Práctica 3: Programación mixta C-asm x86 Linux

Vicente Martínez Rodríguez

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

Primera versión:

```
int v1(unsigned* x) {
	int i, j;
	unsigned result;
	int result_total=0;
	for (i = 0; i < SIZE; i++) {
		result= x[i];
		for (j = 0; j < WSIZE; j++) {
			result_total+= result & 0x1;
			result>>=1;
		}
	}
	return result_total;
}
```

Segunda versión:

```
int v2(unsigned* x) {
	int i, j;
	unsigned result;
	int result_total=0;
	for (i = 0; i < SIZE; i++) {
		result= x[i];
		do{
			result_total+= result & 0x1;
			result>>=1;
		}while(result);
	}
	return result_total;
}
```

Tercera versión:

```
int v3(int* array, int len){
        int  i, result_total=0
        unsigned result;
        for (i=0; i<len; i++){
                result= array[i];
                asm("\n"
                        "bucle1:        \n\t"
                        "shr %[x] \n\t"
                        "adc $0, %[r] \n\t"
                        "test %[x], %[x]\n\t"
                        "jne bucle1 \n\t"
                        : [r] "+r" (result_total)
                        : [x] "r" (result)
                );
        }
          return result_total;
}
```

Cuarta versión:

```
int v4(unsigned* x, int len) {
int val1,val2, i, j, total=0;
for(j=0; j<len; j++){
        val2=x[j];
        val1=0;
        for (i = 0; i < 8; i++) {
                val1 += val2 & 0x01010101;
                val2 >>= 1;
        }
        val1 += (val1 >> 16);
        val1 += (val1 >> 8);
        total+= (val1 & 0xFF);
}
return total;
}
```

Quinta versión:

```
int v5(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 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;
}
```

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

Primera Versión:

```
int version1(unsigned* x) {
        int i, j;
        unsigned elem;
        int xor;
        int result_total=0;
        for (i = 0; i < SIZE; i++) {
                elem= x[i];
                xor=0;
                for (j = 0; j < WSIZE; j++) {
                        xor^= (elem & 0x1);
                        elem>>=1;
                }
      result_total+= xor;
        }
        return result_total;
}
```

Segunda versión:

```
int version2(unsigned* array){
        int i, result_total=0, xor;
        unsigned elemento;
        for (i = 0; i < SIZE; i++) {
                elemento= array[i];
                xor=0;
                do{
                        xor^= elemento & 0x1;
                        elemento>>=1;
                }while(elemento);
                result_total+= xor;
        }
        return result_total;
}
```

Tercera versión:

```
int version3(unsigned *array){
        int i, result_total=0, xor;
        unsigned elemento;
        for(i=0; i<SIZE;i++){
                elemento=array[i];
                xor = 0;
                while (elemento) {
                        xor ^= elemento;
                        elemento >>= 1;
                }
                result_total+=(xor & 0x1);
        }
        return result_total;
}
```

Cuarta versión:

```
int version4(unsigned *array){
        int i, result_total=0, val;
        unsigned elemento;
        for(i=0; i<SIZE; i++){
                elemento=array[i];
                val=0;
                asm("\n"
                "ini3:                  \n\t"
                        "XOR %[x], %[v] \n\t"
                        "SHR %[x] \n\t"
                        "JNZ ini3 \n\t"
                        "AND $1, %[v] \n\t"
                        : [v] "+r" (val)
                        : [x] "r" (elemento)
                );
                result_total+=val;
        }
        return result_total;    }
```

Quinta versión:

```
int version5(unsigned *array){
        int i,j,resultado=0;
        unsigned elemento;
        for(i=0; i<SIZE; i++){
                elemento= array[i];
                for(j=16; j==1; j/=2){
                        elemento^=(elemento>>j);
                }
                resultado+=(elemento & 0x01);
        }
        return resultado;
}
```

Mediciones y Gráficas.

Comparativa distintas versiones de popcount

/proc/cpuinfo

Intel(R) Core(TM) i5-3317U CPU @ 1.70GHz
cache size : 3072 KB

POPCOUNT

gcc -m32 popcount.c -o popcount -O<n>

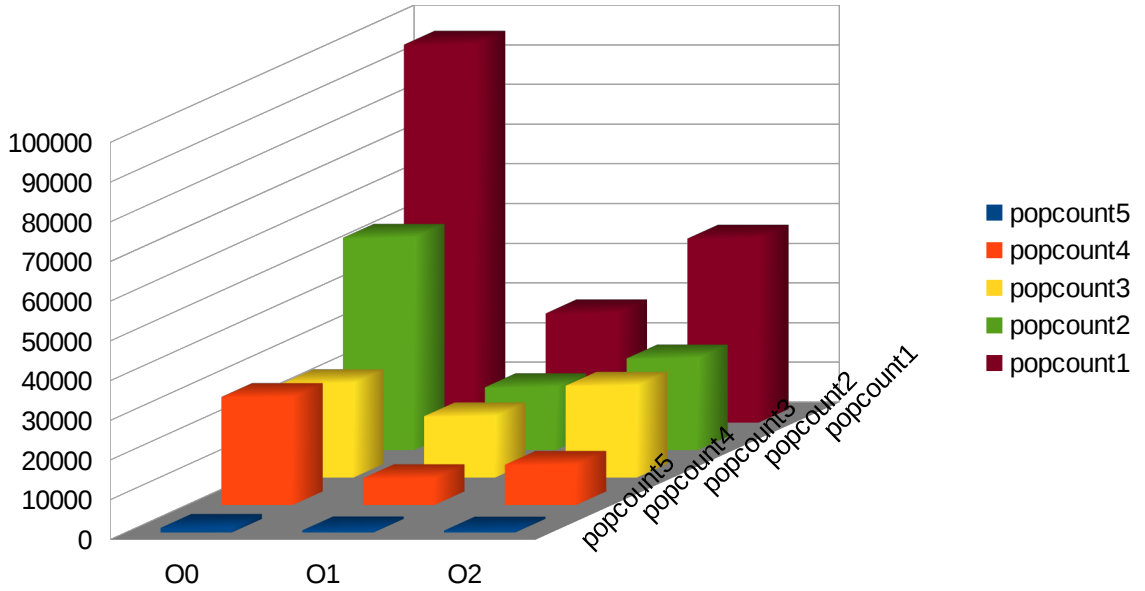
Optimización -O0	0	1	2	3	4	5	6	7	8	9	10	MEDIA
popcount1 (lenguaje C- for)	93733	98569	94165	93900	93486	96181	95256	93653	94546	94997	97579	95233,2
popcount2 (lenguaje C- while)	52949	53213	53267	53018	54867	54527	53459	53221	53080	53303	53059	53501,4
popcount3(lenguajeASM -cuerpo while)	28177	23507	23618	23499	23705	23539	25660	23618	23548	23696	23560	23795
popcount4(I.C.S:APP 3.49-group 8b)	26954	27008	27083	27104	27109	27083	27410	27143	26980	27107	28766	27279,3
popcount5	1204	1219	1204	1226	1195	1205	1203	1228	1205	1205	1284	1217,4

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	MEDIA
popcount1 (lenguaje C- for)	31598	28630	26554	26566	27807	27087	27034	28646	26755	26902	29401	27538,2
popcount2 (lenguaje C- while)	15888	15882	15813	15880	15817	15938	15817	15883	15814	15879	15815	15853,8
popcount3(lenguajeASM -cuerpo while)	15444	15473	15445	15740	15402	15470	15442	15470	15450	15471	15446	**15480,9**
popcount4(I.C.S:APP 3.49-group 8b)	7354	6963	7039	6994	6946	6493	6786	6953	6931	6779	6432	6831,6
popcount5	589	587	580	637	591	590	616	590	589	589	592	596,1

Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	MEDIA
popcount1 (lenguaje C- for)	49307	45889	46829	45280	44209	46633	42163	48110	45205	47471	51796	46358,5
popcount2 (lenguaje C- while)	23111	23118	23267	23114	23182	23073	23075	23113	23198	23106	23077	23132,3
popcount3(lenguajeASM -cuerpo while)	23086	23080	23225	23084	23208	23075	23076	23084	23179	23644	23081	23173,6
popcount4(I.C.S:APP 3.49-group 8b)	10144	10146	10195	10146	10197	10139	10141	10221	10193	10235	10139	10175,2
popcount5	544	545	544	546	546	545	544	546	547	545	557	546,5

Las medias son:

POPCOUNT	O0	O1	O2
popcount5	1217,4	596,1	546,5
popcount4	27279,3	6831,6	10175,2
popcount3	23795	15480,9	23173,6
popcount2	53501,4	15853,8	23132,3
popcount1	95233,2	27538,2	46358,5



PARITY gcc -m32 parity.c -o parity -O<n>

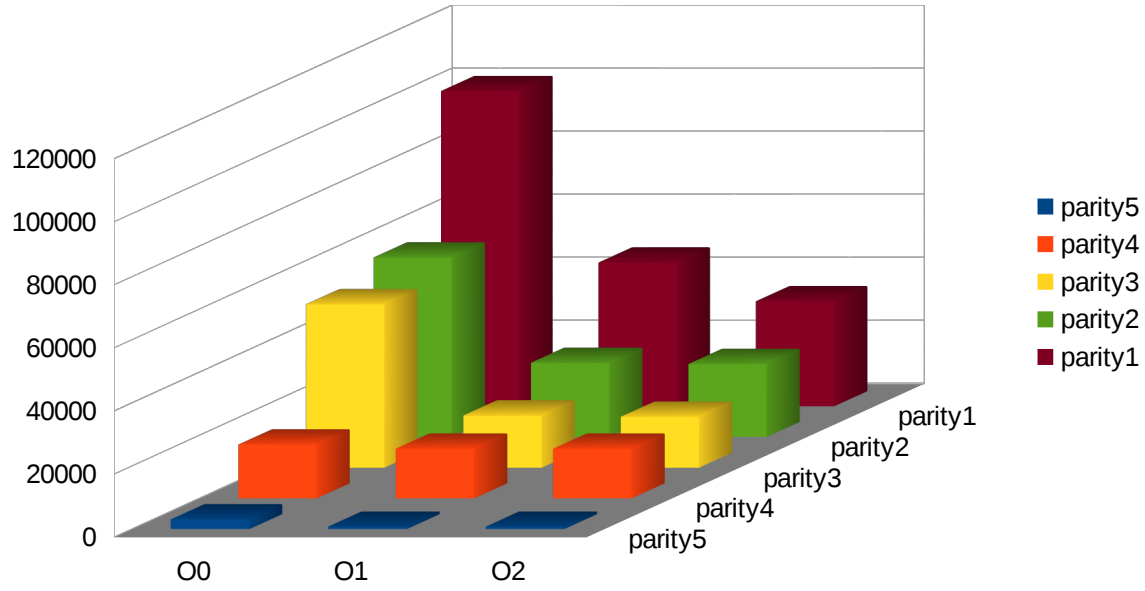
Optimización -O0	0	1	2	3	4	5	6	7	8	9	10	MEDIA
parity1 (lenguaje C- for)	100090	100588	100308	99382	102582	99412	99426	100115	99284	100026	99504	100062,7
parity2 (lenguaje C- while)	56059	56377	56157	55940	55812	55877	55858	56284	55862	66478	55889	57053,4
parity3(I.C.S:APP 3.49-mask final)	52338	50038	49997	49907	54776	49910	63809	50241	49875	50633	49914	51910
parity4(lenguajeASM -cuerpo while)	17181	17145	17072	17106	17067	17113	18106	17145	17083	17138	17147	17212,2
parity5	3210	3206	3234	3195	3193	3247	3194	3216	3198	3213	3197	3209,3

Optimización -O1	0	1	2	3	4	5	6	7	8	9	10	MEDIA
parity1 (lenguaje C- for)	39302	49398	45493	44559	44940	46214	43052	45573	45094	46573	46033	45692,9
parity2 (lenguaje C- while)	23247	23394	23212	23423	23267	26679	23215	23291	23127	23365	23262	23623,5
parity3(I.C.S:APP 3.49-mask final)	16623	16686	16607	16685	16615	16685	16608	16713	16618	16715	16610	16654,2
parity4(lenguajeASM -cuerpo while)	15824	15871	15800	15875	15798	15903	15984	15875	15800	15874	15799	15857,9
parity5	851	850	853	853	853	886	899	855	855	850	847	860,1

Optimización -O2	0	1	2	3	4	5	6	7	8	9	10	MEDIA
parity1 (lenguaje C- for)	30873	32779	34259	33130	30814	32368	33449	35325	34455	33480	34051	33411
parity2 (lenguaje C- while)	23193	23118	23179	23123	23217	23083	23201	23400	23756	23095	23210	23238,2
parity3(I.C.S:APP 3.49-mask final)	15860	15800	15878	20081	15842	15847	15876	15792	15877	15803	15873	16266,9
parity4(lenguajeASM -cuerpo while)	15870	15791	15871	15817	16052	15795	15946	15815	15872	15819	15871	15864,9
parity5	860	837	836	840	1037	835	835	835	838	833	833	855,9

Las medias son:

POPCOUNT	O0	O1	O2
parity5	3209,3	860,1	855,9
parity4	17212,2	15857,9	15864,9
parity3	51910	16654,2	16266,9
parity2	57053,4	23623,5	23238,2
parity1	100062,7	45692,9	33411



Diario de Trabajo:

Miércoles 6 de Noviembre de 2013: He empezado a realizar y comprobar como funcionan y se compilan los códigos del guión tutorial.

Miércoles 13 de Noviembre de 2013: He estado realizando varias versiones del ejercicio 4.1 (version 1, version 2 y parte de la version 3).

Miércoles 27 de Noviembre de 2013: Desde la última clase de prácticas he terminado la parte obligatoria del ejercicio 4.1 y me he quedado por la version 3 del ejercicio 4.2.

Sabado 30 de Noviembre de 2013: He terminado del ejercicio 4.2 las versiones 3, 4, la version 5 creo que no funciona correctamente.