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

3er Control Arquitectura de Computadors **Curs 2011-2012 Q1**

Problema 1. (3 puntos)

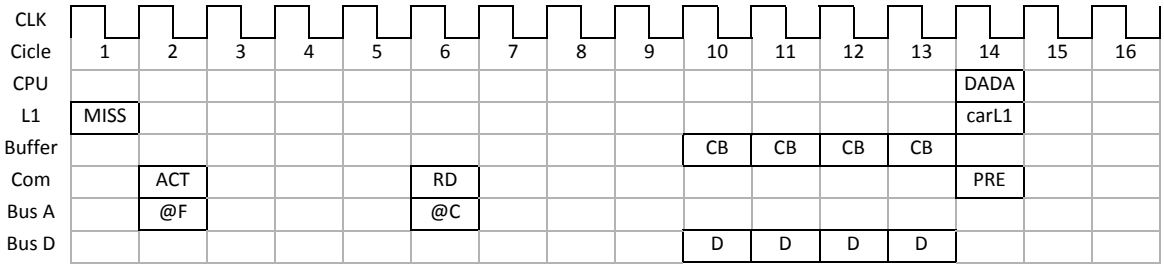
Una CPU de 32 bits tindrà un temps de cicle (Tc) de 10 ns. A l'executar un programa P (que executa $5x10^9$ instruccions) en un simulador on tots els accessos a memòria tarden 1 cicle s'ha mesurat un CPI de 1,8 cicles/ instrucció (que anomenarem CPI_{ideal}).

a) **Calculeu** el temps d'execució (T_{exec}) del programa P en aquest sistema de memòria ideal (en segons).

Per mesurar l'impacte de la cerca d'instruccions en el rendiment, hem modificat el simulador per analitzar un sistema amb una cache d'instruccions (que anomenarem L1) i una memòria principal SDRAM (els accessos a dades segueixen tardant 1 cicle). La mida de les instruccions es de 4 bytes. La mida de bloc de L1 es de 32 bytes i el temps d'accés en cas d'encert a L1 (T_h) es de 1 cicle. Pel programa P la taxa de fallades (m_1) de L1 es del 10%.

b) **Calculeu** quants accessos a L1 fa el programa P.

La memòria SDRAM esta formada per un DIMM de 8 bytes d'amplada amb una latència de fila de 4 cicles, una latència de columna de 4 cicles i un temps per la comanda PRECHARGE de 1 cicle. El següent cronograma mostra els passos en cas de fallada a L1. En el cicle 1 s'accedeix a L1 i es detecta que es una fallada de cache. En el cicle 2 s'envia la comanda ACTIVE i l'adreça de fila (Bus A) per activar la pàgina corresponent de memòria i 4 cicles després (cicle 6) s'envia la comanda READ i l'adreça de columna. Al cicle 10 (4 cicles després de RD) apareixen les dades (4 cicles 8 bytes/cicle) al bus de dades (Bus D). Les dades es van carregant a un *buffer* (cicles CB) a mesura que van apareixent pel bus (cicles etiquetats D). Finalment al cicle 14, un cop s'ha transmès tot el bloc al *buffer,* es passa la instrucció a la CPU (DADA) i en paral.lel s'escriu el bloc a L1 (carL1) i s'activa la comanda PRECHARGE per tancar la pàgina (PRE).



c) **Calculeu** el temps mig d'accés a memòria (T_{mam}) (en nanosegons).

d) **Calculeu** el temps d'execució (T_{exec}) del programa P (en segons).

A aquest sistema afegim un segon nivell de cache (L2) de forma que, si es falla a L1 s'accedeix a L2 i nomes en cas de fallar al segon nivell s'ha d'accedir a memòria principal. La taxa local de fallades (m_2) de L2 es del 30%. La mida de bloc de L2 es també de 32 bytes.

e) **Calculeu** el percentatge d'accessos que fallen a L1 i encerten a L2 i el percentatge d'accessos que fallen a L1 i a L2.

El següent cronograma mostra els passos en cas de fallada a L1 i encert a L2. Al cicle 1 s'accedeix a L1 i es detecta que es una fallada a L1. Un accés a L2 tarda 4 cicles (del 2 al 5). En el cicle 2 (TAG) es llegeix la memòria d'etiquetes, en el cicle 3 (CMP) es comparen les etiquetes i es comprova que es *hit* a L2, en el cicles 4 i 5 es llegeix la memòria de dades de la L2 (RD1 i RD2). Finalment al cicle 6 s'escriu el bloc a L1 (carL1) i, en paral.lel, es passa la instrucció a la CPU (DADA).

CLK																
Cicle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU						DADA										
L1	MISS					carL1										
L2		TAG	CMP	RD1	RD2											

El següent cronograma mostra els passos en cas de fallada a L1 i a L2. Al cicle 1 s'accedeix a L1 i es detecta que es una fallada a L1. En el cicle 2 (TAG) es llegeix la memòria d'etiquetes, en el cicle 3 es comparen les etiquetes i es comprova que es *miss* a L2. Dels cicles 4 al 15 es llegeix el bloc de la SDRAM tal com ja s'ha explicat per la configuració amb un sol nivell de cache. Un cop tenim el bloc al *buffer*, aquest s'escriu simultaniament a L1 (carL1), L2 (2 cicles WR1 i WR2) i es passa la instrucció a la CPU (DADA).

CLK																	
Cicle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
CPU																DADA	
L1	MISS															carL1	
L2		TAG	CMP													WR1	WR2
Buffer												CB	CB	CB	CB		
Com				ACT				RD								PRE	
@				@F				@C									
Datos												D	D	D	D		

f) **Calculeu** el temps mig d'accés a memòria (T_{mam}) (en nanosegons).

g) **Calculeu** el temps d'execució (T_{exec}) del programa P (en segons).

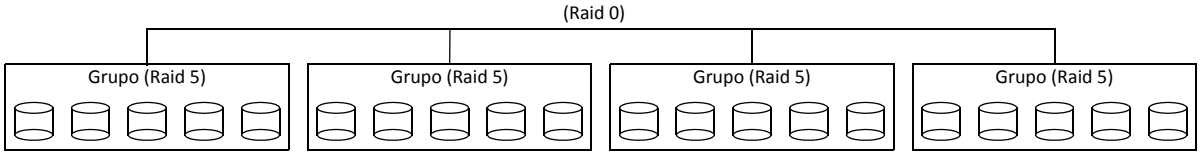
h) **Calculeu** el guany (*speed-up*) del sistema amb L1 i L2 respecte el sistema que només te L1.

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

3er Control Arquitectura de Computadors **Curs 2011-2012 Q1**

Problema 2. (3 puntos)

Disponemos de 20 discos físicos de 1 Tbyte de capacidad por disco, que ofrecen un ancho de banda efectivo de 250 Mbytes/s por disco. Con estos discos deseamos montar un sistema de almacenamiento basado en RAID 50 organizado en 4 grupos de 5 discos.



a) **Calcular** la cantidad de información útil (datos) que puede almacenar el sistema RAID50

Para analizar el ancho de banda en escritura consideraremos por separado el caso en que realizamos accesos a tiras consecutivas y el caso en que realizamos accesos a tiras aleatorias. En el caso de accesos aleatorios, el controlador del RAID ordena las peticiones de acceso para aprovechar al máximo la concurrencia entre discos. En ambos casos consideraremos que disponemos de suficientes peticiones de escritura de forma que el controlador del RAID siempre puede aprovechar el ancho de banda de todos los discos físicos.

En el caso de escrituras secuenciales es posible esperar a tener una cantidad de datos suficiente para calcular la paridad correspondiente y escribir simultáneamente en todos los discos.

b) **Calcular** el ancho de banda efectivo (datos), si hacemos **escrituras** secuenciales, para el sistema RAID50.

En el caso de escrituras aleatorias es necesario leer los datos antiguos y la tira de paridad correspondiente para calcular la nueva paridad y además escribir tanto datos como paridad por cada escritura que se desea realizar. Obsérvese que para escribir 1 tira de datos es necesario realizar 4 operaciones de disco.

c) **Calcular** el ancho de banda efectivo (datos), si hacemos **escrituras** aleatorias, para el sistema RAID50.

Cada disco tiene un tiempo medio entre fallos ($MTTF_{disco}$) de 100.000 horas y el tiempo de recuperar la información en caso de tener que reemplazar un disco ($MTTR_{disco}$) es de 10 horas.

d) **Calcular** el tiempo medio entre fallos de un grupo de 5 discos ($MTTF_{GRUPO}$) suponiendo que los discos son el único componente que puede fallar.

e) **Calcular** el tiempo medio entre fallos del sistema RAID50 ($MTTF_{RAID50}$) suponiendo que los discos son el único componente que puede fallar.

En nuestra implementación, cada grupo de discos tiene su propia fuente de alimentación con un tiempo medio entre fallos ($MTTF_{fuente}$) de 1.000.000 horas

f) **Calcular** el tiempo medio entre fallos de un grupo de 5 discos teniendo en cuenta también los fallos de las fuentes ($MTTF_{GF}$).

g) **Calcular** el tiempo medio entre fallos del sistema RAID50 ($MTTF_{RAID50}$) teniendo en cuenta tanto los posibles fallos de disco cómo los de las fuentes

h) ¿Cuál crees que, en lo que a fiabilidad se refiere, es el componente crítico del sistema? Propon alguna solución para mejorar la fiabilidad del sistema (sin variar la capacidad de almacenamiento de datos del sistema).

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

3er Control Arquitectura de Computadors **Curs 2011-2012 Q1**

Problema 3. (2 puntos)

Debemos decidir el diseño de un procesador de señal (DSP) que ejecutará únicamente el siguiente kernel (observese que es un bucle infinito). Todas las variables son de coma flotante de simple precisión. **Las variables cambian de valor en cada iteración** ya que estan mapeadas a diversos sensores que las actualizan constantemente. Por ello **deben cargarse de y guardarse en memoria cada iteración**.

```
while (true) {
    A=(A-B*C)/(D+E)
}
```

Las dos alternativas de las que disponemos son un procesador Acumulador y otro tipo Memoria/Memoria. La descripción del ISA de ambos procesadores es la siguiente:

	Acumulador		**Memoria / Memoria**	
Tipo ins.	Ejemplo	Operación	Ejemplo	Operación
Aritmética	Add A	ACC=ACC+A	Add A, B, C	C=A+B
Memoria	Load A	ACC=A	--	--
Salto	Br Etiq	PC=PC+despl	Br Etiq	PC=PC+despl

a) Traduce el código anterior al ensamblador del procesador Memoria/Memoria. Podéis usar los registros R1a R8 si lo necesitáis. Observad que NO podéis optimizar el código haciendo loads o stores fuera del bucle dado que los datos son distintos en cada iteración. Escribid cuantas instrucciones dinámicas se ejecutan en cada iteración del bucle y cuantos accesos a memoria de datos se realizan también para cada iteración del bucle.

b) Traduce el código anterior al ensamblador del procesador Acumulador. Podéis usar las variables temporales temp1 a temp8 si lo necesitáis. Recordad que NO podéis optimizar el código haciendo loads o stores fuera del

bucle. Escribid cuantas instrucciones dinámicas se ejecutan en cada iteración del bucle y cuantos accesos a memoria de datos se realizan también para cada iteración del bucle.

Cualquiera que sea el procesador elegido se va a conectar al mismo subsistema de memoria. En el caso de la memoria de datos, para el kernel anterior, esta es capaz de dar un ancho de banda sostenido de 3 GB/s. En cambio la memoria de instrucciones es capaz de ofrecer, también para el kernel anterior, un ancho de banda sostenido de 4 GB/s. Cada instrucción del procesador Acumulador ocupa 4 bytes y cada instrucción del procesador Memoria/Memoria ocupa 16 bytes. Sabemos que la memoria es el cuello de botella del sistema y por tanto el rendimiento del procesador estará únicamente limitado por el ancho de banda con memoria.

c) Explica de forma razonada y justifica cuantitativamente cuál es el procesador capaz de ejecutar el código más rápidamente y calcula cuál es su ganancia con respecto al más lento para el código dado (Pista: calculad cuantas iteraciones por segundo puede hacer cada procesador)**.**

Una vez elegido el tipo de ISA del procesador debemos calcular su rendimiento. A partir de aquí y para cumplir los requisitos de tiempo real del programa suponemos que el procesador ejecuta el código a 5E7 iteraciones del bucle por segundo. Sabemos que para conseguirlo el procesador debería funcionar a 2 GHz y con un voltaje de 2 V. En estas condiciones su carga capacitiva equivalente sería de 5nF y su corriente de fugas de 2A.

d) Calcula el rendimiento del procesador elegido en MFLOPS/W.

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

3er Control Arquitectura de Computadors **Curs 2011-2012 Q1**

Problema 4. (2 puntos)

Dado el siguiente código escrito en C:

```
typedef struct {
   int a;
   char b;
   int c;
   int d;
} s1;
s1 X;
```

a) **Dibujad** cómo quedaría almacenada en memoria la estructuras s1, indicando claramente los desplazamientos respecto al inicio y el tamaño de todos los campos.
Traducid a ensamblador del x86 la siguiente sentencia: "`X.a = X.a + X.d + (int) X.b`", suponed que el registro ebx tiene la dirección de inicio de X.

Disponemos del siguiente código escrito en C:

```
s1 V[1024*256];
for (i=0; i<1024*256; i++)
  V[i].a = V[i].a + V[i].d + (int)V[i].b;
```

Este código se ejecuta en un procesador con una memoria cache de datos 2-asociativa de 8KB, con bloques de tamaño 32B y política de escritura write through+write no allocate.

b) Suponiendo que la dirección de inicio del vector es 0 y que sólo consideramos los accesos al vector V. Calculad: número total de accesos a memoria, número de aciertos/fallos a la cache de datos, la tasa de fallos, el total de bytes leidos de Memoria Principal y el total de bytes escritos en Memoria Principal.
Para hacer estos cálculos, os ayudará rellener la siguiente tabla:

iteración	0			1			2			3			4		
acceso	@	Con.j MC	Fallo MC	@	Conj. MC	Fallo MC	@	Conj. MC	Fallo MC	@	Conj. MC	Fallo MC	@	Conj. MC	Fallo MC
V[i].b (LECT)															
V[i].d (LECT)															
V[i].a (LECT)															
V[i].a (ESCR)															

#Accesos a Memoria:	Tasa de fallos:
#Fallos:	Total bytes leidos de MP:
#Aciertos:	Total bytes escritos en MP: