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

Examen Final d'Arquitectura de Computadors **Curs 2011-2012 Q1**

- Duració de l'examen 2:50 hores
- Les notes sortiran el dia 20 de gener
- La revisió serà el dia 23 de gener

Problema 1. (3 puntos)

Dado el siguiente código escrito en C:

```
typedef struct {
   char a;
   char b;
   char c;
   int d[2];
   double e;
   s1 *f;
} s1;

int examina(s1 *uno, int dos){

   if (uno->d[1]!=dos)
      dos = examina(uno->f,dos+1);
   return dos;
}
```

a) **Dibuja** como quedaría almacenada en memoria la estructura s1, indicando claramente los desplazamientos respecto al inicio y el tamaño de todos los campos.

b) **Traduce** a ensamblador del x86 la rutina examina.

A partir de este punto supondremos que el bloque de activación de la rutina examina es el siguiente:

ebp	<-- ebp
@ret	
@uno	+8
dos	+12

Y que el compilador ha realizado una implementación de dicha rutina que no necesita acceder a ninguna variable local ni salvar ningún registro en memoria. Podemos suponer también que los structs a donde apunta el parámetro **`uno`** son globales y, por tanto, NO están en la pila.

c) **Escribe** en la siguiente tabla las líneas de código de vuestra implementación de la rutina examina que acceden a la PILA y cuantos accesos en total, tanto de lectura como de escritura, realiza cada una de ellas a la pila.

Línea de Código	Accesos a la pila

Total accesos a la pila cuando no se entra en el if:	
Total accesos a la pila desde el principio, entrando en el if y hasta que se salta a la llamada recursiva:	
Total accesos a la pila desde que se retorna de la llamada recursiva hasta el final de la rutina:	

El código anterior se ejecuta en un procesador que dispone de una cache especial de pila de nivel 1. Esta cache captura solo los accesos que se hacen a la pila, es decir, los realizados por las instrucciones que usan la pila implícitamente (como push o pop) o los accesos que usan como registro base ebp o esp. Así pues, para averiguar el comportamiento de este código en este tipo de cache tendremos en cuenta solo los accesos a memoria de la pila. La cache de pila tiene un tamaño de bloque de 32 bytes, un total de 512 bloques y es de acceso directo. Su política de escritura es COPY BACK + WRITE ALLOCATE. Supondremos que la pila está vacía cuando se realiza la primera llamada a la rutina examina y que en la ejecución de prueba la rutina se ejecuta 1024 veces antes de retornar.

d) **Calcula** para las 1024 ejecuciones de la rutina examina la cantidad de fallos y de accesos totales en la cache de pila. Podéis tener en cuenta que de todas las ejecuciones solo no se ejecutará el if en la última.

e) ¿Cual sería la cantidad de fallos si el tamaño de la cache se redujera a la mitad? Justifica la respuesta.

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

Examen Final d'Arquitectura de Computadors **Curs 2011-2012 Q1**

Problema 2. (3 puntos)

Queremos evaluar una aplicación numérica que se puede modelar con el siguiente pseudocódigo:

```
for (i=0;;i++) {
  LectDisco1(i);    // FASE Lectura
  Operar(i);        // FASE Cálculo
  EscrDisco2(i);    // FASE Escritura
}
```

Esta aplicación tiene tres fases bien diferenciadas (no se puede iniciar una fase hasta que acaba la anterior):

- En la fase de Lectura se lee un bloque de datos de 10 GB del disco1 almacenado posiciones consecutivas. Este disco tienen un ancho de banda de 250 MB/s.
- En la fase de cálculo se realizan $25 \cdot 10^9$ operaciones en coma flotante, y la velocidad de ejecución es de 5 GFLOPS.
- En la última fase se escribe un bloque de datos de 7.5 GB en el disco2. Este bloque se almacena en posiciones consecutivas de disco. Esta fase tarda 15 segundos en ejecutarse.

En todo el problema, no vamos a considerar los tiempos de búsqueda y la latencia de los discos, además el tiempo de ejecución de las instrucciones de las fases de lectura y escritura es despreciable.

a) **Calcula** el tiempo de ejecución de las Fases de Lectura y Cálculo.

b) **Calcula** el ancho de banda efectivo de la fase de ESCRITURA. Calculad cuántas iteraciones del bucle se realizan por hora.

La fase de Cálculo puede dividirse en dos partes:

- Una zona secuencial que corresponde al 20% del tiempo total de ejecución.
- Una zona completamente paralelizable que corresponde al 80% del tiempo total de ejecución.

c) **Calcula** el tiempo de ejecución de la fase de cálculo si dispusiéramos de un multiprocesador con 8 CPUs. **Calcula** cuántas iteraciones del bucle se realizan por hora

En general, cuando se paraleliza una aplicación se introduce una sobrecarga adicional por la sincronización necesaria. En este caso, por cada CPU se añade al tiempo de ejecución 0,25 s de sincronización.

d) **Calcula** el tiempo de ejecución de la fase de cálculo, teniendo en cuenta la sincronización, utilizando 8 CPUs. **Calcula** también el número óptimo de CPUs.

Es obvio que en esta aplicación el cuello de botella es el acceso a los discos. Para mejorar el rendimiento de esta aplicación vamos a sustituir:

- el disco 1 por un RAID 5 de 20 discos, con tiras de 2MB.
- el disco 2 por un RAID 6 de 32 discos con tiras de 1MB.

e) Teniendo en cuenta que los discos de los RAIDs tienen un ancho de banda de 500 MB/s, calculad el tiempo de ejecución de las fases de Lectura y Escritura (recordad que los bloques de datos se leen / escriben en posiciones consecutivas).

f) ¿Qué implicaciones tiene en este problema sustituir el RAID5 por un RAID0? justifica la respuesta

El concepto de segmentación, también puede aplicarse a nivel software. Por ejemplo, la ejecución de las primeras iteraciones de nuestra aplicación podría ordenarse de esta forma:

i=-2	i=-1	i=0	i=1	i=2	...
Lect(0)	Operar(0)	Escr(0)			
	Lect(1)	Operar(1)	Escr(1)		
		Lect(2)	Operar(2)	Escr(2)	
			Lect(3)	Operar(3)	...
				Lect(4)	...

Si las fases se ejecutan columna a columna (de la tabla), y dentro de la columna de forma secuencial, el algoritmo es perfectamente válido. Pero, lo interesante es que a partir de la columna i=0, las 3 fases que hay en esa columna son totalmente independientes entre si: Escr(0), se puede ejecutar porque ya se ha realizado Operar(0); Operar(1), se puede ejecutar porque ya se ha realizado Lect(1); y Lect(2) que es independiente. En definitiva, estas 3 fases se pueden ejecutar en paralelo.

El pseudocódigo resultante de aplicar esta optimización podría ser el siguiente:

```
LectDisco1(0);        // FASE Lectura 0
Operar(0);            // FASE Cálculo 0
LectDisco1(1);        // FASE Lectura 1
for (i=0;;i++) { // BUCLE PARALELO
    EscrDisco2(i);      // FASE Escritura
    Operar(i+1);        // FASE Cálculo
    LectDisco1(i+2);    // FASE Lectura
}
```

Las 3 sentencias que hay dentro del **BUCLE PARALELO** se ejecutan en paralelo.

g) Utilizando los datos del primer apartado, calculad cuántas iteraciones del bucle se realizan por hora.

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

Examen Final d'Arquitectura de Computadors **Curs 2011-2012 Q1**

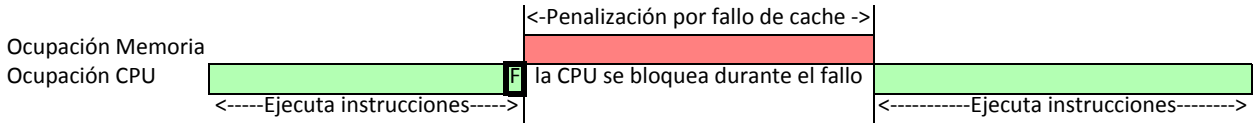
Problema 3. (4 puntos)

Se ha simulado la ejecución de un programa P en un procesador que denominaremos IDEAL. En este procesador IDEAL no hay ninguna penalización por fallo de cache. De esta simulación se ha obtenido que el programa P se ha ejecutado en $5x10^9$ ciclos durante los que ha ejecutado $2x10^9$ instrucciones, de las que $500x10^6$ son instrucciones de acceso a datos (Load/Store) y se han producido $50x10^6$ fallos en la cache de datos (los fallos en la cache de instrucciones son negligibles, con lo que los ignoraremos durante todo el problema). Suponemos que la probabilidad de fallar en cualquier ciclo es la misma y es independiente de que se haya fallado o no en el ciclo anterior.

a) **Calculad** el CPI de P en el procesador IDEAL (CPI_{IDEAL}).

b) **Calculad** el número medio de ciclos transcurridos entre 2 fallos.

El mismo programa lo ejecutamos en un procesador real con las mismas características que el IDEAL con la única diferencia que en caso de fallo en la cache de datos se bloquea la ejecución de instrucciones durante un cierto numero de ciclos que corresponden al tiempo medio de penalización por fallo de cache (Tpf) hasta que se resuelve el fallo. La siguiente figura ilustra este hecho cuando se detecta un fallo (F).

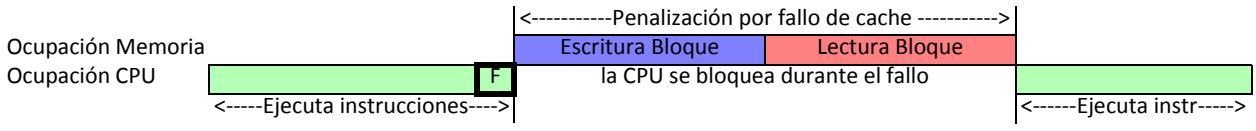


El programa P se ha ejecutado en el procesador real (que llamaremos procesador A) en 3 segundos. Este procesador A funciona a una frecuencia de 2 GHz.

c) **Calculad** el CPI de P en el procesador A (CPI_A)

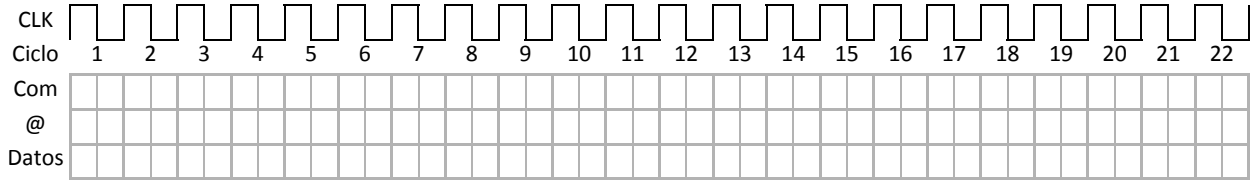
d) **Calculad** el número medio de ciclos de penalización por fallo de cache (Tpf)

La cache de datos sigue una política de escritura COPY BACK + WRITE ALLOCATE. En caso de fallo, si la línea reemplazada ha sido modificada, la penalización es de 30 ciclos, mientras que si no lo ha sido es sólo de 15 ciclos. Se sabe que durante la ejecución de P, en media 1/3 de los bloques de cache han sido modificados (dirty bit = 1). La siguiente figura muestra el cronograma de un fallo en que la línea reemplazada ha sido modificada.



La memoria principal está formada por un módulo DDR-SDRAM (Double Data Rate Synchronous Dynamic RAM) de 64 bits de ancho, que transfiere ráfagas de 64 bytes, funciona a una frecuencia de **2 GHz** y tiene las siguientes latencias: latencia de fila 4 ciclos; latencia de columna 4 ciclos; latencia de precarga 3 ciclos. En el siguiente cronograma, indicaremos en la fila **Com** el ciclo en que se lanza cada comando: ACTIVE (**ACT**) para abrir página, READ (**RD**) para lectura, WRITE (**WR**) para escritura y PRECHARGE (**PRE**) para cerrar página. En la fila **@** el ciclo en que se envía la dirección de fila (**@F**) o de columna (**@C**). Y, en la fila **Datos**, los subciclos en que se transmiten los datos (**D**).

e) **Rellena** el cronograma indicando la ocupación de los distintos recursos para la lectura de un bloque de 64 bytes (en que abrimos y cerramos página).

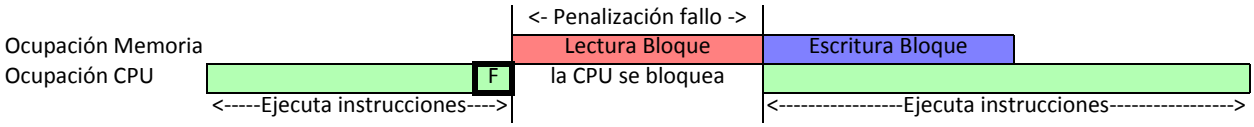


Queremos evaluar la energía consumida en la memoria debida a la actividad del programa (ignoraremos por tanto la energía debida a fugas). La tensión de alimentación de esta memoria es de 1.5 voltios, mientras que la corriente consumida depende de la actividad. Durante toda la operación de lectura (desde que se envía el comando ACTIVE hasta que se completa el PRECHARGE) se consumen 2 Amperios por el funcionamiento de los componentes internos. Durante la transferencia de datos, además de los componentes internos, hay que alimentar los drivers de entrada salida, con lo que se consumen otros 6 Amperios adicionales.

f) **Calcula** la potencia media y la energía consumidas por la DDR durante la lectura de un bloque de 64 bytes.

g) **Calcula** la energía y la potencia media consumidas por la DDR durante la ejecución del programa P (escribir un bloque en la DDR consume la misma energía que una lectura).

Una posible mejora (vista en clase de teoría) consiste en incorporar un buffer para almacenar el bloque reemplazado, de forma que podamos leer primero el bloque que ha provocado el fallo y a continuación escribir en memoria el bloque reemplazado. Esta implementación permite que el procesador pueda seguir ejecutando instrucciones y la cache pueda ser accedida durante la escritura del bloque reemplazado a memoria, tal como muestra la siguiente figura.



De momento supondremos la situación ideal (aunque no realista) en que nunca se produce un fallo de cache mientras se realiza la escritura del bloque reemplazado. A este procesador lo denominaremos procesador B

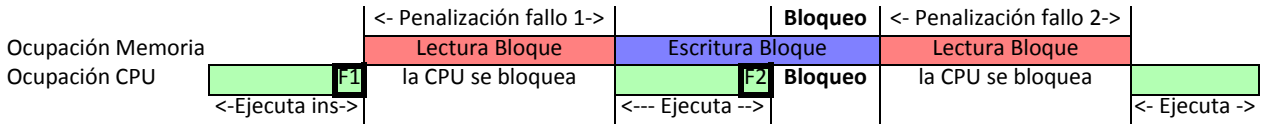
h) **Calculad** el numero de ciclos necesario para ejecutar P en el procesador B

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

Examen Final d'Arquitectura de Computadors **Curs 2011-2012 Q1**

Problema 3. (continuación)

En la implementación real (que denominamos procesador C) dispondremos de un buffer de una sola entrada que nos permite tener como máximo una escritura pendiente. Si durante la escritura del bloque causada por un fallo (F1) se produce un segundo fallo (F2), hay que esperar a que la jerarquía de memoria complete la escritura del bloque antes de que pueda empezar a servir el siguiente fallo, con lo que el procesador se bloqueará por unos ciclos adicionales (**Bloqueo**), tal como muestra la siguiente figura (en el ejemplo suponemos que el siguiente fallo reemplaza un bloque no modificado).



Durante la fase en que la CPU ejecuta instrucciones estas se ejecutan con la misma distribución que en el procesador IDEAL, por lo que el número medio de ciclos en que la CPU ejecuta instrucciones (sin contar el bloqueo debido a la lectura del bloque solicitado por el fallo F1) entre F1 y F2 será el mismo.

a) **Calculad** la probabilidad de que se produzca un segundo fallo durante la escritura de un bloque reemplazado

Hemos medido que, si se produce un segundo fallo durante el intervalo de escritura de un bloque anterior, en media la CPU se bloquea durante 7 ciclos (ciclos correspondientes al intervalo **Bloqueo** de la figura anterior) antes de que se pueda iniciar la lectura del bloque que ha causado el segundo fallo.

b) **Calculad** el numero de ciclos necesario para ejecutar P en el procesador C