Universidad De Granada

Grado ingeniería informática (2017 – 2018)

Estructura De Los Computadores

Practica 3: Popcount y Parity

Trabajo realizado por Antonio Miguel Morillo Chica

---

1. Diario de trabajo popcount y parity.

- M [29/10/2017] : Comenzar a organizar el documento con todas las preguntas.
- T [3/11/2017] : Comencé a montar el popcount y me quedé en el 4.
- M [5/11/2017] : Terminé el popcount y saqué las gráficas.
- M [6/11/2017] : Acabé el código del parity.
- T [10/11/2017] : Saqué los códigos del parity.
- M [16/11/2017] : Monté el pdf tanto de la parte del parity como la parte del popcount.
- T[16/11/2017] : Comenté gran parte del código y acabé de hacer las preguntas de los apéndices.

2. Ejercicios de los archivos suma (algunos).

2.1. Suma_01_S_cdecl

Ejercicio 2: ¿Por qué la función suma preserva ahora %ebx y no hace lo mismo con %edx?

Porque edx es un registro salvainvocado, mientras que por otro lado ebx es un registro salvainvocante.

Ejercicio3: ¿Qué modos de direccionamiento usa la instrucción add(%ebx, %edx, 4), %eax? ¿Cómo se llama a cada componente del primer modo? El último componente se denomina escala. ¿Qué sucedería si lo eliminásemos?

Direccionamiento a memoria y a registro del modo Mem[Reg[R]+S]. ebx y edx es lo que queremos sumar, y guardar en eax. 4 es el factor de escala. Si lo eliminamos tendríamos.

2.2. Suma_02_S_libC

Ejercicio 1: ¿Qué error se obtiene si no se añade –lc al comando de enlazar? ¿Qué tipo de errores? (en tiempo de ensamblado, enlazado o, ejecución...).

Da el siguiente error:

```
ld: se salta el /usr/lib/libc.so incompatible mientras se busca -lc
```

```
ld: se salta el /usr/lib/libc.a incompatible mientras se busca -lc
ld: no se puede encontrar -lc
```

Es un error en el enlazado.

Ejercicio 2: ¿Qué error se obtiene si no se añade la especificación del enlazador dinámico al comando de enlazar? ¿Y si se indica como enlazador un fichero inexistente, ej. <...>.so.3 en lugar de <...>.so.2? ¿Qué tipo de error es? (en tiempo de ensamblado, enlazado, ejecución...)

```
suma_02_S_libC.o: En la función `_start':
/home/mkxv/Documentos/UGR/Segundo/EC/Practicas/Practica 3/src/sumas/suma_02_S_libC.s:26:
referencia a `printf' sin definir
/home/mkxv/Documentos/UGR/Segundo/EC/Practicas/Practica 3/src/sumas/suma_02_S_libC.s:30:
referencia a `exit' sin definir
```

Es un error de enlazado en el que no se encuentras las referencias adecuadas.

Ejercicio 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? ¿Debería obtener todos ese resultado? ¿De qué depende el resultado?

Lo que estamos haciendo es actualizar el marco de pila y poniendolo por debajo del actual. Afecta en que no se mostrará el resultado correcto. El resultado depende de donde ajustemos el marco de pila.

2.3. Suma_03_SC

Ejercicio 2: ¿Qué diferencia a hay entre los comandos Next y Step? (Pista: texto de ayuda) ¿Y entre esos comandos y su versión <...>i? Aprovechando que por primera vez incorporamos lenguaje C, poner un breakpoint en call suma y probar las cuatro variantes, anotando a dónde lleva e exactamente cada una de ellas, y por qué. (Pista: Machine Code Window)

Next ejecuta las instrucciones seguidas hasta encontrar un breakpoint, sin embargo step solo pasa a la siguiente instrucción, va de instrucción en instrucción.

2.4. Suma_04_SC

Ejercicio 2: ¿Una de esas diferencias nos hace pensar que nuestra versión ensamblador de suma no implementa exactamente un bucle for, y podría producir un resultado incorrecto para cierto tamaño de la lista... ¿Cuál? ¿Por qué?

> Como estamos trabajando con 32 bit si el tamaño supera el valor $2^{31} - 1$ se produciría un error pues supera la capacidad del registro.

2.5. suma_05_C

Ejercicio 1: Volver a probar las diferencias entre Next/Step y sus variantes <...>i aprovechando la llamada a suma. Poniendo un breakpoint en la primera linea de main, ¿qué efecto tiene cada uno del os comandos?

> El efecto es que next ejeccuta todo el main ya que lo que hace es ir ejecutando por "funciones" dentro de nuestro programa, es decir, por etiquetas, en cambio next va instrucción a instrucción.

2.6. suma_09_Casm

Ejercicio 5: El código C imprime un mensaje diciendo N*(N-1)/2=, pero luego calcula (SIZE-1) * (SIZE/2). ¿Cuál es la fórmula correcta?

> La primera es la correcta, que se llama sumatoria de gauss..

3. Popcount:

El código del popcount es el siguiente, solo pondré las funciones ya que el main y listas son comunes para todos los alumnos.

```
////////////////////////////////////////////////////////////////////////
// Popcount1:
// ----------------------------------------------------------------
// En cada iteracción analizamos un elemento del array. Una vez
// lo tengamos en la variable X usamos otro bucle para acumular en una variable
// local "res" si existe o no un 1 en la poss final del numero. Tras esto
// introducimos ceros por la izquierda desplazando todos sus unos, así, para
// 01101 pasará a 00110 -> 00011 -> 00001 -> 00000.
//
```

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

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

   return res;
}


////////////////////////////////////////////////////////////////////////
// Popcount2:
// --------------------------------------------------------------------
// Misma idea que antes pero ahora la tener un do .. while nos
// da la oportunidad de parar el buble interno un poco antes, una vez ya se
// hayan contado todos los unos. Si tuviesemos que vontar: 000100, solo
// desplazaría tres veces y acaba con ese número, en la versión anterior
// continuaría con el for.
//
int popcount2 (unsigned* array, int len){
  int  i,     res=0;
  unsigned x;

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


////////////////////////////////////////////////////////////////////////
// Popcount3:
// --------------------------------------------------------------------
// El procedimiento es parecido al del pcount2 pero la
//  diferencia principal es por cada iteracción lo que hago es como hacer una
//  mask = 1 pero 4 veces, 4x8 = 32, por lo que nos ahorramos muchas iteraciones.
//  Por ejemplo, si tenemos:
//        01011011|01011100|01100001|11101010
//      & 00000001|00000001|00000001|00000001
//  Con x>>1 lo que hago es meterle un 0 a 0 para seguir contando en la siguiente
//  iteracción.
//
```

```
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\t" // seguir mientras que x!=0
      "shr %[x]       \n\t" // Desplaza todos los bits a la derecha
      "adc $0x0, %[r] \n\t" // Si se activa el bit de signo sumo carry+result
      "test %[x], %[x]\n\t" // Se hace para activar o no el ZF
      "jnz ini3"            // Salta si no es cero
      :[r] "+r" (result)
      :[x] "r" (x));

  }
  return result;
}

////////////////////////////////////////////////////////////////////////
// Popcount4:
// ---------------------------------------------------------------------
//  El código se basa en aplicar sucesivam mente (8 vec ces) la
//  másca ara 0x0101... a a cada eleme nto, para ir ac cumulando
//  lo os bits de cada byte en una nueva variable val
//
int popcount4 (unsigned* array, int len){
  int  i, j, result=0;
  unsigned x, val;

  for (i=0; i<len; i++){
    val = 0;
    x=array[i];
    for (j=0; j<8; j++) {
      val += x & 0x01010101;
      x>>=1;
    }
    val+=(val>>16);
    val+=(val>>8);
    val&=0xFF;
    result += val;
  }
    return result;
}

////////////////////////////////////////////////////////////////////////
// Popcount5:
// ---------------------------------------------------------------------
// En esta versión se utilizan los registros de 64 para paralelizar
// la suma. La explicación completa la aporta Antonio Cañas como un archivo
// plano de texto que se ve graficamente.
//
```

```
int popcount5 (unsigned* array, int len){
  int i, val, result=0;
  int SSE_mask[] = { 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f, 0x0f0f0f0f};
  int SSE_LUTb[] = { 0x02010100, 0x03020201, 0x03020201, 0x04030302};
  //                    3 2 1 0,    7 6 5 4,   1110 9 8,  15141312
  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" // Copio el array dos veces
        "movdqa  %%xmm0, %%xmm1      \n\t" // Tanto en 0 como en 1
        "movdqu    %[m], %%xmm6      \n\t" // muevo la maskara al 6
        "psrlw       $4, %%xmm1      \n\t" // Cambiamos a doubleword
        "pand    %%xmm6, %%xmm0      \n\t" // Aplico AND con la mascara
        "pand    %%xmm6, %%xmm1      \n\t" // Igual

        "movdqu    %[l], %%xmm2      \n\t" // Como queríamos dos copias
        "movdqa  %%xmm2, %%xmm3      \n\t" // ccopio en 2 y en 3
        "pshufb  %%xmm0, %%xmm2      \n\t" // Intercambio los registros
        "pshufb  %%xmm1, %%xmm3      \n\t" // Igual

        "paddb   %%xmm2, %%xmm3      \n\t" // vector de bytes
        "pxor    %%xmm0, %%xmm0      \n\t" // pongo todo a 0 xor
        "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;
}

//////////////////////////////////////////////////////////////////////
// Popcount6:
// ------------------------------------------------------------------
// Aplica la instrucción popcnt del repertorio SSSE3 cuyo
// funcionamiento interno desconozco pero buscando he
// encontrado algo de información pero nada que explique
// que hace popcnt internamente.
//
// http://danluu.com/assembly-intrinsics/
//

int popcount6 (unsigned* array, int len){
  int i, val, result = 0;
  unsigned x;
```

```
  for(i=0; i<len; i++){
    x = array[i];
    asm("popcnt %[x], %[val]"
        : [val] "=r" (val)
        :   [x] "r" (x)
    );
    result += val;
  }
  return result;
}

////////////////////////////////////////////////////////////////////////
// Popcount7:
// --------------------------------------------------------------------
//  Es la misma idea que antes pero leyendo del buff de dos
//  en dos.
//
int popcount7 (unsigned* array, int len){
  int i, val, result = 0;
  unsigned x1,x2;
  if( len & 0x1)
    printf("leer 64b y len impar?\n");
  for(i=0; i<len; i+=2){
    x1 = array[i];
    x2 = array[i+1];
    asm("popcnt %[x1], %[val] \n\t"
        "popcnt %[x2], %%edi  \n\t"
        "add    %%edi, %[val] \n\t"
        : [val] "=&r" (val)
        :  [x1]  "r"  (x1),
           [x2]  "r"  (x2)
        : "edi");
    result += val;
  }
  return result;
}
```

4. Parity.

El código del parity es el siguiente, solo pondré las funciones ya que el main y listas son comunes para todos los alumnos.

```
////////////////////////////////////////////////////////////////////////
// Parity1:
// --------------------------------------------------------------------
// Por cada elemento leido en un segundo for hago un xor y
// desplazo. Es la misma idea que el popcpunt1.
//
```

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

  for (i = 0; i < len; i++){
    x = array[i];
    val = 0;
    for (j = 0; j < 8 * sizeof(unsigned); j++){
      val ^=x & 0x1;
      x >>= 1;
    }
    res += val;
  }
  return res;
}

//////////////////////////////////////////////////////////////////////
// Parity2:
// ----------------------------------------------------------------
// Exactamente la misma idea que antes pero grcias al while(x)
// si el elemento es 0 paro de contar, o si ya lo he contando
// entero.
//
int parity2(unsigned* array, int len){
int  i,     res=0, val;
unsigned x;

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

//////////////////////////////////////////////////////////////////////
// Parity3:
// ----------------------------------------------------------------
// La misma idea pero realizo la máscara tras realizar los
// desplazamientos y el xor.
//
int parity3(unsigned* array, int len){
  int  i,     res=0, val;
  unsigned x;

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

```
    val = 0;
    while(x) { val ^= x ; x >>= 1; }
    res += val & 0x1;
  }
  return res;
}

////////////////////////////////////////////////////////////////////////
// Parity4:
// -------------------------------------------------------------------
// Aparentemenete es una traducción literal del parity anterior.
// Gracias a jne lo que hacemos es el xor y el deesplazamiento
// siempre y cuando ZF no está activado.
//
int parity4(unsigned* array, int len){
  int  i,     res=0, val;
  unsigned x;

  for (i = 0; i < len; i++){
    x = array[i];
    val = 0;
    asm("\n"
    "init4:                 \n \t"
    " xor %[x],%[v]         \n \t"
    " shr $1,%[x]           \n \t"
    " jne   init4           \n \t"
    : [v] "+r" (val)                // e/s:           inicialmente 0, salilda
valor
    : [x] "r" (x)                   // entrada: valor elemento
    );

    res += val & 0x1;
  }
  return res;
}

////////////////////////////////////////////////////////////////////////
// Parity5:
// -------------------------------------------------------------------
// La idea es someter al elemento del array a XOR y desplazamientos
// sucesivos cada vez a mitad de distancia (16 6, 8, 4, 2, 1)
// hasta que finalmente se hace sencillamente x^=x>>1
//
int parity5(unsigned* array, int len){
  int  i,j;
  unsigned x;
  int result=0, val;
  for (i=0; i < len; i++){
    x = array[i];
    for(j = 16; j >= 1; j /= 2)
      x ^= x >> j;
    x &= 0x01010101;
```

```
    x &= 0xFF ;
    result += x;
  }
  return result;
}
```

5. Resultados de las pruebas.

5.1. Pruebas del popcount.

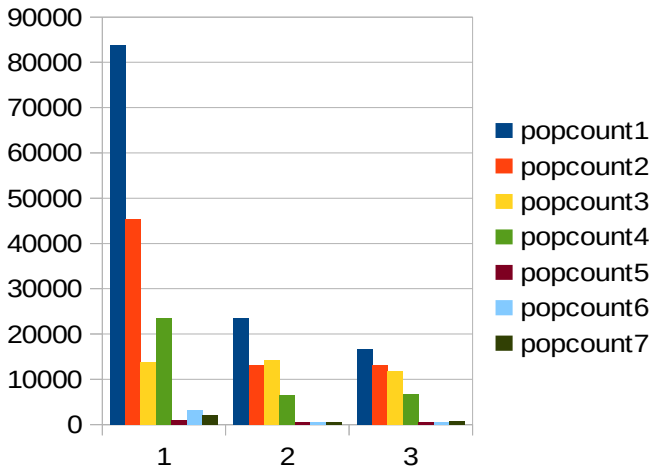
POPCOUNT:	gcc -m32 popcount.c -o popcount -O<n> for ((i=0; i<11; i++)); do echo $i; ./popcount; done|pr -11 -l 20 -w 80

Optimizacion -O0	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Media*
popcount1 (lenguaje C - for)	148003	84804	84685	81825	82366	82064	84760	88067	81390	86416	81748	83812,5
popcount2 (lenguaje C – while)	44938	44911	44828	44848	44880	44871	45014	46830	44865	46709	44885	45264,1
popcount3 (lenguaje ASM - while)	13703	13696	13697	13739	13743	13695	14324	13806	14137	13834	13734	13840,5
popcount4 (l.CD:APP 3.49 - group 8b)	23300	23270	23287	23301	23287	23283	24395	23328	23275	23379	23342	23414,7
popcount5 (ASM SSE3 - pshufb 128b)	1022	1018	1010	1009	1014	1010	1126	1007	1395	1026	1002	1061,7
popcount6 (ASM SSE4 - popcount 32b)	3133	3132	3399	3127	3134	3126	3490	3140	4037	3140	3146	3287,1
popcount7 (ASM SSE4 – opcnt 2x32b)	1973	1972	1989	1979	1975	1977	2193	1972	2787	1976	1981	2080,1

Optimizacion -O1	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Media*
popcount1 (lenguaje C - for)	29456	23392	23460	23461	23353	23276	23359	23405	23982	23444	23241	23437,3
popcount2 (lenguaje C – while)	13542	12556	12863	13423	13441	13392	13430	13465	12815	12447	13338	13117
popcount3 (lenguaje ASM - while)	14119	14076	14161	14133	14064	14099	14058	14056	14152	14592	14056	14144,7
popcount4 (l.CD:APP 3.49 - group 8b)	6352	6535	6414	6528	6587	6379	6321	6343	6445	6493	6422	6446,7
popcount5 (ASM SSE3 - pshufb 128b)	456	475	574	572	456	458	458	459	475	458	462	484,7
popcount6 (ASM SSE4 - popcount 32b)	520	540	601	623	523	521	534	526	543	523	536	547
popcount7 (ASM SSE4 – opcnt 2x32b)	590	537	584	614	546	541	548	538	543	532	543	552,6

Optimizacion -O2	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Media*
popcount1 (lenguaje C - for)	17702	16522	16798	16549	17860	16727	17148	15645	17561	15726	16524	16706
popcount2 (lenguaje C – while)	16646	16429	12682	12710	12310	12137	13196	13486	13026	13176	11811	13096,3
popcount3 (lenguaje ASM - while)	11826	11781	11972	11933	11801	11544	11544	11620	11589	11720	11548	11705,2
popcount4 (l.CD:APP 3.49 - group 8b)	6952	6711	6831	6825	6971	6665	6693	6731	6711	6670	6664	6747,2
popcount5 (ASM SSE3 - pshufb 128b)	472	528	477	452	527	451	452	447	458	451	455	469,8
popcount6 (ASM SSE4 - popcount 32b)	531	528	544	531	537	526	578	540	575	534	536	542,9
popcount7 (ASM SSE4 – opcnt 2x32b)	749	744	740	859	716	708	714	717	720	714	716	734,8

POPCOUNT:	**-O0**	**-O1**	**-O2**
popcount1	83812,5	23437,3	*16706*
popcount2	45264,1	13117	*13096,3*
popcount3	13840,5	14144,7	*11705,2*
popcount4	23414,7	6446,7	*6747,2*
popcount5	1061,7	484,7	*469,8*
popcount6	3287,1	547	*542,9*
popcount7	2080,1	552,6	*734,8*



5.2. Pruebas del parity.

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

Optimizacion -O0	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Media*
parity1 (lenguaje C - for)	142015	86347	86088	86372	86245	86762	89300	86088	86621	86428	86393	86664,4
parity2 (lenguaje C - while)	47445	47257	47704	47250	47303	53955	47354	47326	47364	47501	47419	48043,3
parity3 (1.33:CS:APP 3.22 - mask final)	42031	41924	42716	43970	41904	41992	41930	41902	41954	50662	41971	43092,5
parity4 (ASM - for)	13052	13051	14542	13053	13048	13068	13075	13109	13051	17253	13337	13658,7
parity5 (ASM - SSSE3)	15508	15513	17278	15517	15527	15537	15537	15539	15526	15543	15538	15705,5

Optimizacion -O1	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Media*
parity1 (lenguaje C - for)	47554	17404	16406	16022	15949	16018	16210	15963	16806	16107	17076	16396,1
parity2 (lenguaje C - while)	24839	12221	12484	12089	11363	10801	10724	12147	12024	12208	12150	11821,1
parity3 (1.33:CS:APP 3.22 - mask final)	15248	7913	8225	8304	7749	7754	7911	7899	8028	7865	7844	7949,2
parity4 (ASM - for)	17824	10250	10209	13881	10337	10343	10254	10266	10720	10514	10420	10719,4
parity5 (ASM - SSSE3)	6634	6138	6033	6030	6035	5974	5976	6040	6005	6067	5999	6029,7

Optimizacion -O2	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*Media*
parity1 (lenguaje C - for)	19822	19942	19196	19166	18969	19148	19168	19144	18961	19060	19068	19182,2
parity2 (lenguaje C - while)	13067	13309	12523	12646	11874	12705	13452	13438	15392	12544	12640	13052,3
parity3 (1.33:CS:APP 3.22 - mask final)	8176	7760	7945	8040	8123	7994	7778	8033	7675	7967	7885	7920
parity4 (ASM - for)	7579	7545	7508	7801	9340	7541	7533	7684	7470	7494	7518	7743,4
parity5 (ASM - SSSE3)	5439	5147	4814	4972	5215	4888	4818	4844	5052	4646	5498	4989,4

PARITYS:	-O0	-O1	-O2
parity1	86664,4	16396,1	*19182,2*
parity2	48043,3	11821,1	*13052,3*
parity3	43092,5	7949,2	*7920*
parity4	13658,7	10719,4	*7743,4*
parity5	15705,5	6029,7	*4989,4*