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

2on Control Arquitectura de Computadors **Curs 2015-2016 Q1**

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

Problema 1. (5 puntos)

Un procesador de 32 bits (todos los accesos son a palabras de 4 bytes y las direcciones son también de 32 bits) está conectado a una cache de datos (CD) y una cache de instrucciones (CI) idénticas con la excepción que la cache de datos dispone del hardware necesario para gestionar las políticas de escritura y además implementa escrituras segmentadas. Ambas caches tienen bloques de 32 bytes, asociatividad 4 y un tamaño de cache de 32Kbytes. Estas caches de primer nivel, a su vez, están conectadas a un segundo nivel de cache unificada (L2). A lo largo del problema asumiremos que nunca se producen fallos en L2.

a) **Calcula** para la cache CI (o la CD ya que son idénticas) el número de bloques que contiene la cache, el número de vías, el numero de conjuntos y el tamaño total de la memoria de etiquetas.

Bloques	Vías	Conjuntos	Tamaño memoria etiquetas (en Kbits)

Un acceso a cualquiera de las caches (CI o CD) cuesta 1 ciclo, mover un bloque de L2 a CI o a CD (o viceversa) cuesta 10 ciclos y escribir/leer una palabra (4bytes) en/de L2 cuesta 6 ciclos. Para la implementación de la cache de datos se están barajando 2 alternativas: *Copy Back + Write Allocate* y *Write Through + Write NO Allocate*.

b) **Rellena** para la cache de datos (CD) la siguiente tabla para los casos descritos en la columna **Caso**.
(como ejemplo 3 casos para *Write Through + Write NO Allocate* ya estan rellenados)
Si para alguna de las caches, algún caso no puede darse, pon un guión (---) en las 5 columnas.

			Copy Back + Write Allocate					Write Through + Write NO Allocate				
Caso			**ECD**	**LL2**	**EL2**	**D**	**Ciclos**	**ECD**	**LL2**	**EL2**	**D**	**Ciclos**
lectura	hit											**1**
	miss	NR										
		~~RBM~~										
		RBM						---	---	---	---	---
escritura	hit											
	miss	NR										
		~~RBM~~										
		RBM						---	---	---	---	---

Nomenclatura de las columnas:
Caso: lectura o escritura, que puede ser hit o miss, y en caso de miss, puede que no se reemplaze ningún bloque (NR), que se reemplace un bloque modificado (RBM) o que se reemplace un bloque no modificado (~~RBM~~).
ECD: numero de bytes que la CPU escribe en la CD **(si no escribe en CD no pongas nada)**.
LL2: numero de bytes que se transfieren de L2 a CD **(si no se lee de L2 no pongas nada)**.
EL2: numero de bytes que se escriben en L2 **(si no se escribe en L2 no pongas nada)**.
D: indica si el *dirty bit* (D) se pone a 1 o a 0 **(si D no se modifica no pongas nada)**.
Ciclos: indica los ciclos totales (tiempo de servicio) que tarda cada caso.

Finalmente se ha optado por una cache de datos *Copy Back + Write Allocate*. En este procesador se ejecuta un programa (P) que realiza 5.000 millones ($5x10^9$) de accesos a memoria. Sabemos que el 60% de los accesos son a la cache de instrucciones (CI) y el 40% restante a la cache de datos (CD). La cache de instrucciones tiene una tasa de fallos del 10%, mientras que la cache de datos tiene una tasa de fallos del 35% (los fallos que no reemplazan bloque son despreciables). También se ha detectado que en la L2 se realizan 250 millones de escrituras de 32 bytes cada una.

c) **Calcula** (considerando CI+CD) el total de aciertos que se producen, fallos que reemplazan un bloque modificado (RBM) y fallos que reemplazan un bloque no modificado (RBM) .

	aciertos	**fallos que reemplazan un bloque modificado (RBM)**	**fallos que reemplazan un bloque NO modificado (~~RBM~~)**
Justificación (puede ser un pequeño cálculo)			
TOTAL			

Ambas caches tienen una implementación con acceso paralelo a etiquetas y datos. El tiempo de ciclo (Tc) del procesador viene determinado por el tiempo de acceso a las caches de primer nivel (CI y CD) que en ambos casos es de 1,2 ns. Con esta configuración el programa P se ha ejecutado en 30 segundos.

Para mejorar el tiempo de ciclo del procesador se han organizado las caches de primer nivel (CI y CD) en 2 etapas:

- **1a etapa**: Se lee la memoria de etiquetas de todas las vías, se comparan los TAGs y se selecciona la vía.
- **2a etapa**: Se lee/escribe el dato en la vía seleccionada de la memoria de datos.

Con esta modificación, el Tc es de 1 ns, pero los aciertos tardan un ciclo más (la CPU no realiza un nuevo acceso hasta que se ha completado el anterior). Dado que un acceso a L2 sigue tardando el mismo tiempo pero el tiempo de ciclo es menor, los fallos ~~RBM~~ tardan 2 ciclos más y los RBM tardan 4 ciclos más.

d) **Calcula** el tiempo de ejecución de P con la configuración en dos etapas.

Dado que los resultados no han sido buenos, se ha propuesto añadir un predictor de vía a la organización en dos etapas (el Tc sigue siendo de 1 ns). Esta configuración es muy similar al predictor de vía descrito en clase:

- **1a etapa**: Se lee la memoria de etiquetas de todas las vías, se comparan los TAGs y se selecciona la vía.
  En paralelo se lee el dato en la memoria de datos en la vía seleccionada por el predictor.
  Si es acierto de predictor se completa el acceso en esta etapa (las escrituras también se pueden hacer en un ciclo mediante el mecanismo de escrituras segmentadas).
- **2a etapa**: Si ha sido fallo de predictor, se lee/escribe el dato en la memoria de datos de la vía seleccionada por la comparación de TAGS.

Los fallos de cache siguen tardando lo mismo que en la organización en dos etapas. En ambas caches, el 75% de los aciertos de cache (hit) son aciertos del predictor de vía.

e) **Calcula** el tiempo de ejecución de P con la configuración con predictor de vía.

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

2on Control Arquitectura de Computadors **Curs 2015-2016 Q1**

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

Problema 2. (2,5 puntos)

Dado el siguiente código escrito en C:

```
struct S{
  char k;
  int V[3];
};

int subru(S p2, int v2[3]);

int examen(int v1[5], int i, int *p1) {
  int v3[10];
  S s1;
  int j;
  ...
  j = subru(s1, v3);
  ...
};
```

a) **Dibuja** el struct S indicando claramente el tamaño de cada campo, su desplazamiento respecto al inicio de la estructura y el tamaño total del struct.

b) **Dibuja** el bloque de activación de la subrutina examen, indicando claramente los desplazamientos y el tamaño de cada uno de los campos.

c) **Traduce** a ensamblador del IA32 la sentencia `j = subru(s1, v3);`

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

2on Control Arquitectura de Computadors **Curs 2015-2016 Q1**

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

Problema 3. (2,5 puntos)

Hemos evaluado el siguiente código en C:

```
int v[1000];              // Cada entero ocupa 4 bytes
for (i=0; i<1000; i++)    // La variable i está en un registro
  tmp = tmp + v[i];       // La variable tmp está en un registro
```

con los siguientes resultados:

- 1000 accesos a memoria de datos.
- 125 fallos en la cache de datos.

a) ¿Cuál es el tamaño de bloque en la cache de datos? Justifica la respuesta.

Hemos evaluado el siguiente código escrito en ensamblador x86:

```
LOOP: MOVL 0(%EBX), %EAX
      ADDL 4*1024(%EBX), %EAX
      MOVL %EAX, 8*1024(%EBX)
      ADDL $4, %EBX
      DECL %ECX       ; inicialmente ECX vale 1000
      JNZ LOOP        ; el bucle da 1000 iteraciones
```

con los siguientes resultados:

- 3000 accesos a memoria de datos
- 3000 fallos de cache de datos

b) Sabiendo que la cache es directa (tamaño mayor de 1KB) con la política de escritura *Write Throug y Write NO Allocate*, ¿cuales son los posibles tamaños de cache que producen este número de fallos? Justifica la respuesta.

c) Sabiendo que la cache es 2-asociativa (tamaño mayor de 1KB), ¿cual es la política de escritura y los posibles tamaños de cache que producen este número de fallos? Justifica la respuesta.

Dado el siguiente código escrito en C:

```
int M1[1024][1024], M2[1024][1024], a[1024];// alineados a 16KB
for (i=0; i<1024; i++)                      // i está en un registro
   tmp = tmp + a[i] * M1[i][3] * M2[3][i];  // tmp esta en un registro
```

d) Sabiendo que un entero ocupa 4 bytes y que nuestra cache de datos usa bloques de tamaño 32bytes, calculad cuántos bloques de memoria se han de mover de memoria principal a memoria cache para ejecutar el bucle anterior. Suponed que la cache es infinitamente grande.

a:	bloques
M1:	bloques
M2:	bloques
TOTAL:	bloques

e) Sabiendo que un entero ocupa 4 bytes y que el tamaño de página de nuestro sistema de memoria virtual es de 16KB, calculad cuántas páginas de memoria virtual se utilizan al ejecutar el bucle anterior.

a:	pág.
M1:	pág.
M2:	pág.
TOTAL:	páginas