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.

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
```

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

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									
b									
c									

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

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										
b										
c										

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

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.

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

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)

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**.

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**.

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.

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)