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

3er Control Arquitectura de Computadors **Curs 2013-2014 Q2**

- **Temps: 11:00 a 14:00**
- **Poseu clarament amb LLETRES MAJÚSCULES a cada full els cognoms i el nom**

Problema 1. (3 puntos)

Queremos ejecutar el siguiente código en una nueva CPU:

```
for(i=0; i<M; i++)
  C[i] = A[i] + 3,5*B[i];
```

Los vectores A, B y C son de números en coma flotante de doble precisión (8 bytes). Las direcciones de inicio de los vectores A, B y C son 0xA0000000, 0xB0000000 y 0xC0000000 respectivamente.

Analizando el código hemos visto que en 1 iteración del bucle se ejecutan 8 instrucciones, entre las que tenemos: 2 operaciones en coma flotante, 2 lecturas de memoria y 1 escritura en memoria. Sabiendo, que M es un valor MUY GRANDE y que el procesador funciona a 2.5 GHz.

a) **Calcula** el CPI necesario para que este código se ejecute con un rendimiento de 5 GFLOPS.

b) **Calcula** el ancho de banda necesario entre la CPU y la cache de datos para alcanzar los 5 GFLOPS.

Queremos evaluar diferentes tipos de cache de datos con este procesador. Todas las caches tendrán 4 KB y líneas de cache de 64B. (Nota: Los vectores A, B y C son varios centenares de miles de veces más grandes que la cache, cuando sea necesario, dad vuestros resultados en función de M, sin tener en cuenta los restos de las últimas iteraciones)

c) Dada una **cache directa** con política de escritura **copy back con write allocate**.

Calcula en que línea de cache se mapea A[0], B[0], C[0], A[8], B[8] y C[8].

Calcula cuántos fallos de cache se producen al ejecutar el bucle anterior (en función de M).

Calcula cuántos bytes se leen y escriben de Memoria Principal al ejecutar el bucle anterior (en función de M).

Bytes leídos de MP:

Bytes escritos en MP:

d) Dada una **cache 2-asociativa** con política de escritura **copy back con write allocate** y algoritmo de reemplazo LRU.

Calcula en que conjunto de cache se mapea A[0], B[0], C[0], A[8], B[8] y C[8].

Calcula cuántos fallos de cache se producen al ejecutar el bucle anterior (en función de M).

Calcula cuántos bytes se leen y escriben de Memoria Principal al ejecutar el bucle anterior (en función de M).

Bytes leídos de MP:

Bytes escritos en MP:

e) Dada una **cache 2-asociativa** con política de escritura **write through con write NO allocate** y reemplazo LRU.

Calcula cuántos fallos de cache se producen al ejecutar el bucle anterior (en función de M).

Calcula cuántos bytes se leen / escriben de Memoria Principal al ejecutar el bucle anterior (en función de M).

Bytes leídos de MP:

Bytes escritos en MP:

f) Dada una **cache 4-asociativa** con política de escritura **copy back con write allocate** y reemplazo LRU.

Calcula en que conjunto de cache se mapea A[0], B[0], C[0], A[8], B[8] y C[8].

Calcula cuántos fallos de cache se producen al ejecutar el bucle anterior (en función de M).

Calcula cuántos bytes se leen y escriben de Memoria Principal al ejecutar el bucle anterior (en función de M):

Bytes leídos de MP:

Bytes escritos en MP:

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

3er Control Arquitectura de Computadors **Curs 2013-2014 Q2**

Problema 2. (4 puntos)

Queremos estudiar el efecto en el rendimiento de la búsqueda y descodificación de instrucciones en un procesador segmentado superescalar con ejecución fuera de orden de la familia x86. Para ello simulamos la ejecución de un programa de prueba P en un simulador parametrizable. Para simplificar el problema ignoraremos las interferencias que nos puedan causar los accesos a datos y nos centraremos exclusivamente en los accesos a instrucciones, por lo que suponemos que los accesos a datos siempre aciertan en la cache de datos.

En los procesadores segmentados pueden transcurrir decenas de ciclos hasta que se calcula la dirección destino del salto y se conoce si el salto será tomado (*taken*) o no (*not taken*). La alternativa más simple consiste en buscar siempre instrucciones de forma secuencial, con lo que los saltos no tomados no incurren en ninguna penalización, sin embargo los saltos tomados incurren en una penalización que en nuestro procesador es de 10 ciclos.

Con el simulador hemos obtenido que el 20% de las instrucciones dinámicas de P son saltos y que el 75% de los saltos son tomados (*taken*). También hemos obtenido que, en un procesador ideal donde los saltos no penalizan, el CPI es de 0,5 ciclos/instrucción.

a) **Calcula** el CPI del procesador con búsqueda secuencial de instrucciones.

Ante tal perdida de rendimiento se ha decidido incorporar un predictor de saltos al procesador con el que hemos obtenido un CPI de 0,9 ciclos/instrucción. En el diseño propuesto la penalización debida a un salto mal predicho es de 20 ciclos.

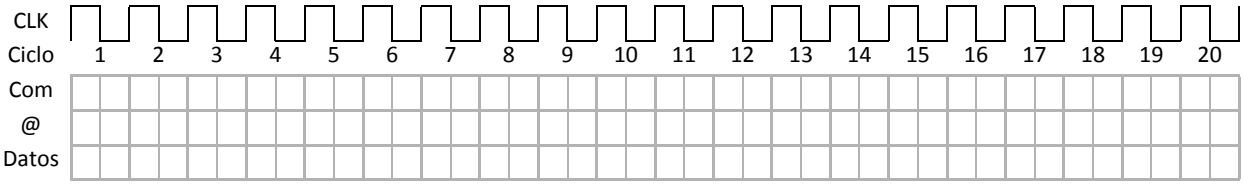
b) **Calcula** la tasa de fallos del predictor de saltos.

Finalmente se ha optado por un predictor de saltos más sofisticado, que usaremos en los siguientes apartados, con una tasa de aciertos del 95% con el que se obtiene un CPI de 0,7 ciclos/instrucción. Hasta ahora hemos considerado que la lectura de las instrucciones se realizaba de forma ideal. Sabemos que se ejecutan $10x10^9$ instrucciones dinámicas y que se realizan $4x10^9$ accesos a la cache de instrucciones, es decir se realizan sólo 0,4 accesos a la cache por instrucción. El numero de accesos a la cache es menor que el numero de instrucciones ejecutadas debido a que los procesadores superescalares leen varias instrucciones cada vez que acceden a la cache de instrucciones. Sabemos además que nuestra cache de instrucciones tiene una tasa de fallos de 0,05 fallos/acceso y la penalización es de 100 ciclos/fallo.

c) **Calcula** los ciclos perdidos debidos a los fallos en la cache de instrucciones y el CPI real del procesador.

El conjunto procesador-cache está conectado a un sistema de memoria principal mediante un único canal de 64 bits al que se ha conectado un DIMM de 4 Gbytes. Este DIMM tiene 8 chips de memoria DDR-SDRAM (Double Data Rate Synchronous DRAM) de un byte de ancho cada uno. El DIMM esta configurado para leer/escribir ráfagas de 64 bytes. La latencia de fila es de 5 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos.

d) **Rellena** el siguiente cronograma indicando la ocupación de los distintos recursos para una operación de lectura de un bloque de 64 bytes de la memoria DDR-SDRAM.

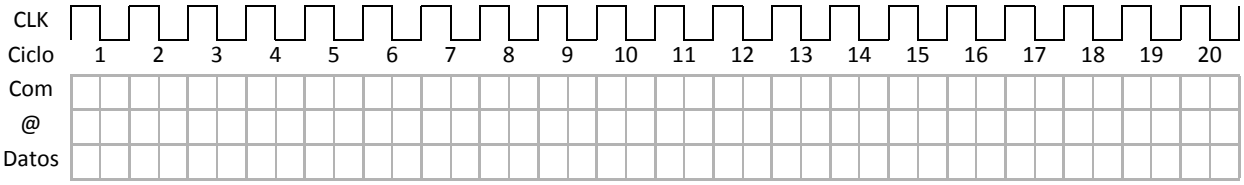


Esta DDR funciona a una frecuencia de 1 GHz. La potencia consumida por la DDR depende de la actividad. Durante un acceso a un bloque de 64 bytes se consumen 10 W (desde que se envía el comando ACTIVE hasta que se completa el PRECHARGE), mientras que el resto del tiempo consume sólo 1 W. Sabemos que el programa P ha tardado 9 segundos y se han realizado $0{,}2 \times 10^9$ accesos de 64 bytes a la DDR.

e) **Calcula** la potencia media consumida por la memoria DDR durante la ejecución del programa P.

Dado que el rendimiento obtenido hasta ahora es bastante pobre, se ha sugerido hacer prefetch de instrucciones. Se sugiere hacer prefetch en caso de fallo, de forma que en cada fallo se leerán dos bloques de 64 bytes cada uno, el que ha provocado el fallo y el siguiente. En este caso, el controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que ambos accesos sean servidos lo más rápidamente posible. Supongamos que los dos bloques están en la misma página de DRAM y esta no está abierta.

f) **Rellena** el siguiente cronograma indicando la ocupación de los distintos recursos para las dos operaciones de lectura de forma que se maximice el ancho de banda útil del bus.



Nuestro procesador traduce las instrucciones x86 a microoperaciones RISC (uops). Cada acceso a la cache consume 2,5 nJ (sea acierto o fallo) y cada instrucción x86 gasta 1 nJ en ser descodificada y traducida a uops. Recuerda que el programa P ejecuta 10×10^9 instrucciones dinámicas y realiza 4×10^9 accesos a la cache de instrucciones.

g) **Calcula** la energía consumida por la búsqueda y traducción de las instrucciones x86 durante la ejecución de P.

Se ha decidido introducir una cache de micro-ops donde se guardan las uops que ya han sido traducidas para que en caso de hit no sea necesario descodificar y traducir las instrucciones x86 de nuevo. Con el simulador se ha medido que para ejecutar P se generan 15×10^9 uops dinámicas y se realizan 5×10^9 accesos a la cache de uops. Gracias a esto se ha reducido el número de accesos a la cache de instrucciones a $0{,}4 \times 10^9$ accesos y el numero de instrucciones x86 descodificadas a 1×10^9 instrucciones. Un acceso a la cache de uops consume 1,6 nJ.

h) **Calcula** la energía consumida por el sistema con cache de uops durante la ejecución de P.

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

3er Control Arquitectura de Computadors **Curs 2013-2014 Q2**

Problema 3. (3 puntos)

Se ha ejecutado un programa P en un sistema con un solo procesador y un solo disco D (un PC de sobremesa que denominaremos PC1) y se ha visto que su tiempo de ejecución es de T horas. Para poder estimar el rendimiento del programa P hemos medido (en el PC1) que el programa se ejecuta en tres fases bien diferenciadas:

Fase 1: Código SECUENCIAL que no puede paralelizarse, ocupa el 18% del tiempo de la ejecución de P en el PC1.

Fase 2: Código PARALELIZABLE, ocupa el 64% del tiempo de la ejecución de P en el PC1.

Fase 3: Código de E/S que pasa todo su tiempo accediendo en el disco D, ocupa el 18% del tiempo de la ejecución de P en el PC1.

Con el objeto de reducir el tiempo de ejecución del programa, se baraja la opción de substituir el procesador del PC1 por un sistema multiprocesador de 16 procesadores idénticos al del PC1, manteniendo igual el resto del sistema. A este sistema multiprocesador le llamaremos PC2.

a) **Calcula** el máximo speed-up que podría conseguirse al ejecutar el programa P con el PC2.

Sabemos que el programa P tiene 10^5 instrucciones estáticas y ejecuta 10^{17} instrucciones dinámicas. En la Fase 2, la única que realiza cálculos en punto flotante, se ejecutan $800x10^{14}$ instrucciones, de las cuales $500x10^{14}$ son instrucciones de coma flotante que realizan un total de $200x10^{14}$ operaciones de coma flotante en simple precisión.

b) **Calcula** los MIPS y MFLOPS al ejecutar el programa P en el PC1 si la ejecución tarda $5x10^3$ segundos.

c) **Calcula** la ganancia máxima en MIPS y MFLOPS al ejecutar el programa P en el PC2 en vez de en PC1.

Otra opción que se ha barajado para mejorar el rendimiento del sistema PC1 es añadir un RAID de discos en lugar del disco duro D. El ancho de banda del disco D es de 250GBytes/s. El RAID nos permite paralelizar la Fase 3, ya que en esta fase hay suficientes accesos como para saturar el ancho de banda de todos los discos.

El RAID de discos del que disponemos tiene 6 discos y puede configurarse como **RAID 10** o **RAID 5**.

d) **Describe** las principales características de cada uno de estos sistemas RAID, dibujando un esquema de cómo se distribuyen los datos y especificando el tipo de entrelazado, el porcentaje de información redundante, el numero de discos que han de fallar para que el sistema deje de ser operativo, el ancho de banda **máximo** de las lecturas y el ancho de banda **máximo** de las escrituras.
NOTA: Considerar el mejor de los casos entre accesos secuenciales y aleatorios

Decidimos configurar el sistema de discos como RAID 5.

e) **Calcula** el speed-up máximo, **sobre el PC2**, que podemos esperar al ejecutar el programa P en el PC2 usando el RAID 5 de 6 discos en lugar del disco duro D, asumiendo que todos los accesos a disco son lecturas secuenciales.