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

2on Control Arquitectura de Computadors **Curs 2014-2015 Q1**

- **Temps: 13:30 a 15:30**
- **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 {
  int i;
  char b;
  int j;
} Str;

int sub(Str s, Str m[4][10], char *c)
{
  int i;
  char v[10];
  ...
}
```

a) **Dibuja** el bloque de activación de la subrutina **sub** indicando claramente el tamaño y desplazamiento de cada campo respecto al registro %ebp.

b) **Traduce** a ensamblador la sentencia "`m[s.i][s.j].b = ‘a’;`" que se encuentra dentro de la rutina sub.

c) **Traduce** a ensamblador la sentencia "`while (*c != v[i]) i++;`" que se encuentra dentro de la rutina sub, suponiendo que la variable i esta almacenada en el registro %esi.

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

2on Control Arquitectura de Computadors **Curs 2014-2015 Q1**

Problema 2. (4 puntos)

a) **Define** de forma clara y concisa el comportamiento una cache con política write through + write NO allocate en caso de escritura de un byte con acierto y en caso de escritura de un byte con fallo:

b) **Define** de forma clara y concisa el comportamiento una cache con política copy back + write allocate en caso de escritura de un byte con acierto y en caso de escritura de un byte con fallo:

Dado el siguiente código escrito en C, y suponiendo que el vector v está almacenado en la dirección física 0x00000000:

```
int v[2000000];
....
for (i=0, i<1000000, i+=256)
     v[i]=v[i+1024]-v[i+3*1024];
```

Suponiendo que la memoria utiliza **páginas de tamaño 8KB** y que utilizamos un **TLB de 4 entradas (reemplazo LRU)**, responde a las siguientes preguntas:

c) Para cada uno de los accesos (a=v[i], b=v[i+1024], c=v[i+3*1024), indica a qué página física se accede en cada una de las 17 primeras iteraciones del bucle.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a																	
b																	
c																	

d) Calcula la cantidad de **aciertos** y **fallos de TLB**, en todo el bucle: . . . . . .

Dado el siguiente código escrito en C, y suponiendo que el vector v está almacenado en la dirección física 0x00000000:

```
int v[2000000];
....
for (i=0, i<1000000, i++)
     v[i]=v[i+1024]-v[i+3*1024];
```

e) Calcula la cantidad de aciertos y fallos de cache para la ejecución de todo el bucle suponiendo una cache completamente asociativa de 8 Kbytes y líneas de 32 bytes con política copy back + write no allocate.

f) Calcula la cantidad de aciertos y fallos de cache para la ejecución de todo el bucle suponiendo una cache directa de 4 Kbytes y líneas de 32 bytes con política copy back + write no allocate.

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

2on Control Arquitectura de Computadors **Curs 2014-2015 Q1**

Problema 3. (3 puntos)

a) Puede una cache asociativa de 2 vías tener una tasa de aciertos menor que una cache directa si el resto de los parámetros son iguales? Justifica la respuesta (si la respuesta es afirmativa basta con un ejemplo).

El desenrollado de un bucle es una optimización que usan a veces los compiladores, consistente en repetir el cuerpo de un bucle N veces, y reducir el número de iteraciones por un factor de N.

b) Que efecto tiene esta optimización de código en la cache de datos? Y en la de instrucciones?

Tenemos un cache de 1KB, asociativa de 4 vías, con líneas de 32 bytes y reemplazo LRU. Dado el siguiente código:

```
for (i=0; i<N; i++)
  for (j=0; j<M; j++)
    acum += v[j]*f(i);
```

Todas las variables excepto el vector v están en registros. N y M son constantes muy grandes (mayores que 2^{15}) y M es un múltiplo de 1024. Los elementos de v ocupan 8 bytes y sabemos que v está alineado a un múltiplo de 1KB. La función f(i) realiza un cálculo en función de la variable i; para simplificar el problema podemos asumir que f(i) no provoca fallos de cache ni interfiere con los elementos de v.

c) Transforma el código para minimizar los fallos de cache.

d) Añade el mínimo número de instrucciones de prebúsqueda (prefetch) para eliminar todos los fallos restantes, modificando el código si es necesario. La sintaxis de una instrucción de prebúsqueda es: `prefetch(dirección)`. Por ejemplo para prebúscar elemento 3 del vector v usaremos `prefetch(&v[3])`. El cálculo de f(i) requiere más tiempo de que necesita un prefetch.

e) Demuestra que si en una cache asociativa por conjuntos con reemplazo LRU doblamos su tamaño duplicando el número de conjuntos, la tasa de aciertos va a ser siempre igual o mayor (nunca menor), para cualquier secuencia de referencias. Es decir que siempre que es fallo en la cache mayor, lo es también en la menor.