COGNOMS: . . . . . NOM: . . . . . .

2on Control Arquitectura de Computadors **Curs 2016-2017 Q1**

- **Temps: 13:15 a 15:15**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 1. (3 puntos)

Dado el siguiente código escrito en C:

```
typedef struct {
  char a[3];
} s1;
char F(s1 p1[2], char p2, s1 p3){
  s1 vl1[2];
  char vl2;
  s1 vl3;
  ...
}
```

a) **Dibuja** como quedaría almacenada en memoria la estructura s1 y el bloque de activación de la función F, indicando claramente los desplazamientos y el tamaño de todos los campos.

s1 (3 bytes)

1	a[0]	+0
1	a[1]	+1
1	a[2]	+2

F

3	vl1[0]	-12(%ebp)
3	vl1[1]	-9(%ebp)
1	vl2	-6(%ebp)
3	vl3	-5(%ebp)
2	---	-2(%ebp)
4	ebp old	0(%ebp)
4	ret	4(%ebp)
4	p1	8(%ebp)
1	p2	12(%ebp)
3	---	
3	p3	16(%ebp)
1	---	

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

`return F(vl1,vl2,p3);` `//` **Nota: los char se devuelven en %al**

```
pushl 16(%ebp)
pushl -6(%ebp)
leal -12(%ebp), %eax
pushl %eax
call F
addl $12, %esp
movl %ebp, %esp
popl %ebp
ret
```

COGNOMS: . . . . . NOM: . . . . . .

2on Control Arquitectura de Computadors **Curs 2016-2017 Q1**

- **Temps: 13:15 a 15:15**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 2. (3,5 puntos)

Se quiere diseñar una memoria cache de datos con políticas de escritura *write through* y *write NO allocate*:

Se han obtenido por simulación las siguientes medidas para un determinado programa:

- porcentaje de escrituras (sobre el total de accesos): 15%
- tasa de aciertos: 0,9

La memoria cache es de mapeo directo y se leen etiquetas y datos en paralelo. En caso de fallo de lectura, el bloque de MP se escribe en la MC y posteriormente el dato se envía a la CPU desde la MC. El tiempo de acceso (Tsa) a memoria cache (MC) es de 10 ns tanto para lectura como escritura. El tiempo de acceso a memoria principal (MP) para escribir una palabra es de 90 ns. Para leer o escribir un bloque en la MP se emplean 130 ns.

a) **Calcula** el tiempo empleado en realizar 1000 accesos consecutivos

Lectura: 0,85*1000 = 850 accesos

Número de aciertos: 0,9 aciertos/acc*0,85*1000 acc = 765 aciertos

Número de fallos: 0,1 fallos/acc x 0,85*1000 acc = 85 fallos

Tiempo aciertos: 765 acc*10x10^{-9} seg/acc = 7,65*10^{-6} segundos

Tiempo fallos: 85 acc*(10+130+10)*10^{-9} seg/acc = 12,75*10^{-6} segundos

Escritura: 0,15*1000 = 150 acc. Tiempo escrituras: 150 acc*90*10^{-9} seg/acc = 13,5 * 10^{-6} segundos

Tiempo total: 7,65*10^{-6} segundos + 12,75*10^{-6} segundos + 13,5*10^{-6} segundos = 33,9*10^{-6} segundos

---

Forma 2 de hacerlo:

T = 1000 accesos * Tma

1000*Tma = 1000*(Tsa + m_lec * Tpf_lecturas + m_esc * Tpf_escrituras) =

= 1000*(10ns + 0,1*0,85 * (130ns + 10ns) + 0,15 * (90ns - 10ns)) =

= 1000*(10ns + 11,9ns + 12ns) = 33,9*10^{-6} segundos

Forma 3 de hacerlo:

Aciertos: Lectura - 0,85 * 0,9 * 10ns/acceso * 1000 accesos = 7,65* 10^{-6} segundos

Escrituras - 0,9 * 0,15 * 90ns/acceso * 1000 accesos = 12,15* 10^{-6} segundos

Fallos: Lectura - 0,85 * 0,1 * (10+130+10 ns/acceso) * 1000 accesos = 12,75* 10^{-6} segundos

Escrituras - 0,1 * 0,15 * 90ns/acceso * 1000 accesos = 1,35* 10^{-6} segundos

Tiempo total: 7,65*10^{-6} s + 12,15*10^{-6} s + 12,75*10^{-6} s + 1,35*10^{-6} s = 33,9*10^{-6} segundos

Dado el siguiente código escrito en ensamblador del x86:

```
        movl $0, %ebx
        movl $0, %esi
   for:
        cmpl $512*1000, %esi
        jge end

    (a) movl (%ebx, %esi, 4), %eax
    (b) addl 2*4*1024(%ebx, %esi, 4), %eax
    (c) movl %eax, 3*4*1024(%ebx, %esi, 4)

        addl $1, %esi
        jmp for
   end:
```

Sabemos que el código se ejecuta en un sistema con memoria cache y memoria virtual. La memoria virtual utiliza páginas de tamaño 4KB y disponemos de un TLB de 4 entradas y reemplazo LRU. La memoria cache de datos (únicos accesos a memoria que contemplaremos en este problema) es *Write Through + Write No Allocate*, de 2 vías con reemplazo LRU, tamaño 4 KB y 8 bytes por bloque. Responde a las siguientes preguntas:

b) **Calcula,** para cada uno de los accesos etiquetados como (a, b, c), el conjunto de la memoria cache al que se accede en cada una de las 9 primeras iteraciones del bucle

iteración	0	1	2	3	4	5	6	7	8
a	0	0	1	1	2	2	3	3	4
b	0	0	1	1	2	2	3	3	4
c	0	0	1	1	2	2	3	3	4

Calcula la cantidad de aciertos y de fallos de cache, en todo el código.

El bucle se ejecuta 512*1000 veces, y se produce el siguiente patrón de aciertos/fallos cada 2 iteraciones:
a) 1F 1A
b) 1F 1A
c) 1F 1F

Aciertos = (512 * 1000 / 2) * (1+1) = 512000

Fallos = (512 * 1000 / 2) * (1+1+2) = 1024000

c) **Indica,** para cada uno de los accesos indicados (etiquetas a, b, c), a qué página de la memoria virtual se accede en cada una de las siguientes iteraciones del bucle (recuerda que los accesos son a 4 bytes).

iteración	0	1*512	2*512	3*512	4*512	5*512	6*512	7*512	8*512	9*512
a	0	0	1	1	2	2	3	3	4	4
b	2	2	3	3	4	4	5	5	6	6
c	3	3	4	4	5	5	6	6	7	7

Calcula la cantidad de aciertos y de fallos de TLB, en todo el código.

Se falla en el TLB cada vez que se accede a una nueva página, el resto de accesos a esa página son simepre aciertos.
Hay 1024 accesos a cada página de cada instrucción (a,b, c). Cada instrucción accede a 500 páginas y en el bucle se accede a (512*1000*4+3*4*1024)/4096 = 503 páginas distintas.
a) falla al acceder las paginas 0 y 1, el resto reusa las que accede b)
b) falla al acceder la pagina 2, el resto reusa las que accede c)
c) falla al acceder las 500 paginas

Fallos = 503
Aciertos = 512*1000*3 - 503 = 1535497

COGNOMS: . . . . . NOM: . . . . . .

2on Control Arquitectura de Computadors **Curs 2016-2017 Q1**

- **Temps: 13:15 a 15:15**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 3. (3,5 puntos)

En una **CPU** ejecutamos un programa (X). Esta **CPU** está conectada a una cache de instrucciones (**$I**) y una cache de datos (**$D**), esta última con políticas de escritura **copy back** + **write allocate**. La siguiente tabla muestra algunos datos obtenidos para ambas caches al ejecutar el programa X:

Característica	$I	$D
Número de accesos a memoria por instrucción (nr)	1 ref/inst	0,5 ref/inst
Tasa de fallos (m)	10%	20%
Consumo de energía en caso de acierto (Ea)	1 nJ	1 nJ
Penalización en consumo de energía en caso de fallo al reemplazar un bloque no modificado (Epf)	20 nJ	20 nJ
Penalización en consumo de energía en caso de fallo al reemplazar un bloque modificado (EpfM)	---	40 nJ
Porcentaje de bloques modificados (pm)	0%	25%

a) **Calcula** la energía media por acceso (EmaI) consumida por la jerarquía de memoria para los accesos a instrucciones.

EmaI = Ea + m*Epf = 1 nJ/acceso + 0,1 fallos/acceso * 20 nJ/fallo = 3 nJ/acceso

b) **Calcula** la energía media por acceso (EmaD) consumida por la jerarquía de memoria para los accesos a datos.

EmaD = Ea + m*(pm*EpfM + (1-pm)*Epf) =
= 1 nJ/a + 0,2 f/a * (0,25 * 40 nJ/fallo + 0,75 * 20 nJ/fallo) = 6 nJ/acceso

Hemos medido que, en promedio, la ejecución de una instrucción consume 2 nJ (nano Joules) en el caso ideal en que los accesos a memoria no consumen nada.

c) **Calcula** la energía media consumida por la ejecución de una instrucción teniendo en cuenta la jerarquía de memoria (Eexe)

Eexec = Eid + 1*EmI + 0,5*Emd =
= 2 nJ/instr + 1 acc/instr * 3 nJ/acc + 0,5 acc/instr * 6nJ/acc = 8 nJ/Instrucción

El conjunto formado por **CPU**+**$I**+**$D** (que llamaremos *núcleo*) esta conectado a una cache de segundo nivel (**L2**) mucho mayor que las de primer nivel. El programa X ejecuta $5x10^9$ instrucciones, todos los accesos del programa X son de 4 bytes (tanto a instrucciones como datos) y los bloques de cache de **$I**, **$D** y **L2** son todos de 32 bytes.

d) **Calcula** cuantos bytes lee el *núcleo* desde **L2** y cuantos bytes escribe el *núcleo* en **L2**.

Bytes leidos = bytes $I + bytes $D =
1 a/i * 0,1 f/a * 5e9 i * 32 bytes/f + 0,5 a/i * 0,2 f/a * 5e9 i * 32 bytes/f = 32e9 bytes leidos

Bytes escritos $D = 0,5 a/i * 0,2 f/a * 0,25 escr/f * 5e9 i * 32 bytes/escr = 4e9 bytes escritos

Dado el siguiente fragmento de código:

```
for (i=0; i<N; i++)
  suma = suma + v[i];
```

Tanto el código como las variables i, N y suma se encuentran almacenados en **$I** y **$D** respectivamente. Los elementos del vector v son de 4 bytes (recuerda que los bloques de **L2** son de 32 bytes).

Hemos ejecutado 2 veces consecutivas el mismo bucle y hemos medido los ciclos de la segunda ejecución del bucle:

- Para valores de N medios (**L2** > tamaño de v > 2 veces **$D**) el bucle se ejecuta en 20*N ciclos.
- Para valores de N muy grandes (v es muchísimo mayor que la **L2**) el bucle se ejecuta en 45*N ciclos.

e) **Calcula** el tiempo de penalización (en ciclos) en caso de fallo en **L2**.

Para valores de N medios: No falla ->20 ciclos /iteracion

Para valores de N muy grandes: Cada 8 iteraciones 1 fallo

25 ciclos penalización/iteracion * 8 iteraciones/fallo = 200 ciclos penalización/fallo

A la cache **L2** le añadimos un mecanismo de *prefetch* hardware. Cuando se accede un bloque (i) se desencadena prefetch del bloque siguiente (i+1) siempre que el bloque (i+1) no se encuentre ya en la cache (en cuyo caso es innecesario hacer *prefetch*) o no haya un *prefecth* previo del bloque (i+1) pendiente de completar (en cuyo caso solo hay que esperar que se complete).

f) **Calcula** el número máximo de ciclos que puede durar un *prefetch* para que el bucle anterior se ejecute en 20*N ciclos para N muy grandes.

Se hace un prefetch cada 8 iteraciones
20 ciclos/iteracion * 8 iteraciones / prefecth = 160 ciclos/prefetch

g) **Calcula** los ciclos que tarda en ejecutarse el bucle (para N muy grande) si un *prefetch* dura los mismos ciclos que la penalización por fallo (apartado e)

200 ciclos / prefecth * 1 prefecth / 8 iteraciones = 25 ciclos/iteracion

25*N ciclos