Práctica 3: Programación Mixta C-asm x86 Linux

Nombre: Javier Martín Gómez

Curso: 2ºC

DNI: 77143672Q

Diario de Trabajo:

Semana 7: Comenzamos con la práctica 3, el profesor utiliza las 2 horas para explicar la práctica.

Semana 8: Comienzo con Popcount, tengo muchas dudas así que se las pregunto al profesor y a mis compañeros

Semana 9: Acabo Popcount en mi casa y comienzo con Parity, nos ayudamos entre mis compañeros y yo, con seguimos avanzar bastante.

Semana 10: Consigo acabar Parity en mi casa, utilizo las clases de prácticas para repasarlo y ver que está correcto.

Popcount

```
//  segÃ°n la versiÃ³n de gcc y opciones de optimizaciÃ³n usadas, tal vez haga falta
//  usar gcc â€"fno-omit-frame-pointer si gcc quitara el marco pila (%ebp)

/*for((i=0;i<11;i++));do echo $i ; ./popcount;done | pr -11 -l 20 -w 80


for((i=0;i<10;i++));do ./popcount;done
*/

#include <stdio.h>    // para printf()
#include <stdlib.h>   // para exit()
#include <sys/time.h> // para gettimeofday(), struct timeval
                                                                                //COMIE
NZO DE MODIFICACION
#define TEST 0 //PONER A 0 PARA HACER TEST
#define COPY_PASTE_CALC 1   //PONER A 1 PARA MOSTRAR SOLO SEGUNDOS

#if ! TEST
       #define NBITS 20
       #define SIZE (1<<NBITS) // tamaÃ±o suficiente para tiempo apreciable
       unsigned lista[SIZE]; // unsigned para que desplazamiento derecha sea logico
       #define RESULT (NBITS*(1<<NBITS-1)) // nueva formula

#else
       /*
       #define SIZE 4
       unsigned lista[SIZE]={0x80000000,0x00100000,0x00000800,0x00000001};
       #define RESULT 4
       */
```

```
        /*
        #define SIZE 8
        unsigned lista[SIZE]={0x7fffffff,0xffefffff,0xfffffeff,0xfffffffe,
                              0x01000024,0x00356700,0x8900ac00,0x00bd00ef};
        #define RESULT 156
        */



        /*
        #define SIZE 8
        unsigned lista[SIZE]={0x0       ,0x10204080,0x3590ac06,0x70b0d0e0,
                              0xffffffff,0x12345678,0x9abcdef0,0xcafebeef};
        #define RESULT 116
        */

#endif


        //FIN MODIFICACION

int resultado=0;



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


    for (i=0; i<len; i++){
        x=array[i];
        for (j=0; j<8*sizeof(int) ; j++){
                result += x & 0x1;
                x >>= 1;
        }
   }

    return result;
}



int popcount2(unsigned* array, int len)
{
    int  i;
    unsigned x;
    int result=0;


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

    return result;
}

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


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

```
        asm("\n"
        "ini3:                          \n"
        "shr %[x]                       \n" //El nÃºmero desplazado queda en CF
        "adc $0  , %[r]                    \n" //AÃ±ade 0 + Carry Flag
        "cmp $0  , %[x]                    \n"
        "jne ini3                       \n"
        :       [r]"+r" (result)
        :       [x] "r" (x)             );
        }

    return result;
}


int popcount4(unsigned* array, int len){
        int val = 0;
        int result=0;
        int i;
        int j;
        unsigned x;
        for(i=0 ; i < len ; i++){
        val = 0;
        x = array[i];
                for(j = 0 ; j < 8 ; j++){
                        val += x & 0x01010101/*01010101L*/;
                        x >>= 1;
                }
                val += (val >> 16);
                val += (val >> 8);
                result += (val & 0xFF);
        }
        return result;
}

int popcount5(unsigned* array, int len) {
         int i;
         int val, result = 0;
         int SSE_mask[] = { 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f };
         int SSE_LUTb[] = { 0x02010100, 0x03020201, 0x03020201, 0x04030302 };


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

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

                                   "paddb   %%xmm2, %%xmm3    \n"
                                   "pxor    %%xmm0, %%xmm0    \n"
                                   "psadbw  %%xmm0, %%xmm3    \n"
                                   "movhlps %%xmm3, %%xmm0    \n"
                                   "paddd   %%xmm3, %%xmm0    \n"
                                   "movd    %%xmm0, %[val]    \n"
                                   : [val]"=r"    (val)
                                   : [x]   "m"    (array[i]),
                                     [m]   "m" (SSE_mask[0]),
                                     [l]   "m" (SSE_LUTb[0])
                  );
                  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 COPY_PASTE_CALC                                  //AÃ‘ADIDO: A la hora de hacer
copypaste que imprima solo segundos.
    printf(  "%ld" "\n" ,       tv_usecs);          //AÃ‘ADIDO
#else                                                //AÃ‘ADIDO
    printf("resultado = %d\t", resultado);
    printf("%s:%9ld us\n", msg, tv_usecs);
#endif                                               //AÃ‘ADIDO
}

int main()
{
    #if ! TEST //AÃ‘ADIDO
       int i;                // inicializar array
       for (i=0; i<SIZE; i++)// se queda en cache
        lista[i]=i;
    #endif  //AÃ‘ADIDO. Esto solo se hace si no estamos haciendo el copypaste (el test)

    crono(popcount1, "popcount1 (lenguaje C -     for)");         //MODIFICADO
    crono(popcount2, "popcount2 (lenguaje C -   While)");
    crono(popcount3, "popcount3 (lenguaje C -BucleASM)");         //MODIFICADO
    crono(popcount4, "popcount4 (lenguaje C -    Tree)");
    crono(popcount5, "popcount5 (Leng ASM   -   SSSE3)");

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

Parity

```
//  segÃºn la versiÃ³n de gcc y opciones de optimizaciÃ³n usadas, tal vez haga falta
//  usar gcc â€“fno-omit-frame-pointer si gcc quitara el marco pila (%ebp)

/*for((i=0;i<11;i++));do echo $i ; ./popcount;done | pr -11 -l 20 -w 80


for((i=0;i<10;i++));do ./popcount;done
*/

#include <stdio.h>    // para printf()
#include <stdlib.h>   // para exit()
#include <sys/time.h> // para gettimeofday(), struct timeval
                                                                                //COMIE
NZO DE MODIFICACION
#define TEST 0        //PONER A 0 PARA HACER TEST
#define COPY_PASTE_CALC 1   //PONER A 1 PARA MOSTRAR SOLO SEGUNDOS

#if ! TEST
       #define NBITS 20
       #define SIZE (1<<NBITS) // tamaÃ±o suficiente para tiempo apreciable
       unsigned lista[SIZE]; // unsigned para que desplazamiento derecha sea logico
       #define RESULT (1<<NBITS-1) // nueva formula   //PARITY

#else
       /*
       #define SIZE 4
       unsigned lista[SIZE]={0x80000000,0x00100000,0x00000800,0x00000001};
       #define RESULT 4
       */



       /*
       #define SIZE 8
       unsigned lista[SIZE]={0x7fffffff,0xffefffff,0xfffffeff,0xfffffffe,
                             0x01000024,0x00356700,0x8900ac00,0x00bd00ef};
       #define RESULT 8 //PARITY
       */



       /*
       #define SIZE 8
       unsigned lista[SIZE]={0x0       ,0x10204080,0x3590ac06,0x70b0d0e0,
                             0xffffffff,0x12345678,0x9abcdef0,0xcafebeef};
       #define RESULT 2 //PARITY
       */

#endif


       //FIN MODIFICACION

int resultado=0;



int parity1(unsigned* array, int len)
{
    int  i, j;
    unsigned x;
    int val, result=0;



    for (i=0; i<len; i++){
       x=array[i];
       val=0; // VAL LIBRE EN CADA ITERACIÃ“N .. TRANSPARENCIA 43
       for (j=0; j<8*sizeof(int) ; j++){

              val^= x & 1;
              x >>= 1;
```

```
        }
        result += val;
   }

    return result;
}



int parity2(unsigned* array, int len) //PARITY 3 EN PAGINA 352
{
    int  i;
    unsigned x;
    int val,result=0;


    for (i=0; i<len; i++ )
    {
       x=array[i];
       val = 0;
       do
       {
              val^=x&1;
              x >>= 1;
       } while (x);
       result += val;
       }

    return result;
}

int parity3(unsigned* array, int len){
    int  i;
    unsigned x;
    int val,result=0;


    for (i=0; i<len; i++ )
    {
       x=array[i];
       val = 0;
       do
       {
              val^=x;
              x >>= 1;
       } while (x);
       result += val & 0x1;  //SE HACE LA MÃ¡SCARA AL FINAL
       }

    return result;
}

int parity4(unsigned* array, int len){
    int  i;
    unsigned x;
    int val,result=0;


    for (i=0; i<len; i++ )
    {
       x=array[i];
       val = 0;
       asm("\n"
       "ini4:                \n"
       "xor %[x], %[v]              \n"
       "shr   $1, %[x]              \n" //Afecta al ZF
       "jnz ini4             \n" //Saltar si el flag de 0 NO estÃ¡ activado.
       : [v]"+r"      (val)
       : [x] "r"      (x)
       );
       result += val & 0x1;  //SE HACE LA MÃ¡SCARA AL FINAL
    }

    return result;
}
int parity5(unsigned* array, int len) // CS:APP
{
```

```
        int i, j, res=0;
        unsigned x;
        for (i=0; i<len; i++) {
                x = array[i];
                for (j=16; j>0; j/=2)
                        x ^= x >> j;
                res += (x & 0x01);
        }
        return res;
}

int parity6(unsigned* array, int len) {
        int j;
        unsigned entero = 0;
        int resultado = 0;

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

                entero = array[j];
                asm(
                        "mov    %[x],   %%edx       \n"
                                "shr    $16,    %%edx       \n"
                                "xor    %[x],   %%edx       \n"
                                "xor    %%dh,   %%dl        \n"
                                "setpo  %%dl                \n"
                                "movzx  %%dl,   %[x]        \n"
                                : [x] "+r" (entero)
                        :
                        : "edx"
                                                );
                resultado += entero;
        }
        return resultado;
}


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 COPY_PASTE_CALC                                 //AÃ‘ADIDO: A la hora de hacer
copypaste que imprima solo segundos.
    printf(  "%ld" "\n" ,       tv_usecs);          //AÃ‘ADIDO
#else                                               //AÃ‘ADIDO
    printf("resultado = %d\t", resultado);
    printf("%s:%9ld us\n", msg, tv_usecs);
#endif                                              //AÃ‘ADIDO
}

int main()
{
    #if ! TEST //AÃ‘ADIDO
        int i;                // inicializar array
        for (i=0; i<SIZE; i++)// se queda en cache
         lista[i]=i;
    #endif  //AÃ‘ADIDO. Esto solo se hace si no estamos haciendo el copypaste (el test)

    crono(parity1, "parity1 (lenguaje C -        For)");          //MODIFICADO - PARITY
    crono(parity2, "parity2 (lenguaje C -      While)");          //MODIFICADO - PARITY
    crono(parity3, "parity3 (lenguaje C - Final Mask)");
    crono(parity4, "parity4 (while ASM  - Final Mask)");
    crono(parity5, "parity5 (lenguaje C -       Tree)");
    crono(parity6, "parity6 (leng ASM   -SETcc-MOVZx)");

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

Gráfica Popcount

/proc/cpuinfo — Intel(R) Core(TM) i5-4690 CPU @ 3.50GHz

6144 KB

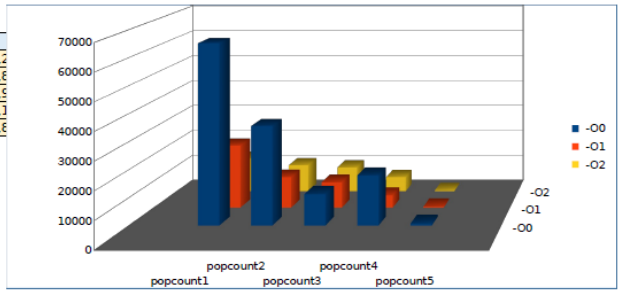
POPCOUNTt: gcc -m32 popcount.c -o popcount -O<n>
for((i=0;i<11;i++));do echo $i ; ./popcount;done | pr -11 -l 20 -w 80
Ignorar medición 0, repetir columna si alguna medición se sale demasiado de la media.

Optimización -O0		1	2	3	4	5	6	7	8	9	10	MEDIA
popcount1 (lenguaje C - | for)	80657	69047	69553	63383	68690	63491	63500	62342	63384	64802	63447	65163,9
popcount2 (lenguaje C - | While)	38473	37516	38743	35285	35912	35669	35046	35371	33626	34384	34519	35607,1
popcount3 (lenguaje C - | BucleASM)	11667	12679	10597	11202	9701	9699	10611	9389	16071	11384	10592	11192,5
popcount4 (lenguaje C - | Tree)	18374	18638	18541	16791	18190	18809	18517	18087	18022	16688	16926	17920,9
popcount5 (leng ASM - | SSSE3)	717	693	772	669	479	784	797	731	711	665	671	697,2

Optimización -O1		1	2	3	4	5	6	7	8	9	10	MEDIA
popcount1 (lenguaje C - | for)	30980	23337	23967	22958	23069	23729	22388	23170	22881	23162	23412	23207,3
popcount2 (lenguaje C - | While)	22249	12345	11166	11114	11802	10094	11983	12037	11132	11539	10455	11366,7
popcount3 (lenguaje C - | BucleASM)	8945	8884	9916	9774	8599	10483	9700	9580	8845	8587	10050	9441,8
popcount4 (lenguaje C - | Tree)	5182	4192	4378	4233	4567	4990	4088	4029	5177	5634	4659	4594,7
popcount5 (leng ASM - | SSSE3)	348	332	342	335	332	427	135	331	352	413	347	334,6

Optimización -O2		1	2	3	4	5	6	7	8	9	10	MEDIA
popcount1 (lenguaje C - | for)	12472	13417	13108	13649	13629	11544	12064	12701	12094	12648	12728	12758,2
popcount2 (lenguaje C - | While)	18654	11101	9681	9682	9863	12118	10627	9585	10915	9549	8687	10180,8
popcount3 (lenguaje C - | BucleASM)	19319	9216	8076	9173	9553	8612	9242	10468	8860	9701	10218	9311,9
popcount4 (lenguaje C - | Tree)	5764	5786	5753	4514	5509	6392	5626	5583	5222	4556	6120	5506,1
popcount5 (leng ASM - | SSSE3)	254	344	333	341	333	335	356	385	337	386	378	352,8

POPCOUNT:	-O0	-O1	-O2
popcount1	65163,9	23207,3	12758,2
popcount2	35607,1	11366,7	10180,8
popcount3	11192,5	9441,8	9311,9
popcount4	17920,9	4594,7	5506,1
popcount5	697,2	334,6	352,8



Gráfica Parity

/proc/cpuinfo — Intel(R) Core(TM) i5-4690 CPU @ 3.50GHz
6144 KB

PARITY: gcc -m32 parity.c -o parity -O<n>
for((i=0;i<11;i++));do echo $i ; ./parity;done | pr -11 -l 20 -w 80
Ignorar medición 0, repetir columna si alguna medición se sale demasiado de la media.

Optimización -O0		1	2	3	4	5	6	7	8	9	10	MEDIA
parity1 (lenguaje C - | for)	82360	69719	77400	69565	69282	72023	68622	76760	68520	77761	71522	72117,4
parity2 (lenguaje C - | While)	38235	37502	36170	43654	40973	38078	37310	36026	36301	38559	44656	38922,9
parity3 (lenguaje C - | Final Mask)	32980	32408	32720	32641	33087	33157	32800	33432	33358	33676	33033	33031,2
parity4 (While ASM - | Final Mask)	10144	9913	10464	9133	10719	9531	9432	9887	10021	10691	10813	10060,4
parity5 (lenguaje C - | Tree)	10322	10956	9722	11998	9802	10777	10670	11059	11546	11955	11514	10999,9
parity6 (leng ASM - SETcc-MOVZx)		2363	2364	2929	2633	2356	2881	3122	2553	2556	2661	2405	2646

Optimización -O1		1	2	3	4	5	6	7	8	9	10	MEDIA
parity1 (lenguaje C - | for)	11393	13233	12063	12636	12064	12897	11987	11859	10904	11989	11882	12151,4
parity2 (lenguaje C - | While)	9666	8616	9536	8570	9366	8344	9507	8534	8812	9051	8988	8932,4
parity3 (lenguaje C - | Final Mask)	6272	6329	5813	5248	5547	6160	5919	5617	6175	5938	5577	5832,3
parity4 (While ASM - | Final Mask)	13401	6363	9705	7125	6919	6889	6956	6897	6678	6791	10791	7511,4
parity5 (lenguaje C - | Tree)	4467	4220	4284	4958	4482	4401	4330	5309	4845	4802	3949	4558
parity6 (leng ASM - SETcc-MOVZx)		708	679	702	728	491	408	703	723	693	695	1316	713,8

Optimización -O2		1	2	3	4	5	6	7	8	9	10	MEDIA
parity1 (lenguaje C - | for)	13567	13101	11990	13836	13507	13261	12936	13319	12824	13325	13020	13111,9
parity2 (lenguaje C - | While)	20250	10296	11012	9577	9719	10080	9498	9976	10432	9674	10242	10050,6
parity3 (lenguaje C - | Final Mask)	6394	6462	7232	6608	7570	6287	6860	5982	5773	6838	6743	6635,5
parity4 (While ASM - | Final Mask)	15586	8718	7019	7621	7435	8076	8645	7050	8400	8237	8181	7938,2
parity5 (lenguaje C - | Tree)	3679	3741	3893	3271	3335	3336	3278	4853	4424	3597	3522	3725
parity6 (leng ASM - SETcc-MOVZx)		839	751	492	693	678	685	697	766	703	749	702	691,6

PARITY:	-O0	-O1	-O2
parity1	72117,4	12151,4	13111,9
parity2	38922,9	8932,4	10050,6
parity3	33031,2	5832,3	6635,5
parity4	10060,4	7511,4	7938,2
parity5	10999,9	4558	3725
parity6	2646	713,8	691,6