Cognoms: . . . . . Nom: . . . . . .

1er Control Arquitectura de Computadors **Curs 2012-2013 Q1**

Problema 1. (3 puntos)

Disponemos de un computador con un procesador a 3.0 Ghz. Un programa de prueba P realiza 300 millones de operaciones de punto flotante y ejecuta 1200 millones de instrucciones que se distribuyen de la siguiente forma

	punto flotante	**enteras**	**memoria**
Nunero de instrucciones	500 millones	300 millones	400 millones
CPI	4	2	7

a) **Calcula** el tiempo de ejecución del programa P.

Ciclos = $500x10^6$ i * 4 c/i + $300x10^6$ i * 2 c/i + $400x10^6$ c * 7c/i = $5,4x10^9$ ciclos

Texe = $5,4x10^9$ ciclos / $3x10^9$ ciclos/s = 1,8 segundos

b) **Calcula** el CPI del programa P.

CPI = $5,4x10^9$ ciclos / $1,2x10^9$ i = 4,5 c/i

c) **Calcula** el rendimiento en MIPS y MFLOPS de P.

MIPS = $1200x10^6$ instr / (1,8 s *10^6) = 666,7 MIPS

MFLOPS = $300x10^6$ ops / (1,8 s *10^6) = 166,7 MFLOPS

El fabricante del procesador está estudiando una modificación en el mismo que supondría una ganancia de 1,2 en las instrucciones de coma flotante, una ganancia de 0,75 en las instrucciones de enteros y una ganacia de 1, 5 en las instrucciones de memoria.

d) ¿Es beneficiosa esta mejora cuando ejecutamos P? Justificad la respuesta.

ciclos CF = $500x10^6$ * 4 = $2000x10^6$ ciclos --> 37%
ciclos ENT = $300x10^6$ * 2 = $600x10^6$ ciclos --> 11,1%
ciclos MEM = $400x10^6$ * 7 = $2800x10^6$ ciclos --> 51,9%

Ganancia = 1/(0,37/1,2+0,111/0,75+0,519/1,5) = 1,25

El nuevo procesador es claramente más rápido.

Esta CPU, tiene una carga capacitiva equivalente de 15 nF (nanofaradios), y una corriente de fugas de 10 A y funciona a un voltaje de 1,2 V.

e) **Calcula** la potencia media debida a fugas, la debida a conmutación y la total para el programa P.

P fugas = I*V = 10 A * 1,2 V = 12W
Pconmutacion = $C*V^2*f = 15x10^{-9}$ F * $(1,2\ V)^2$ * $3x10^9$ Hz = 64,8 W
Ptotal = 12 W + 64,8 W = **76,8 W**

Este computador está formado por los componentes mostrados en la tabla siguiente. La tabla también muestra el numero de componentes de cada tipo y el tiempo medio hasta fallo (MTTF) de cada componente.

Componente	Fuente alimentación	CPU	Ventilador CPU	Placa base	DIMMs	Discos duros	Tarjetas graficas
Nº	1	1	1	1	4	2	2
MTTF (horas)	100.000	1.000.000	100.000	200.000	1.000.000	125.000	500.000

El tiempo medio para reemplazar un componente que ha fallado (*mean time to repair*) es de 5 horas y la probabilidad de fallo sigue una distribución exponencial.

f) **Calcula** el tiempo medio hasta fallos del hardware (MTTF), el tiempo medio entre fallos (MTBF) y la disponibilidad del sistema.

MTTF = 1/(1/100000+1/1000000+1/100000+1/200000+4/1000000+2/125000+2/500000) = **20000 horas**

MTBF = MTTF + MTTR = **20005 horas**

disponibilidad = 20000 h / 20005 h * 100 = **99,975%**

Cognoms: . . . . . Nom: . . . . . .

1er Control Arquitectura de Computadors **Curs 2012-2013 Q1**

Problema 2. (3 puntos)

Dado el siguiente código escrito en C:

```
typedef struct {
  int a;
  char b;
  char c;
  int *d;
  short e;
} s1;

typedef struct {
  s1 f;
  short g[7];
} s2;

short F(s1 *alto, int bola);
int examen(s1 uno, char dos, s2 *tres){
  int i, j;
  s2 M[10][10];
  ...
}
```

a) **Dibuja** como quedarían almacenadas en memoria las estructuras s1 y s2, indicando claramente los deplazamientos respecto al inicio y el tamaño de todos los campos.

s1 (14 bytes)

4	a	+0
1	b	+4
1	c	+5
2	---	
4	d	+8
2	e	12

s2 (28 bytes)

14	f	+0
2	g[0]	+14
2	g[1]	
8	---	
2	g[7]	

b) **Dibuja** el bloque de activación de la función examina, indicando claramente los desplazamientos relativos al registro EBP necesarios para acceder a los parámetros y a las variables locales.

examina

	Reg	
4	i	-2808
4	j	-2804
2800	M	-2800
4	ebp old	<--%ebp
4	ret	
16	uno	8
1	dos	24
3	---	
4	tres	28

c) **Traduce** la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examina:

```
M[i][7].f.c = dos;
```

```
imull $28, -2808(%ebp), %eax
addl $7*28, %eax
movb 24(%ebp), %dl
movb %dl, 5-2800(%ebp, %eax)
```

d) **Traduce** la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examina:

```
tres->g[1]=F(&uno, 47);
```

```
leal -2804(%ebp), %eax
pushl %eax
pushl $47
leal 8(%ebp), %eax
pushl %eax
call F
addl $12, %esp
movl 28(%ebp), %ebx
movw %ax, 12(%ebx)
```

e) **Traduce** la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examina (este código se ha de traducir a 8 instrucciones x86):

```
M[0][0].f.a = uno.a;
M[1][0].f.c = uno.c;
M[0][1].f.d = uno.d;
M[1][1].f.e = uno.e;
```

```
movl 8(%ebp), %eax
movl %eax, -2800(%ebp)
movb 8+5(%ebp), %al
movb %al, -2800+280+5(%ebp)
movl 8+8(%ebp), %eax
movl %eax, -2800+28+8(%ebp)
movw 8+12(%ebp), %ax
movw %ax, -2800-280+28+12(%ebp)
```

Cognoms: . . . . . Nom: . . . . . .

1er Control Arquitectura de Computadors **Curs 2012-2013 Q1**

Problema 3. (4 puntos)

Dada la siguiente subrutina escrita en C:

```
void sumar (int V[], int W[])
{
  int i;
  for (i=0; i<1000000; i++)
    V[i] = V[i]+W[i];
}
```

a) **Traduce** la subrutina a lenguaje ensamblador x86 sin hacer ningún tipo de optimización (salvo poner la variable i en un registro).

```
sumar:   pushl %ebp
         movl %esp, %ebp             ; enlace dinámico
         subl $4, %esp               ; i en -4(%ebp)
         pushl %esi
         pushl %edi
         movl 8(%ebp), %esi
         movl 12(%ebp), %edi
         xorl %eax, %eax             ; i=0
for:     cmpl $1000000, %eax
         jge endfor
         movl (%edi,%eax,4), %ecx    ; %ecx<-W[i]
         addl %ecx, (%esi,%eax,4)    ; V[i] <- V[i] + W[i]
         incl %eax
         jmp for
endfor:  popl %edi
         popl %esi
         movl %ebp,%esp              ; deshacer enlace dinámico
         popl %ebp
         ret
```

b) Suponiendo que todas las instrucciones x86 tardasen 2 ciclos en ejecutarse en un procesador que funciona a 2 GHz, **calcula** los MIPS y el tiempo de ejecución de la subrutina.

Consideraremos sólo las instrucciones del interior del bucle, ya que el impacto de las otras es despreciable.
MIPS = $2x10^9$ ciclos / 2 ciclos/inst / 10^6 = 1000 MIPS
N = $6*1000000=6x10^6$

T=N*CPI/F= $6x10^6$ inst * 2 c/inst / $2x10^9$ c/s = $6x10^{-3}$ segundos

En el modo de 32 bits que estudiamos en este curso, las instrucciones multimedia SSE usan los 8 registros de 128 bits %xmm0 -%xmm7. La instrucción **paddd op1, op2** realiza la operación **op2 = op2 + op1**. Los dos operandos son de 128 bits, y la instrucción realiza la suma de 4 enteros de 32 bits empaquetados en cada uno de los operandos. La instrucción **movdqa op1, op2** realiza la operación **op2 <- op1**, donde op1 y op2 son operandos de 128 bits.

c) **Optimiza** la subrutina usando estas instrucciones.

```
sumar:   pushl %ebp
         movl %ebp, %esp              ; enlace dinámico
         subl $4, %esp                ; i en -4(%ebp)
         pushl %esi
         pushl %edi
         movl 8(%ebp), %esi
         movl 12(%ebp), %edi
         xorl %eax, %eax              ; i=0
for:     cmpl $1000000, %eax
         jge endfor
         movdqa (%edi,%eax,4), %xmm0  ; %xmm0<-W[i:i+3]
         paddd %xmm0, (%esi, %eax, 4); V[i:i+3]<-V[i:i+3] + W[i:i+3]
         addl $4, %eax
         jmp for
endfor:  popl %edi
         popl %esi
         movl %ebp,%esp               ; deshacer enlace dinámico
         popl %ebp
         ret
```

d) Suponiendo que todas las instrucciones x86 tardasen 2 ciclos en ejecutarse en un procesador que funciona a 2 GHz, a excepción de las instrucciones multimedia, que tardan 4 ciclos, **calcula** los MIPS en el código optimizado y la ganancia respecto al código original en tanto por ciento.

CPI = (4 * 2 + 2 * 4) / 6 = 2,66 ciclos/i
MIPS = $2x10^9$ ciclos / 2,66 ciclos/inst / 10^6 = 751,8 MIPS
N = 6 * 250000 = $1,5x10^6$

T=N*CPI/F= $1,5x10^6$ * 2,66 c/inst / $2x10^9$ c/s = $2x10^{-3}$ segundos

ganancia = Ta/Tb = $6x10^{-3}$ segundos / $2x10^{-3}$ segundos = 3
Ganancia = (Ta/Tb -1) * 100 = 200 %

e) La opción SIMD es más eficiente que la original, pero sin embargo tiene menos MIPS. Justifica este hecho.

Los MIPS miden el número de instrucciones por segundo, que no es una medida de rendimiento. En este caso, la versión SIMD ejecuta un 30% menos de instrucciones por segundo, pero como ejecuta la cuarta parte de instrucciones es mucho más significativo este último dato.