3er Control Arquitectura de Computadors — Curs 2010-2011 Q2

Problema 1. (1 punto).

Dado el siguiente código escrito en C:

```
typedef struct {
  short a;
  char b;
  char c;
  char d;
  short e;
} s1;

char *examen(s1 paruno, s2 *pardos){
char vl1;
   ...
}
```

```
typedef struct {
  char e[5];
  s1 f;
} s2;
```

a) **Dibuja** como quedarían almacenadas en memoria las estructuras s1 y s2, indicando claramente los desplazamientos respecto al inicio y el tamaño de todos los campos.

b) **Dibuja** el bloque de activación de la función examen, indicando claramente los desplazamientos relativos al registro EBP necesarios para acceder a los parámetros y a las variables locales.

Problema 2. (3 puntos)

Disponemos de una CPU que funciona a 3 GHz. En esta CPU hemos simulado la ejecución de un programa con un sistema de memoria en donde todos los accesos a memoria tardan 1 ciclo sean aciertos o fallos (denominaremos **CPU_{IDEAL}** a esta combinación simulada) y hemos obtenido que el programa se ejecuta en $15x10^9$ ciclos, ejecuta $5x10^9$ instrucciones, realiza $6x10^9$ accesos a memoria y de estos, $500x10^6$ son fallos de cache.

a) **Calcula** el numero medio de ciclos entre accesos, la probabilidad de acceder a memoria en un ciclo determinado, el CPI, el tiempo de ejecución del programa en la **CPU_{IDEAL}**.

Queremos integrar esta CPU con una cache unificada (datos + instrucciones) de mapeo directo. Esta cache no está segmentada y su tiempo de acceso es de 0,6 ns. Obsérvese que el tiempo de acceso es mayor que el tiempo de ciclo del procesador. En estas condiciones, todos los accesos a memoria se ven penalizados en 1 ciclo (aciertos y fallos). En caso de que el acceso sea un fallo de cache, hay una penalización adicional de 30 ciclos más.

b) **Calcula** los ciclos totales, el CPI, y el tiempo de ejecución cuando ejecutamos el programa con la cache directa.

Esta CPU tiene 2 modos de funcionamiento: modo de alto rendimiento, la CPU funciona a una frecuencia de 3 GHz con un consumo de 70 W; y el modo de bajo consumo en el que la CPU funciona a 1 GHz. En ambos modos la potencia de fugas es 10W. Suponiendo que C y V tienen el mismo valor en ambos modos de funcionamiento.

c) **Calcula** la potencia de conmutación en el modo de alto rendimiento. **Calcula** la potencia total en el modo de bajo consumo.

d) **Calcula** la energía consumida por el procesador con cache unificada ejecutando el programa de prueba, suponiendo que la CPU está en el modo de alto rendimiento.

Con el objetivo de reducir el consumo, se ha modificado el control del procesador para que cuando se produce un fallo de cache, durante los 30 ciclos de penalización (a 3 GHz) el procesador pasa al modo de bajo consumo. El tiempo de penalización por fallo en cache depende del nivel superior de la jerarquía y en este caso es constante e independiente de la frecuencia de reloj de la CPU.

e) **Calcula** el tiempo que la CPU estará en modo bajo consumo (tiempo perdido por fallos de cache).

f) **Calcula** la energía consumida por el procesador ejecutando el programa de prueba, suponiendo que la CPU pasa al modo de bajo consumo mientras se resuelve un fallo de cache. Calcula la potencia media en este último caso.

A partir de aquí supondremos que la CPU funciona siempre a 3 GHz.

Nuestra CPU es capaz de continuar ejecutando instrucciones mientras se accede la cache. Sin embargo en el apartado anterior, bloqueamos la CPU en cada acceso a cache, para evitar lanzar un segundo acceso a la cache antes de que acabe el anterior.

Una posible mejora, que denominaremos control de bloqueos de cache, consiste en no bloquear la CPU en cada acceso, sino solamente si se inicia un acceso antes de que el anterior haya terminado. La CPU no soporta loads no bloqueantes, por lo que en caso de fallo siempre bloquearemos la CPU mientras se trae el bloque del siguiente nivel de la jerarquía (ciclos de penalización adicional). Sabemos que la probabilidad de realizar un acceso es la misma en todos los ciclos y es independiente de lo sucedido en ciclos anteriores. Durante los ciclos que no está bloqueada, la CPU se comporta exactamente igual que en el caso ideal.

g) **Calcula** la probabilidad de que al realizar un acceso a memoria la cache esté ocupada.

h) **Calcula** los ciclos totales, el CPI, y el tiempo de ejecución cuando ejecutamos el programa en la CPU con control de bloqueos de cache.

Otra mejora que podemos añadir, es organizar la cache en 4 bancos. En una cache organizada en bancos el acceso a cada banco es independiente, por lo que es posible acceder a un banco aunque otro esté ocupado. La mejora consiste en bloquear la CPU solamente en caso de que accedamos a un banco ocupado. En nuestro caso, sabemos que en cada acceso la probabilidad de acceder a cualquiera de los 4 bancos es la misma, y que es independiente de los accesos anteriores. Como en el caso anterior, la CPU no soporta loads no bloqueantes, por lo que en caso de fallo siempre bloquearemos la CPU mientras se trae el bloque del siguiente nivel de la jerarquía (ciclos de penalización adicional).

i) **Calcula** los ciclos totales, el CPI, y el tiempo de ejecución cuando ejecutamos el programa en la CPU con la cache multibanco.

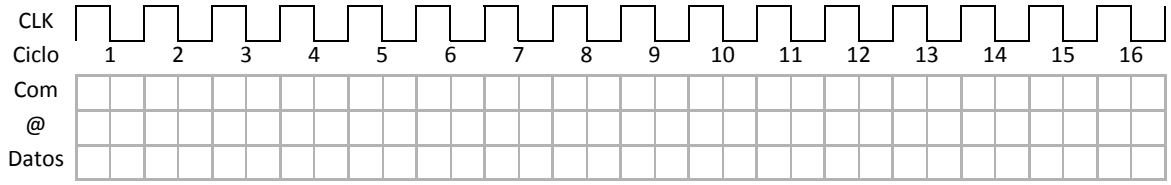
Problema 3. (3 puntos)

Disponemos de un DIMM de memoria **DDR** (Double Data Rate) con 8 chips de 1 byte cada uno por DIMM, la latencia de fila es de 4 ciclos, la de columna es de 3 ciclos y la de precarga es de 2 ciclos. El bus de esta memoria funciona a una frecuencia de reloj de 800 MHz. Los accesos se hacen en ráfagas de 4 paquetes de 8 bytes, es decir se transmiten 32 bytes de datos en cada acceso.

Para indicar la ocupación de los distintos recursos utilizaremos la siguiente nomenclatura:

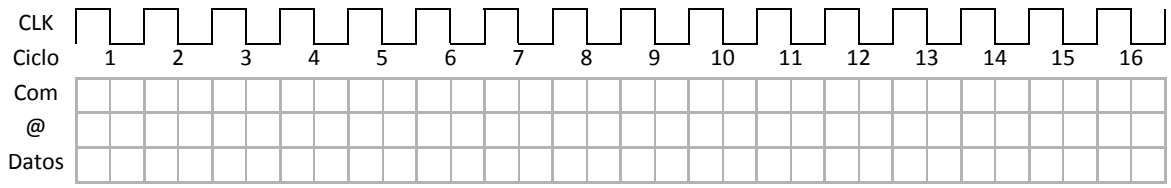
- **ACT**: comando ACTIVE (abrir página)
- **RD**: comando READ
- **WR**: comando WRITE
- **PRE**: comando PRECHARGE (cerrar página)
- **@F**: ciclo en que se envía la dirección de fila
- **@C**: ciclo en que se envía la dirección de columna
- **D**: transmisión de un paquete de 8 bytes

a) **Rellena** el siguiente cronograma indicando la ocupación de los distintos recursos para una lectura de 32 bytes.



Este DIMM lo conectamos a una CPU integrada con un controlador de memoria y un primer nivel de cache con caches de instrucciones y datos separadas, ambas con bloques de 64 bytes. Cuando el controlador de memoria de este procesador desea leer un bloque de memoria de 64 bytes tiene que abrir página, realizar dos lecturas de 32 bytes y cerrar página.

b) **Rellena** el siguiente cronograma indicando la ocupación de los distintos recursos para leer un bloque de memoria de 64 bytes de forma que se haga lo más rápidamente posible.



Esta CPU es un procesador RISC segmentado. En esta CPU ejecutamos el siguiente código, en donde **v** y **c** son variables de punto flotante en doble precisión (8bytes).

```
for (i=0; i<N; i++)
   v[i] = v[i] * c;
```

El compilador ha alineado el vector **v** a un múltiplo del tamaño de bloque y ha traducido el cuerpo del bucle al siguiente código máquina RISC:

```
// valor inicial de los registros: rf1 = variable c; r1  = @ inicio de v; r2  = N
loop: loadf  rf2 <- M[R1]
      mulf   rf3 <- rf2 * rf1
      storef M[r1] <- rf3
      add    r1 <- r1 + 8
      sub    r2 <- r2 - 1
      jnz    r2, loop
```

Se ha calculado que, en ausencia de fallos de cache, el bucle se ejecuta con un CPI de 1,5 ciclos/instrucción. Dado que se trata de un código muy pequeño, que cabe perfectamente en la cache de instrucciones, ignoraremos el efecto de los fallos de instrucciones y nos centraremos en los accesos a datos. La cache de datos es de mapeo directo, tiene una capacidad de 64Kbytes, un tamaño de bloque de 64 bytes y sigue una política de escritura copy back.

c) **Explica** brevemente porque en este fragmento de código no importa si la cache es write allocate o write NO allocate.

Ejecutamos el bucle anterior con N = 8192 (8*1024)

d) **Calcula** cuantos fallos de cache (de datos) se producen, cuantos bloques leemos de memoria principal y cuantos bloques escribimos.

La CPU y las caches funcionan a una frecuencia de 1,6 GHz (el doble del bus) por lo que 1 ciclo de bus se corresponde a 2 ciclos de CPU. Teniendo en cuenta el tiempo de acceso a la DDR más los tiempos de transmisión de los datos y el tiempo de arbitraje, la penalización en caso de fallo es de 25 ciclos de CPU si el bloque a reemplazar tiene el *dirty bit* == 0, mientras que si el bloque a reemplazar tiene el *dirty bit* == 1 la penalización en caso de fallo es de 50 ciclos.

e) **Calcula** el CPI del bucle con N = 8192

En caso que ejecutásemos el bucle con N = 16384 (16*1024)

f) **Calcula** cuantos fallos de cache (de datos) se producen, cuantos bloques leemos de memoria principal y cuantos bloques escribimos.

g) **Calcula** el CPI de la ejecución con N = 16384

Una posible mejora consiste en añadir un mecanismo de *prefetch hardware* de forma que cuando un bloque de cache se accede por primera vez se genera un prefecth del siguiente bloque de memoria. Este bloque que se prebusca puede que reemplace a otro bloque de la cache, por lo que nuevamente se pueden dar los casos de *dirty bit* == 0 y *dirty bit* == 1. El tiempo que tarda el prefetch en ambos casos es el mismo que las respectivas penalizaciones por fallo (25 y 50 ciclos respectivamente).

h) ¿Cuantas instrucciones se ejecutan desde que el procesador accede por 1a vez el 1er dato de un bloque de cache hasta que accede el 1er dato del siguiente bloque de cache? ¿Cuantos ciclos tarda en ejecutarlas sabiendo que en esas instrucciones no tiene fallos en cache? ¿Es suficiente para esconder la latencia de un prefecth?

i) **Calcula** el CPI del bucle con N = 16384 con el mecanismo de prefecth descrito anteriormente.

Problema 4. (2 puntos)

Una base de datos ocupa 3,6 Tbytes de información. Como medida de seguridad se realizan regularmente backups de la base de datos. Para ello se dispone de un sistema de backup basado en cinta con un ancho de banda efectivo de 50 Mbytes/s. En caso que se produzca una perdida de los datos podremos recuperar el último backup. Dado que el resto del sistema soporta anchos de banda mucho mayores, la cinta de backup es el cuello de botella que limita el tiempo de recuperar el backup.

a) **Calcula** el tiempo (en horas) que se tarda en recuperar un backup de la base de datos completa.

El sistema de disco para almacenar la base de datos esta formado por 20 discos de 500 Gbytes de capacidad. Cada disco ofrece un ancho de banda efectivo de 100 Mbytes/s y tiene un tiempo medio entre fallos ($MTTF_{disco}$) de 100.000 horas. Por cuestiones de rendimiento, el sistema de disco se ha organizado como un RAID 0. Suponemos que disponemos de suficientes peticiones de acceso para saturar el ancho de banda de los discos.

b) **Calcula** el ancho de banda efectivo del sistema de disco.

c) **Calcula** el MTTF del sistema de disco.

En caso de que falle un disco en un RAID 0 hay que reemplazar el disco que ha fallado y recuperar el backup completo de la base de datos, hasta que no se haya recuperado toda la base de datos el sistema no estará disponible. Nuestro sistema de disco dispone de un disco adicional de forma que la sustitución del disco fallido por el nuevo sea inmediata y se pueda empezar a recuperar el backup tan pronto se detecta el fallo.

d) **Calcula** la disponibilidad del sistema de disco con la organización RAID 0

Dado que esta disponibilidad se considera insuficiente, se ha decidido evaluar la posibilidad de organizar los discos mediante un RAID 5. En nuestra base de datos se producen accesos de forma aleatoria, aunque se dispone de suficientes accesos para saturar el ancho de banda de todos los discos.

e) **Calcula** el ancho de banda efectivo del sistema si todos los accesos son lecturas.
Calcula el ancho de banda efectivo del sistema si todos los accesos son escrituras.

Sabemos que en nuestra base de datos se producen un 20% de escrituras.

f) **Calcula** el % de ancho de banda que se perderia respecto al RAID 0

En el sistema RAID 5 en caso de que falle un disco se puede recuperar ese disco ($MTTR_{disco}$) en 2 horas. En caso que falle el sistema de disco nuevamente hay que recuperar la base de datos completa del backup en cinta.

g) **Calcula** el MTTF del sistema.

h) **Calcula** la disponibilidad del sistema.

Problema 5. Problema de instrucciones (1 punto)

Los actuales procesadores CISC de la familia x86 traducen internamente las instrucciones de lenguaje máquina x86 a microoperaciones (que denominaremos uops). A continuación se presenta un código ensamblador y su correspondiente traducción a uops::

```
  movl $0, %ecx
loop:
  cmpl $1000000, %ecx
  jge fin
  movl x, %eax
  imull V(,ecx,4), %eax
  addl %eax, suma
  incl %ecx
  jmp loop
fin:
```

```
  movl %ecx <- 0
loop:
  cmpl %ecx - $1000000
  jge fin
  load %eax <- M[x]
  load %r1 <- M[%ecx*4+V]
  imull %eax <- %r1*%eax
  load %r1 <- M[suma]
  addl %r1 <- %r1 + %eax
  store M[suma] <- %r1
  addl %ecx <- %ecx + $1
  jmp loop
fin:
```

Sabemos que cada uop se codifica con 6 bytes. Además este procesador, que funciona a 2 GHz, incorpora una cache de uops (de nivel 0) que permite evitar la traducción reiterada del mismo código. En el código anterior se ha medido el ancho de banda de la cache de uops y se ha visto que este es de 30 Gbytes/s.

a) **Calcula** cual es el UPC (uops por ciclo) al que se ejecuta este código.

b) **Calcula** cual es el CPI (Ciclos Por Instrucción) al que se ejecuta el código x86 anterior.

c) **Calcula** el tiempo de ejecución del programa.

d) Con los datos de que dispones, ¿es posible calcular el ancho de banda efectivo de la cache de datos del procesador? Si la respuesta es sí, **calcula** el ancho de banda.

e) Con los datos de que dispones, ¿es posible calcular el ancho de banda efectivo de la cache de instrucciones (no la de uops) del procesador? Si la respuesta es sí, **calcula** el ancho de banda.

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

3er Control Arquitectura de Computadors **Curs 2010-2011 Q2**

Problema 1. (1 punto).

a) **Dibuja** como quedarían almacenadas en memoria las estructuras s1 y s2, indicando claramente los desplazamientos respecto al inicio y el tamaño de todos los campos.

b) **Dibuja** el bloque de activación de la función examen, indicando claramente los desplazamientos relativos al registro EBP necesarios para acceder a los parámetros y a las variables locales.

Problema 2. (3 puntos)

a) **Calcula** el numero medio de ciclos entre accesos, la probabilidad de acceder a memoria en un ciclo determinado, el CPI, el tiempo de ejecución del programa en la $\mathbf{CPU_{IDEAL}}$.

b) **Calcula** los ciclos totales, el CPI, y el tiempo de ejecución cuando ejecutamos el programa con la cache directa.

c) **Calcula** la potencia de conmutación en el modo de alto rendimiento. **Calcula** la potencia total en el modo de bajo consumo.

d) **Calcula** la energía consumida por el procesador con cache unificada ejecutando el programa de prueba, suponiendo que la CPU está en el modo de alto rendimiento.

e) **Calcula** el tiempo que la CPU estará en modo bajo consumo (tiempo perdido por fallos de cache).

9

f) **Calcula** la energía consumida por el procesador ejecutando el programa de prueba, suponiendo que la CPU pasa al modo de bajo consumo mientras se resuelve un fallo de cache. Calcula la potencia media en este último caso.

9

g) **Calcula** la probabilidad de que al realizar un acceso a memoria la cache esté ocupada.

h) **Calcula** los ciclos totales, el CPI, y el tiempo de ejecución cuando ejecutamos el programa en la CPU con control de bloqueos de cache.

i) **Calcula** los ciclos totales, el CPI, y el tiempo de ejecución cuando ejecutamos el programa en la CPU con la cache multibanco.

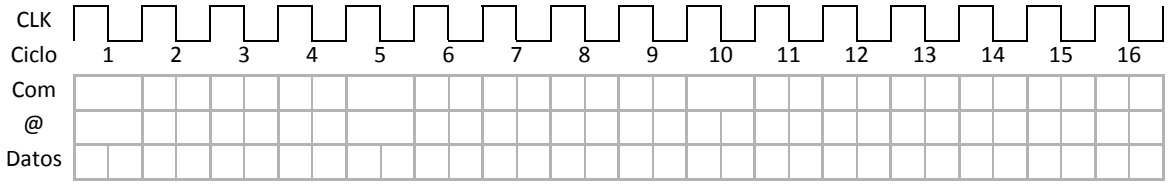
9

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

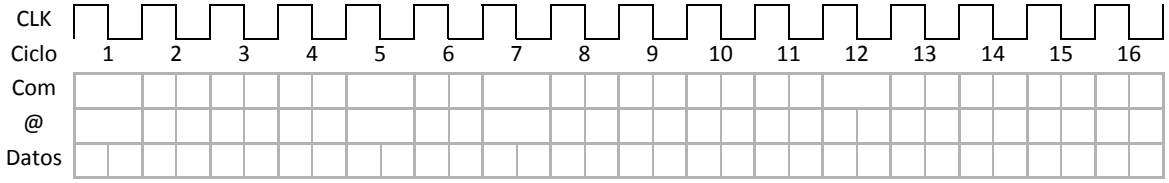
3er Control Arquitectura de Computadors **Curs 2010-2011 Q2**

Problema 3. (3 puntos)

a) **Rellena** el siguiente cronograma indicando la ocupación de los distintos recursos para una lectura de 32 bytes.



b) **Rellena** el siguiente cronograma indicando la ocupación de los distintos recursos para leer un bloque de memoria de 64 bytes de forma que se haga lo más rápidamente posible.



c) **Explica** brevemente porque en este fragmento de código no importa si la cache es write allocate o write NO allocate.

d) **Calcula** cuantos fallos de cache (de datos) se producen, cuantos bloques leemos de memoria principal y cuantos bloques escribimos.

e) **Calcula** el CPI del bucle con N = 8192

f) **Calcula** cuantos fallos de cache (de datos) se producen, cuantos bloques leemos de memoria principal y cuantos bloques escribimos.

g) **Calcula** el CPI de la ejecución con N = 16384

h) ¿Cuantas instrucciones se ejecutan desde que el procesador accede por 1a vez el 1er dato de un bloque de cache hasta que accede el 1er dato del siguiente bloque de cache? ¿Cuantos ciclos tarda en ejecutarlas sabiendo que en esas instrucciones no tiene fallos en cache? ¿Es suficiente para esconder la latencia de un prefecth?

i) **Calcula** el CPI del bucle con N = 16384 con el mecanismo de prefecth descrito anteriormente.

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

3er Control Arquitectura de Computadors **Curs 2010-2011 Q2**

Problema 4. (2 puntos)

a) **Calcula** el tiempo (en horas) que se tarda en recuperar un backup de la base de datos completa.

b) **Calcula** el ancho de banda efectivo del sistema de disco.

c) **Calcula** el MTTF del sistema de disco.

d) **Calcula** la disponibilidad del sistema de disco con la organización RAID 0

e) **Calcula** el ancho de banda efectivo del sistema si todos los accesos son lecturas.
Calcula el ancho de banda efectivo del sistema si todos los accesos son escrituras.

f) **Calcula** el % de ancho de banda que se perderia respecto al RAID 0

g) **Calcula** el MTTF del sistema.

h) **Calcula** la disponibilidad del sistema.

Problema 5. (1 punto)

a) **Calcula** cual es el UPC (uops por ciclo) al que se ejecuta este código.

b) **Calcula** cual es el CPI (Ciclos Por Instrucción) al que se ejecuta el código x86 anterior.

c) **Calcula** el tiempo de ejecución del programa

d) Con los datos de que dispones, ¿es posible calcular el ancho de banda efectivo de la cache de datos del procesador? Si la respuesta es sí, **calcula** el ancho de banda.

e) Con los datos de que dispones, ¿es posible calcular el ancho de banda efectivo de la cache de instrucciones (no la de uops) del procesador? Si la respuesta es sí, **calcula** el ancho de banda.