204080,0x3590ac06,0x70b0d0e0,0xffffffff,0x12345678,0x9abcdef0,0xcafebeef};
int resultado=0;
```

```
////////////////////////////////////////////////////////////////
/*
*   Método que realizará la suma de los bits de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 1 - Usando un bucle for interno
*
*   \param array --> el conjunto de elementos que sumaremos
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int popCount1(int* array, int len)
{
   int  i,j,  res=0,result;
   for (i=0; i<len; i++){ //Recorremos el array de elementos
      result = 0;
      for (j = 0; j < WSIZE; j++) { //Recorremos los bits de cada elemento
         unsigned mask = 1 << j;          //Creamos la máscara
         result += (array[i] & mask) != 0; //Aplicamos la máscara 0x1
      }
      res+=result;                        //Acumulamos el resultado de cada elemento
   }
   return res;
}

////////////////////////////////////////////////////////////////
/*
*   Método que realizará la suma de los bits de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 2 - Usando un bucle while interno
*
*   \param array --> el conjunto de elementos que sumaremos
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int popCount2(int* array, int len)
{
   int  i,j,  res=0,result;
   for (i=0; i<len; i++){ //Recorremos el array de elementos
      result = 0;
      unsigned value=array[i];
      do{  //Recorremos los bits de cada elemento
         result+= value & 0x1; //Aplicamos la máscara 0x1
         value >>= 1;         //Realizamos el desplazamiento del elemento
      }while(value);
      res+=result;            //Acumulamos el resultado de cada elemento
   }
   return res;
}
```

```
////////////////////////////////////////////////////////////////
/*
*   Método que realizará la suma de los bits de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 3 - Usando un bucle while interno en ensamblador
*
*   \param array --> el conjunto de elementos que sumaremos
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int popCount3(int* array, int len)
{
   int  i,j,  res=0,result;
   for (i=0; i<len; i++){
      result = 0;
      unsigned x=array[i];      // elemento a comprobar
      asm("ini3:          \n\t" //etiqueta
         "shr %[x]       \n\t" // Desplazamiento a la derecha
         "adc $0,%[r]    \n\t" //add carreo -> result
         "cmp $0,%[x]    \n\t" //comp 0:x
         "jne ini3       \n\t" // if !=0 goto ini3
         :  [r]"+r" (result)   // output-input
         :  [x]  "r" (x)  );   // input

      res+=result; //Acumulamos el resultado de cada elemento
   }
   return res;
}
```

```
////////////////////////////////////////////////////////////////
/*
*   Método que realizará la suma de los bits de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 4 - CS:APP Ejercicio 3.49
*
*   \param array --> el conjunto de elementos que sumaremos
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int popCount4(int* array, int len)
{
   int  i,j,  res=0,result,val;
   unsigned x;
   for (i=0; i<len; i++){      //Recorremos cada elemento
      result = 0;
      val=0;
      x=array[i];
      for (j = 0; j < 8;j++) {  //Recorremos cada bit del elemento actual
         val += x & 0x0101010101010101L; //Aplicamos la máscara
         x >>= 1;
      }

      val += (val >> 16);
      val += (val >> 8);
      res+=val & 0xFF; //Acumulamos el resultado de cada elemento
   }
   return res;
}

////////////////////////////////////////////////////////////////
/*
*   Método que realizará la suma de los bits de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 5 - ASM SSE3 - pshufb 128b
*
*   \param array --> el conjunto de elementos que sumaremos
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int popCount5(int* array, int len)
{
   int i;
   int val,result=0;
   int volatile SSE_mask[] = {0x0f0f0f0f,0x0f0f0f0f,0x0f0f0f0f,0x0f0f0f0f};
   int volatile SSE_LUTb[] = {0x02010100,0x03020201,0x03020201,0x04030302};

   for(i=0;i<len;i+=4){
```

```
      asm("movdqu %[x]   ,%%xmm0  \n\t"
         "movdqa %%xmm0 ,%%xmm1  \n\t" //dos copias de x
         "movdqu %[m]   ,%%xmm6  \n\t" //máscara
         "psrlw  $4     ,%%xmm1  \n\t"
         "pand   %%xmm6 ,%%xmm0  \n\t" // xmm0 - nibbles inferiores
         "pand   %%xmm6 ,%%xmm1  \n\t" // xmm1 - nibbles superiores

         "movdqu %[l]   ,%%xmm2  \n\t" // ..como pshufb sobreescribe LUT
         "movdqa %%xmm2 ,%%xmm3  \n\t" // .. queremos 2 copias
         "pshufb %%xmm0 ,%%xmm2  \n\t" // xmm2 = vector popcount inferiores
         "pshufb %%xmm1 ,%%xmm3  \n\t" // xmm3 = vector popcount superiores

         "paddb   %%xmm2, %%xmm3 \n\t" // xmm3 - vector popcount bytes
         "pxor    %%xmm0, %%xmm0 \n\t" // xmm0 = 0,0,0,0
         "psadbw  %%xmm0, %%xmm3 \n\t" // xmm3 = [pcnt bytes 0..7][pcnt bytes8..15]
         "movhlps %%xmm3, %%xmm0 \n\t" // xmm0 = [       0      ][pcnt bytes0..7]
         "paddd   %%xmm3, %%xmm0 \n\t" // xmm0 = [    no usado   ][pcnt bytes0..15]
         "movd    %%xmm0, %[val] \n\t"
         : [val] "=r" (val)
         :   [x] "m" (array[i]),
            [m] "m" (SSE_mask[0]),
            [l] "m" (SSE_LUTb[0])

         );
            result+=val;                  //Acumulamos el resultado de cada elemento
   }
   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);
   printf("resultado = %d\t", resultado);
   printf("%s:%9ld us\n", msg, tv_usecs);
}

int main()
{
   int i;                        // inicializar array
   for (i=0; i<SIZE; i++)     // se queda en cache
            lista[i]=i;
   crono(popCount1, "popCount1 (en lenguaje C    )");
   crono(popCount2, "popCount2 (en lenguaje C    )");
   crono(popCount3, "popCount3 (en lenguaje C    )");
   crono(popCount4, "popCount4 (en lenguaje C    )");
   crono(popCount5, "popCount5 (en lenguaje C    )");
   exit(0);}
```

2.2 Ejercicio 2

```
////////////////////////////////////////////////////////////////
/*
*   Antonio David López Machado - Curso 2016/2017
*   Práctica 3 - Ejercicio 2
*/
////////////////////////////////////////////////////////////////

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

#define SIZE (1<<20)        // tamaño suficiente para tiempo apreciable
#define WSIZE 8*sizeof(unsigned)
unsigned lista[SIZE];
//unsigned lista[SIZE]={0x80000000,0x00100000,0x00000800,0x00000001};
//unsigned
lista[SIZE]={0x7fffffff,0xffefffff,0xfffff7ff,0xfffffffe,0x10000024,0x00356700,0x8900ac00,0x00bd00ef};
//unsigned
lista[SIZE]={0x0,0x10204080,0x3590ac06,0x70b0d0e0,0xffffffff,0x12345678,0x9abcdef0,0xcafebeef};
int resultado=0;

////////////////////////////////////////////////////////////////
/*
*   Método que realiza la suma de las paridades de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 1 - Usando un bucle for interno
*
*   \param array --> el conjunto de elementos al que sumaremos su paridad
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int parityCount1(int* array, int len)
{
   int  i,j,  res=0,result;
   for (i=0; i<len; i++){                  //Recorremos la lista de elementos
      result = 0;
      for (j = 0; j < WSIZE; j++) {           // Recorremos cada bit del elemento actual
         unsigned mask = 1 << j;          //Creamos la máscara
         result ^= (array[i] & mask) != 0;//Aplicamos dicha máscara
      }
      res+=result; //Acumulamos el resultado de cada elemento
   }
   return res;
}
```

```
////////////////////////////////////////////////////////////////
/*
*   Metodo que realizara la suma de las paridades de un conjunto de elementos
*   pasados como parametros.
*
*   Versión 2 - Usando un bucle while interno
*
*   \param array --> el conjunto de elementos al que sumaremos su paridad
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int parityCount2(int* array, int len)
{
   int  i,j,  res=0,result;
   for (i=0; i<len; i++){      //Recorremos la lista de elementos
      result = 0;
      unsigned value=array[i];
      do{                      // Recorremos cada bit del elemento actual
         result^= value & 0x1;        //Aplicamos la mascara
         value >>= 1;                 //Realizamos el desplazamiento al elemento actual
      }while(value);
      res+=result; //Acumulamos el resultado de cada elemento
   }
   return res;
}

////////////////////////////////////////////////////////////////
/*
*   Método que realizará la suma de las paridades de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 3 - Usando un bucle while interno pero aplicando la máscara al
*            resultado total
*   \param array --> el conjunto de elementos al que sumaremos su paridad
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int parityCount3(int* array, int len)
{
   int  i,j,  res=0,result;
   for (i=0; i<len; i++){                 //Recorremos la lista de elementos
      result = 0;
      unsigned value=array[i];
      while(value){           // Recorremos cada bit del elemento actual
         result^= value;
         value >>= 1;         //Realizamos el desplazamiento sobre el elemento actual
      }
      res+=result & 0x1  //Aplicamos la máscara al resultado y lo acumulamos
   }
   return res;}
```

```
////////////////////////////////////////////////////////////////
/*
*   Método que realiza la suma de las paridades de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 4- Usando un bucle while interno en ensamblador
*
*   \param array --> el conjunto de elementos al que sumaremos su paridad
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
int parityCount4(int* array, int len)
{
   int  i,j,  res=0,val;
   unsigned x;
   for (i=0; i<len; i++){      //Recorremos la lista de elementos
      val=0;
      x=array[i];
      asm("ini3:          \n\t" //etiqueta
         "xor %[x],%[v]   \n\t" //xor val,x
         "shr %[x]        \n\t" //Desplazamiento a la derecha

         "cmp $0,%[x]     \n\t" //Comparamos 0:x
         "jne ini3        \n\t" // if !=0 goto ini3
         :  [v]"+r" (val)       // output-input
         :  [x]  "r" (x)  );    // input
      res+=val & 0x1; //Aplicamos la máscara
   }
   return res;
}
```

```
////////////////////////////////////////////////////////////////
/*
*   Método que realiza la suma de las paridades de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 5 -  CS: APP 3.49 - 32bits
*
*   \param array --> el conjunto de elementos al que sumaremos su paridad
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
////////////////////////////////////////////////////////////////
```

```
int parityCount5(int* array, int len)
{
   int  i,j,  res=0,result,val;
   unsigned x;
   for (i=0; i<len; i++){      //Recorremos la lista de elementos
      result = 0;
      val=0;
      x=array[i];              //Elemento actual
      for (j = 16; j > 0; j/=2) {  //Realizamos consecutivos desplazamientos al elemento actual
         x ^= (x >> j);
      }

      res+=x & 0x1; //Aplicamos la máscara al resultado y lo acumulamos
   }
   return res;
}

//////////////////////////////////////////////////////////////////
/*
*   Método que realiza la suma de las paridades de un conjunto de elementos
*   pasados como parámetros.
*
*   Versión 6 - ASM-cuerpo for - setpo
*
*   \param array --> el conjunto de elementos al que sumaremos su paridad
*   \param len --> el tamaño del array
*
*   \return int --> resultado de la suma
*/
//////////////////////////////////////////////////////////////////
int parityCount6(int* array, int len)
{
   int  i,j,  res=0,result,val;
   unsigned x;
   for (i=0; i<len; i++){
      result = 0;
      val=0;
      x=array[i];

         asm(
           "mov %[x], %%edx \n\t"   //Introducimos nuestro elemento en un registro
           "shr %%edx       \n\t"   //Desplazamiento realizado al elemento actual
           "xor %[x], %%edx   \n\t"   //Xor -> elemento original , elemento original desplazado
           "xor %%dh, %%dl   \n\t"   //Xor -> bytes superiores y bytes inferiores del registro
           "setp o%%dl        \n\t"   //Si parity flag = 1
           "movzx %%dl, %[x] \n\t"   //Volcamos el resultado
           : [x]"+r" (x)         //input-output
           : [j] "r" (j)
           : "edx"
         );

      res+=x & 0x1;    //Aplicamos la máscara y acumulamos el elemento
   }
   return res;}
```

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

int main()
{
   int i;         // inicializar array
   for (i=0; i<SIZE; i++)  // se queda en cache
    lista[i]=i;

   crono(parityCount1, "parityCount1 (en lenguaje C    )");
   crono(parityCount2, "parityCount2 (en lenguaje C    )");
   crono(parityCount3, "parityCount3 (en lenguaje C    )");
   crono(parityCount4, "parityCount4 (en lenguaje C    )");
   crono(parityCount5, "parityCount5 (en lenguaje C    )");
   crono(parityCount6, "parityCount6 (en lenguaje C    )");

   exit(0);
}
```

3. Ejercicios Opcionales

3.1 Sesión de depuración 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 coincide el volcado así obtenido con la Figura 3.
Si coinciden donde podemos ver lo siguiente
- 0x08048084 es la dirección de retorno
-0X080490b7 -> es la posición de inicio de nuestra lista
-0x00000009 -> el segundo argumento es el número de elementos

2- El volcado de 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>?

Lo podemos saber por el tamaño de la lista (<cuántos>) y qué tipo de dato tiene(<que>)
Para obtener los 9 enteros de nuestra lista necesitaremos introducir la siguiente instrucción en ddd -> x /9xw $0x80490b7
donde $0x80490b7 es la dirección de nuestro array

3- ¿Por qué la función suma preserva ahora %ebx y no hace lo mismo con %edx?
Porque el registro %ebx es salva invocado y %edx no, por dicha razón no hace falta preservar edx .

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

-Utiliza un modo de direccionamiento indirecto a registro y un direccionamiento a memoria.

-(%ebx,%edx,4)
- -%ebx → registro base
- -%edx → registro indice
- -4 → Factor de escala

-Que no realizaría correctamente la selección de los elementos de nuestra lista por lo que no realizara correctamente la suma.

3.2 Sesión de depuración suma_02_s_libC

1- ¿Qué error se obtiene si no se añade -lc al comando de enlazar?¿Qué tipo de error es? (en tiempo de ensamblado,enlazado, ejecución….)
suma_02_S_libC.o: In function `_start':
/home/antoniolm/Downloads/2 Ficheros fuente (1)/suma_02_S_libC.s:26: undefined reference to `printf'
/home/antoniolm/Downloads/2 Ficheros fuente (1)/suma_02_S_libC.s:30: undefined reference to `exit'

El error se produce en tiempo de enlazado y es debido a que no realizamos el enlace mediante -lc por lo cual al intentar enlazarlo no encuentra las funciones printf y exit.

3. Proporcionar Instrucciones paso a paso para obtener un volcado como el primero de la Figura 5 (argumentos de suma):
-en modo gráfico ddd (Examine <cant><fmt> <tam> desde <dir>) y
-en modo comando gdb (x/<fmt><addr>)
Paso 1 - Añadir un breakpoint en la instrucción call suma.
Paso 2 - Ejecutar el programa.

Paso 3.1 -Modo Gráfico - Data -> Memory

Paso 4.1 - Examine 5 hex words(4) from $esp

Paso 3.2 -Modo comando - Introducir => x /5xw $esp

5. Repetir 3 para el segundo volcado de la Figura 5 (argumentos de printf).
Paso 1 - Añadir un breakpoint en la instrucción call printf.
Paso 2 - Ejecutar el programa.

Paso 3.1 -Modo Gráfico - Data -> Memory
Paso 4.1 - Examine 5 hex words(4) from $esp

Paso 3.2 -Modo comando - Introducir => x /5xw $esp

6. Repetir 3 para el tercer volcado de la Figura 5 (argumentos de exit)
Paso 1 - Añadir un breakpoint en la instrucción call exit.
Paso 2 - Ejecutar el programa.

Paso 3.1 -Modo Gráfico - Data -> Memory
Paso 4.1 - Examine 4 hex words(4) from $esp

Paso 3.2 -Modo comando - Introducir => x /4xw $esp

8. En ese momento,justo antes de llamar a exit, podemos modificar el puntero de pila con el comando gdb: set $esp=$esp+4. ¿Qué pasa entonces?¿En qué afecta eso a nuestro programa?¿deberían obtener todos ese resultado?¿De qué depende el resultado?
Al realizar ese set perdemos el push 0 realizado antes del call exit ya que sería como haber realizado un pop en nuestra pila.Por ello no se produce correctamente la llamada a exit.

9. Repetir 8 poniendo -4 en lugar de +4
En este caso al restarle 4 a la pila es como si realizáramos un push del elemento siguiente en la pila el cual en mi caso es 0x25. Por esto no se produce correctamente la llamada a exit ya que no tendríamos como primer elemento el 0 necesario sino que tendríamos este 0x25 el cual no es su código de llamada.

3.3 Sesión de depuración suma_03_SC

2- Qué diferencia hay entre los comandos Next y Step?¿Y entre esos comando y su versión <..>i?

La principal diferencia es que step te va informando de porqué etiquetas va pasando mientras que next no te da ningun información.

Las versiones con i es debido a que en este tipo de comando se realiza un movimiento a través de las instrucciones máquina mientras que en la versión sin i es en líneas de código.

4 Comparación de tiempos

4.1 Comparación de tiempos en popCount

Para la realización de la comparación entre las diferentes versiones hemos realizado 11 ejecuciones de cada versión para realizar la comparación de una forma más precisa.

Los resultados de estas ejecuciones lo podemos ver en las tablas 01,02 y 03,El equipo utilizado tenía las siguientes características:

-Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
-cache size : 4096 KB

Optimización -O0	0	1	2	3	4	5	6	7	8	9	10	Media
popCount1	97501	96029	95588	95253	97903	96752	96253	95699	98463	96771	96664	96625,09091
popCount2	47587	47216	47161	47669	47865	48461	47025	48556	48220	48220	48127	47827,90909
popCount3	14545	14288	15205	14225	14199	14252	14736	14856	14193	14118	16033	14604,54545
popCount4	22055	22445	22467	22188	22158	23016	22680	24951	22757	22161	24312	22835,45455
popCount5	941	962	1025	945	945	982	931	1080	953	1033	1021	983,4545455

Tabla 01 - Popcount con optimización -O0

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	Media
popCount1	36708	33166	32161	32328	31981	31965	31792	32016	31901	32151	31855	32547,63636
popCount2	11738	11393	11939	12110	11746	10996	11934	11165	12205	11980	12164	11760,90909
popCount3	13268	12375	11765	11568	12534	11727	13045	12330	12024	11681	11438	12159,54545
popCount4	7174	6686	6652	6489	7545	7468	6793	6951	6653	6792	6535	6885,272727
popCount5	538	497	588	526	586	532	540	683	528	564	527	555,3636364

Tabla 02 - Popcount con optimización -O1

Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	Media
popCount1	37518	35466	35687	35907	35162	34780	38016	35384	34811	35370	35973	35824,90909
popCount2	12840	14487	13917	13496	12633	12032	13884	13085	12522	13356	14196	13313,45455
popCount3	12412	11526	11685	11575	11883	11816	11559	11534	11528	11948	11389	11714,09091
popCount4	4955	4952	4830	6066	4930	4870	4797	5782	5200	6033	4910	5211,363636
popCount5	509	550	530	571	563	579	529	1292	557	517	534	611,9090909

Tabla 03 - Popcount con optimización -O2

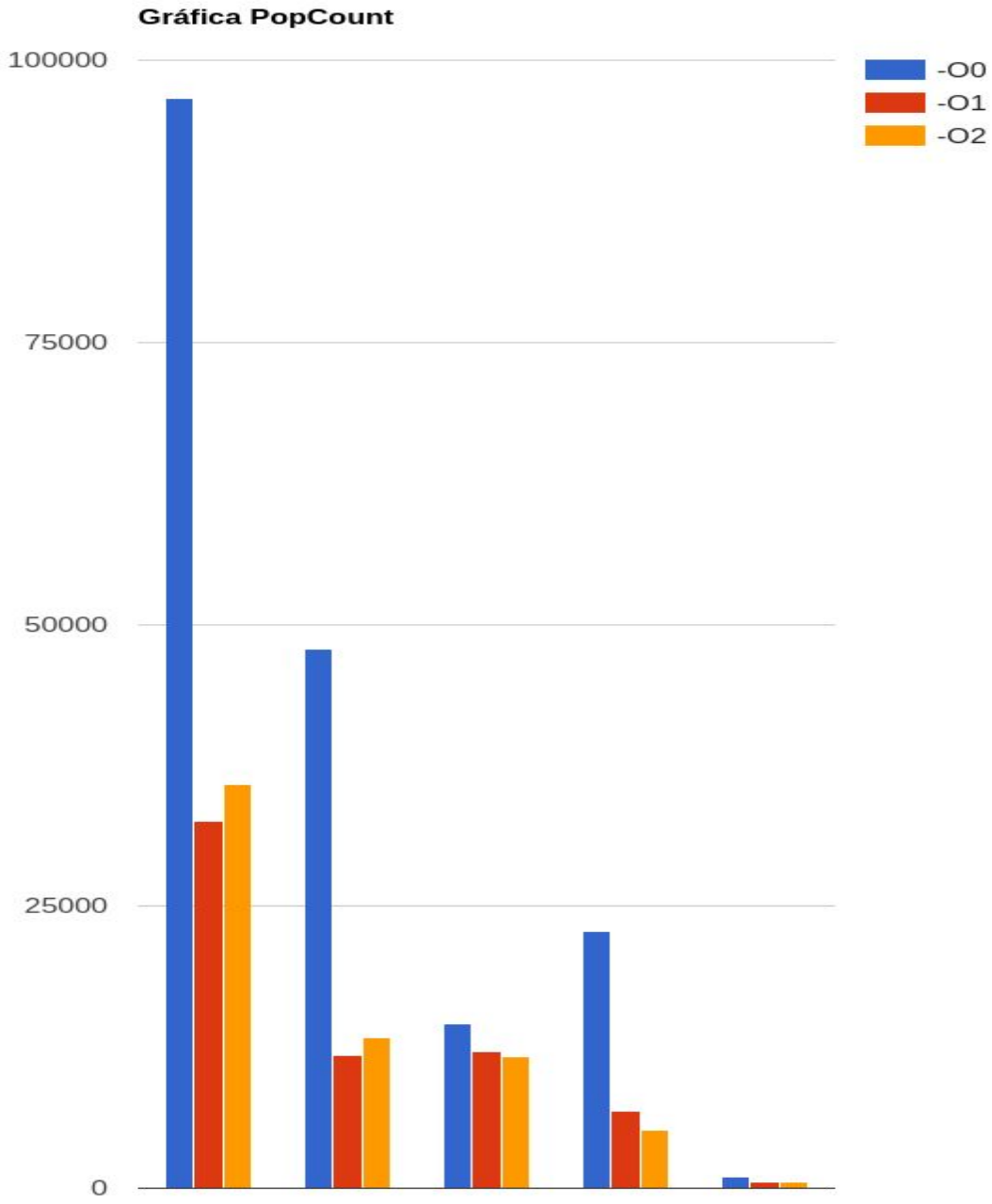

Figura 01 - Gráfica comparativa de las versiones de popCount

Como podemos ver en la figura 01 la versión 5 es la versión más eficiente de todas. La peor versión en la primera de todas donde alcanza unos casi de 100000ms mientras que en la última versión alcanza unos tiempos de 500ms.

4.2 Comparación de tiempos en parityCount

Para la realización de la comparación entre las diferentes versiones hemos realizado 11 ejecuciones de cada versión para realizar la comparación de una forma más precisa.

Los resultados de estas ejecuciones lo podemos ver en la tabla 04,05 y 06.El equipo utilizado tenía las siguientes características:

-Intel(R) Core(TM) i7-4510U CPU @ 2.00GHz
-cache size : 4096 KB

Optimización -O0	0	1	2	3	4	5	6	7	8	9	10	Media
parityCount1	97304	95552	97312	96810	98017	97395	103599	98854	100303	98868	102374	9876
parityCount2	47587	47325	47550	46936	49058	49012	51983	49185	49349	50145	49650	48889
parityCount3	42900	48937	45503	41847	44200	42725	42775	42994	43167	44395	43354	43890
parityCount4	19706	13416	12928	18915	12910	13290	13583	14097	13152	13201	13718	14446
parityCount5	14825	14502	15309	14862	14180	16344	14580	16032	14893	14669	15371	15051
parityCount6	3582	3438	3714	3702	3790	3314	3516	3791	3281	3534	3286	3540

Tabla 04 - ParityCount con optimización -O0

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	Media
parityCount1	32531	32601	31588	32973	32155	32717	31665	31446	31702	32633	33230	32294
parityCount2	11007	12245	14268	11857	11783	11581	11097	11401	11882	12448	12364	11993
parityCount3	15820	15156	14726	13544	15836	15243	15091	14982	15808	15322	16809	15303
parityCount4	8136	8192	8666	8213	8779	7870	8232	8228	8492	8114	9501	8402
parityCount5	5929	6227	5942	6948	6238	6225	6059	5879	5674	5497	6505	6102
parityCount6	1141	1151	1127	1443	1112	1170	1269	1151	1097	1160	1138	1178

Tabla 05 - ParityCount con optimización -O1

Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	Media
parity Count1	43229	35236	34925	36547	35008	34630	35288	35423	35350	35982	36375	36181
parity Count2	12355	13636	12750	14519	11745	13450	11857	11100	12069	12689	12258	12584
parity Count3	8926	7877	11409	7752	8786	8781	8624	8316	7542	8182	8094	8571
parity Count4	8054	7809	8618	8093	8131	7933	7808	8085	9116	7826	8631	8191
parity Count5	4639	4637	4701	4553	5077	4679	4776	4916	4643	4647	4769	4730
parity Count6	1163	1122	1123	1221	1089	1110	1159	1130	1146	1148	1130	1140

Tabla 06 - ParityCount con optimización -O2

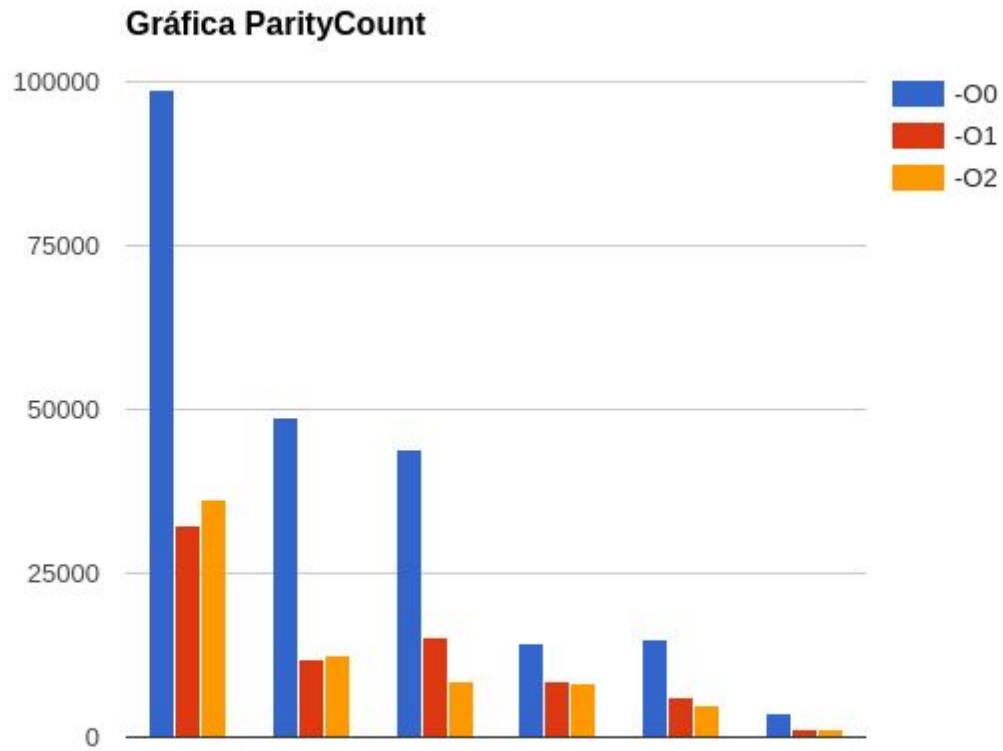

Figura 02 - Gráfica comparativa de las versiones de popCount

Como podemos ver en la figura 02 la versión 6 es la versión más eficiente de todas. La peor versión en la primera de todas donde alcanza unos casi de 100000ms mientras que en la última versión alcanza unos tiempos de 1000ms.