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

1er Control Arquitectura de Computadors **Curs 2010-2011 Q2**

Problema 1. (4 puntos)

Un 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	Disco duro	SSD *
Nº	1	1	1	1	4	1	1
MTTF (horas)	125.000	1.000.000	100.000	200.000	1.000.000	100.000	500.000

* SSD: Solid State Disc (disco de estado solido)

El tiempo medio para reemplazar un componente que ha fallado (*mean time to repair*) es de 10 horas.

a) **Calcula** el tiempo medio hasta fallos del sistema (MTTF).

MTTF = 1/(1/125000+1/1000000+1/100000+1/200000+4/1000000+1/100000+1/500000) = **25000 horas**

b) **Calcula** el tiempo medio entre fallos (MTBF).

MTBF = MTTF + MTTR = **25010 horas**

c) **Calcula** la disponibilidad del sistema.

25000 h / 25010 h * 100 = **99,96%**

La CPU de este sistema tiene una superficie de 200 mm^2 y se fabrica en una oblea de silicio con una superficie útil de 64.000 mm^2. El coste energético de la oblea y el proceso de impresión y verificación de los dados es de 25.600 MJoules. Durante este proceso el factor de yield es del 80%. El coste de empaquetado y test final de las CPUs es de 25 MJoules por dado y el yield final de las CPUs después del test final es del 87,5%.

d) **Calcula** el coste energético de un dado (antes del empaquetado y testeo final).

64.000 mm^2 / 200 mm^2/dado = 320 dados
320 dados x 0,8 dados buenos/dado = 256 dados buenos
25.600 MJ/oblea / 256 dados/oblea = **100 MJoules / dado**

e) **Calcula** el coste energético final de una CPU.

coste por oblea con empaquetado y testeo = 256 dados x (100 MJ/dado + 25 MJ/dado) = 32.000 MJ/oblea
CPUs funcionales = 256 dados * 0,875 = 224 CPUs
coste por CPU = 32.000 MJ/oblea / 224 CPUs/oblea = **142,9 Mjoules/CPU**

En este sistema tenemos instalado el entorno usado en el laboratorio de AC y hemos medido que un programa se ha ejecutado en 2 segundos usando $6x10^9$ ciclos y ha ejecutado $4,8x10^9$ instrucciones

f) **Calcula** el CPI del programa y la frecuencia de la CPU (usa el prefijo del sistema internacional más adecuado).

CPI = $4,8x10^9$ instrucciones / $6x10^9$ ciclos = **1,25 c/i**
Frec = $6x10^9$ ciclos / 2 segundos = $3x10^9$ ciclos/segundo = **3 GHz**

El tiempo de ejecución calculado anteriormente se corresponde al tiempo de CPU (usuario + sistema). Usando el comando "time" de linux hemos obtenido que el tiempo de CPU representa solo el 20% del tiempo total del programa (wall time). El 80% restante es tiempo de entrada/salida (accesos al disco duro concretamente). Cada acceso al disco duro tarda 8 milisegundos, mientras que si los datos estuviesen en el disco SSD cada acceso tardaría 10 microsegundos.

g) **Calcula** la ganancia en la parte de entrada salida si los datos del programa estuviesen en el SSD en lugar de el disco duro.

Ganancia = $8x10^{-3}$ segundos /acceso / $10x10^{-6}$ segundos /acceso = **800**

h) **Calcula** la ganancia total en el programa a partir de la ganancia en entrada salida.

Ganancia = 1/((1-fm)+fm/gm) = 1/((1-0,8)+0,8/800) = **4,975**

A pleno rendimiento la CPU tiene una carga capacitiva equivalente de 16 nF (nanoFaradios), funciona a un voltaje de 1,25 V y una frecuencia de 2GHz. Se ha determinado que esta CPU tiene una corriente de fugas de 8 A.

i) **Calcula** la potencia media debida a fugas, la debida a conmutación y la total cuando la CPU esta a pleno rendimiento.

P fugas = I*V = 8 A * 1,25 V = 10W
Pconmutacion = $C*V^2*f$ = $16x10^{-9}$ F * $(1,25\ V)^2$ * $2x10^9$ Hz = 50 W
Ptotal = 10 W + 50 W = **60 W**

Las CPUs actuales, cuando no están a plena carga, reducen el voltaje y la frecuencia para ahorrar energía. En modo bajo consumo nuestra CPU consume tan solo 20 W. Sabemos que nuestro sistema está 4 horas diarias en modo alto rendimiento, 10 horas en modo bajo consumo y el resto esta totalmente apagado (consumo 0 W).

j) **Calcula** la energía que ahorramos cada día gracias a la reducción de frecuencia y voltaje de la CPU (usa el prefijo del sistema internacional más adecuado).

El ahorro solo se produce en los periodos de bajo rendimiento

Ahorro = tiempo * reducción potencia = 10 h * 3600 s/h * (60 W - 20 W) = $1,44x10^6$ Joules = **1,44 MJoules**

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

1er Control Arquitectura de Computadors **Curs 2010-2011 Q2**

Problema 2. (3 puntos)

Dado el siguiente código escrito en C:

```
typedef struct {
  int a;
  char b;
  char c;
  double d;
} s1;

typedef struct {
  short e[5];
  s1 f;
} s2;

short F(s1 *alto, int bola, char *cola);
int examina(s1 uno, char dos, s2 *tres){
  char vl1;
  int vl2;
  ...
}
```

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 (16 bytes)

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

s2 (28 bytes)

	e[0]	+0
10	. .	
	e[4]	
2	---	
16	f	+12

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.

1	vl1	-8
3	---	
4	vl2	-4
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:

```
vl1=dos+uno.b;
```

```
movb 24(%ebp),%al
addb 12(%ebp),%al
movb %al,-8(%ebp)
```

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

```
tres->e[1]=F(&uno, vl2, &uno.c);
```

```
leal 13(%ebp), %eax
pushl %eax
pushl -4(%ebp)
leal 8(%ebp), %eax
pushl %eax
call F
addl $12, %esp
movl 28(%ebp), %ecx
movw %ax, 2(%ecx)
```

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

```
if (vl2 > 0)
    vl2 = tres->f.a;
```

```
cmpl $0,-4(%ebp)
jle fin
movl 28(%ebp),%ecx
movl 12(%ecx),%ecx
movl %ecx,-4(%ebp)
fin:
```

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

1er Control Arquitectura de Computadors **Curs 2010-2011 Q2**

Problema 3. (3 puntos)

Dado el siguiente código escrito en C:

```
int Exa(int v[], int x);
int XProb3(int v[], int *p, int m){
  int i;
  for (i=0; i<1000000; i++)
    v[i] += Exa(v, *p);
  return *p + m;
}
```

a) **Dibuja** el bloque de activación de de la subrutina Xprob3.

b) **Traduce** a ensamblador del x86 la subrutina Xprob3.

```
Xprob3: pushl %ebp
        movl %esp,%ebp
        subl $4, %esp
        pushl %esi
        pushl %ebx
        movl 8(%ebp),%ebx
        xorl %esi,%esi
for:    cmpl $1000000, %esi
        jge endfor
        movl 12(%ebp), %eax
        pushl (%eax)
        pushl %ebx
        call Exa
        addl $8, %esp
        addl %eax, (%ebx, %esi, 4)
        incl %esi
        jmp for
endfor: movl 12(%ebp), %eax
        movl (%eax), %eax
        addl 16(%ebp), %eax
        popl %ebx
        popl %esi
        movl %ebp, %esp
        popl %ebp
        ret
```

Bloque activación