COGNOMS:

NOM:

IMPORTANTE leer atentamente antes de empezar el examen: Escriba los apellidos y el nombre antes de empezar el examen. Escriba un solo carácter por recuadro, en mayúsculas y lo más claramente posible. Es importante que no haya tachones ni borrones y que cada carácter quede enmarcado dentro de su recuadro sin llegar a tocar los bordes. Use un único cuadro en blanco para separar los apellidos y nombres compuestos si es el caso. No escriba fuera de los recuadros.

Problema 1. (2 puntos)

Una **CPU** está conectada a una cache y una memoria principal formada por un único módulo DIMM estándar de 16 GBytes. Este DIMM tiene 8 chips de memoria **DDR**-SDRAM (**Double Data Rate** Synchronous DRAM) de 1 byte de ancho cada uno. La DDR-SDRAM tiene 8 bancos. El DIMM esta configurado para leer/escribir ráfagas de 64 bytes (justo el tamaño de bloque de las caches). La latencia de fila es de 3 ciclos, la latencia de columna de 4 ciclos y la latencia de precarga de 2 ciclos. Cuando se solicitan múltiples bloques a la DDR-SDRAM, el controlador de memoria envía los comandos necesarios a la DDR-SDRAM de forma que los bloques sean transferidos lo más rápidamente posible.

La siguiente tabla muestra en qué banco y qué página de DRAM (fila) se encuentran los bloques etiquetados con las letras A B C D E.

Bloque	A	B	C	D	E
Banco	0	0	1	1	5
Página	10	10	10	25	40

Rellena los siguientes cronogramas para la lectura de varios bloques de 64 bytes (**sin cambiar el orden de los accesos**), en función de la ubicación de los bloques involucrados de forma que se minimice el tiempo total. Indica la ocupación de los distintos recursos de la memoria DDR: bus de datos, bus de direcciones y bus de comandos. En todos los cronogramas supondremos que no hay ninguna página de DRAM abierta previamente y que al final deben quedan todas cerradas.

a) **Rellena** el siguiente cronograma para la lectura de los bloques AB.

Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Com	A C			R D				R D								P R	- -										
@	@F			@C				@C																			
Datos								D D	D D	D D	D D	D D	D D	D D	D D												

b) **Rellena** el siguiente cronograma para la lectura de los bloques AC.

Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Com	A C			R D	A C			R D				P R	- -			P R	- -										
@	@F			@C	@F			@C																			
Datos								D D	D D	D D	D D	D D	D D	D D	D D												

c) **Rellena** el siguiente cronograma para la lectura de los bloques CD.

Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Com	A C			R D								P R	- -	A C			R D								P R	- -	
@	@F			@C										@F			@C										
Datos								D D	D D	D D	D D										D D	D D	D D	D D			

d) **Rellena** el siguiente cronograma para la lectura de los bloques ADBE.

Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
Com	A C			R D	A C			R D				R D	A C			R D	P R	- -		P R	- -			P R	- -		
@	@F			@C	@F			@C				@C	@F														
Datos								D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D	D D				

COGNOMS:

NOM:

Problema 2. (3 puntos)

Un procesador funciona a una frecuencia de 1GHz y esta conectado a una cache de datos nivel 1 (L1) 2-asociativa de 16 KBytes de capacidad con bloques de 64 bytes. La cache no tiene ninguna de las mejoras vistas en clase. En caso de acierto, los accesos a la cache tardan 1 ciclo y en caso de fallo tenemos una penalización media de 20 ciclos. Hemos determinado que la cache se encuentra en el camino crítico del procesador, por lo que la frecuencia (y por tanto el tiempo de ciclo) del procesador vienen determinados por el diseño de la cache.

En la siguiente tabla, las filas indican posibles mejoras (vistas en clase) que podríamos introducir en L1. Las columnas indican algunos parámetros de rendimiento en los que estas mejoras podrían influir, los tiempos tanto de acceso como de penalización se consideraran en **ns** y no en ciclos ya que el tiempo de ciclo puede cambiar. Cada mejora será considerada de forma individual, sin cambiar ningún otro parámetro de la cache. Para indicar el impacto de las mejoras en los parámetros de rendimiento rellenad la tabla usando la siguiente nomenclatura: (**M**) Mejora, (**NA**) No Afecta, (**E**) Empeora.

a) **Indica** el impacto de las mejoras en los parámetros de rendimiento.

	Parámetro de rendimiento			
Mejora/cambio	Frecuencia del procesador	Tiempo (ns) de acceso a la cache L1 en caso de acierto	Tasa de fallos de la cache L1	Tiempo (ns) medio de penalización en caso de fallo
Cambiar a cache Directa de 16 KBytes	M	M	E	NA
Añadir una cache L2 de 256 KBytes	NA	NA	NA	M
Aumentar el tamaño de la cache a 32 KBytes	E	E	M	NA
Añadir continuación anticipada	NA	NA	NA	M
Segmentar la cache en 2 etapas	M	E	NA	NA
Organizar la cache en 4 bancos	M	M	NA	NA
Añadir un mecanismo de prefetch hardware	NA	NA	M	M
Hacer la cache no bloqueante	NA	NA	NA	M

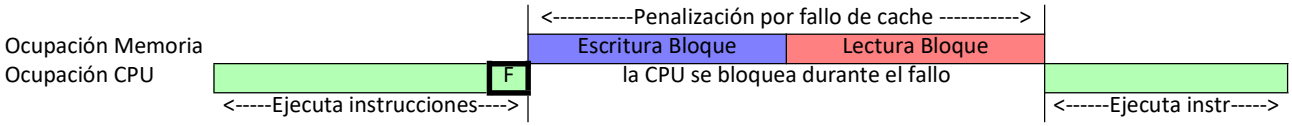
Nota 1: El apartado vale 1.5 puntos, cada fallo o cuadro en blanco resta 0.1 puntos en el apartado (mínimo 0).

Nota 2: La decisión de si un cambio mejora, empeora o no afecta a un parámetro debéis basarla en el caso general. Por ejemplo: aunque en determinadas circunstancias un predictor de vía podría empeorar el consumo de energía, en general lo mejorará.

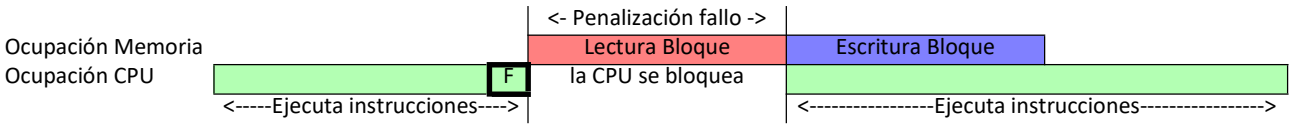
Finalmente se ha decidido estudiar el impacto de las políticas de escritura.

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).

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 hasta que se resuelve el fallo. 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 reemplazados han sido modificados (dirty bit = 1). La siguiente figura muestra el cronograma de un fallo en que la línea reemplazada ha sido modificada.



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

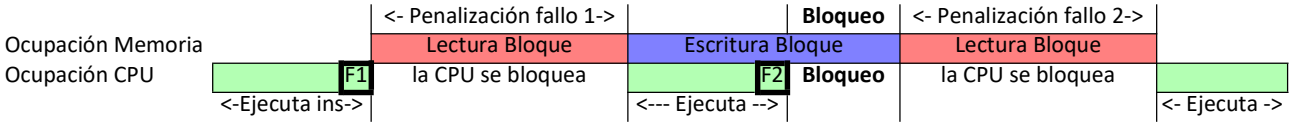
b) **Calculad** el numero de ciclos necesario para ejecutar P en el procesador B

penalización por fallo = 15 ciclos

$ciclos_B = ciclos_{IDEAL}$ + #fallos * Penalización per fallo

$ciclos_B = 5x10^9$ ciclos + $50x10^6$ fallos * 15 ciclos/fallo = $5,75\ x10^9$ ciclos

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, todos los ciclos tienen la misma probabilidad de que se produzca un nuevo fallo y sabemos que el número medio de ciclos en que la CPU ejecuta instrucciones (sin contar los bloqueos) entre dos fallos es de 100 ciclos.

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

probabilidad de fallar en un ciclo es p = 1/100 (inversa del tiempo medio entre fallos)

probabilidad de tener un fallo en 15 ciclos (intervalo de servicio de F1) es 1 - probabilidad de no fallar en ningún ciclo (repetimos un proceso independiente 15 veces con probabilidad p)

P(fallo en el intervalo) = $1 - (1 - p)^{15} = 1-(1-1/100)^{15} = 0,14$

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 (correspondientes al intervalo **Bloqueo** de la figura anterior) antes de que se pueda iniciar la lectura del bloque que ha causado el segundo fallo.

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

penalización por bloqueo = 7 ciclos

Solo hay bloqueo con probabilidad P si el bloque ha sido modificado

$ciclos_C = ciclos_B$ + #fallos * bloques modificados *probabilidad bloqueo * Penalización media por bloqueo

$ciclos_C = 5,75\ x10^9$ ciclos + $50x10^6$ fallos * 1/3 * 0,14 bloqueos/fallo * 7 ciclos/bloqueo = $5,77\ x10^9$ ciclos

COGNOMS:

NOM:

Problema 3. (1.5 puntos)

Queremos estudiar el efecto de la búsqueda y decodificación de instrucciones en el rendimiento de un procesador segmentado superescalar con ejecución fuera de orden. Para ello, simulamos la ejecución de un programa de prueba P en un simulador parametrizable. Para simplificar el problema, nos centraremos exclusivamente en los accesos a instrucciones, y supondremos que en los accesos a datos nunca se produce un fallo al acceder a la cache de datos.

En los procesadores segmentados pueden transcurrir decenas de ciclos hasta que se calcula la dirección destino del salto y el procesador decide si el salto se realizará (*taken*) o por el contrario el programa seguirá ejecutándose en secuencia (*not taken*). La alternativa más simple consiste en ejecutar por defecto las instrucciones de forma secuencial, con lo que los saltos *not taken* no incurren en ninguna penalización. Sin embargo, en nuestro caso los saltos *taken* incurren en una penalización de 10 ciclos. A este procesador le llamaremos PSeq.

Con el simulador hemos obtenido que el 30% de las instrucciones dinámicas del programa P son saltos, y que el 80% de los saltos son *taken*. También hemos obtenido que, en un procesador ideal donde los saltos *taken* no penalizan, el CPI sería de 0,6 ciclos/instrucción.

a) **Calcula** el CPI del procesador PSeq.

CPI = 0,6 c/i + 0,3 saltos/ins * 0,80 taken/salto * 10 ciclos/taken = 3 ciclos/instrucción

Para mejorar el rendimiento del procesador se ha decidido incorporar un predictor de saltos al procesador PSeq, con el que se ha obtenido un CPI de 0,93 ciclos/instrucción. En este diseño, la penalización debida a un salto mal predicho es de 22 ciclos, mientras que los saltos bien predichos no tienen penalización.

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

0,93 c/i = 0,6 i/c + 0,3 saltos/ins * TF fallos/salto * 22 ciclos/fallo
TF = 0,05 fallos/salto

Finalmente, se ha optado por usar un predictor de saltos más sofisticado que tiene una tasa de aciertos del 98% con el que se obtiene un CPI de 0,8 ciclos/instrucción. Hasta ahora no hemos considerado los fallos en la cache de instrucciones. Los procesadores superescalares leen varias instrucciones cada vez que acceden a la cache de instrucciones, por lo que el número de accesos a la cache es menor que el numero de instrucciones ejecutadas. Sabemos que en P se ejecutan $10x10^9$ instrucciones dinámicas y que se realizan $5x10^9$ accesos a la cache de instrucciones, es decir, se realizan sólo un 50% de accesos a la cache por instrucción. Sabemos además que la cache de instrucciones tiene una tasa de fallos de 0,04 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.

Ciclos = $5x10^9$ accesos * 0,04 fallos/acceso * 100 ciclos/fallo = $20x10^9$ ciclos
CPI = 0,8 c/i + $20x10^9$ ciclos / $10x10^9$ instrucciones = 2,8 ciclos/instrucción

COGNOMS:

NOM:

Problema 4. (3,5 puntos)

La empresa ACME nos ha contratado para reducir el tiempo de ejecución de una aplicación. Hemos ejecutado dicha aplicación en un ordenador PC1 con un solo procesador y un solo disco, y hemos comprobado que la aplicación ejecuta de forma iterativa tres fases bien diferenciadas, tal como se ve en la figura:

```
for (;;) {
  LeerDatos();
  ProcesarDatos();
  EscribirResultados();
}
```

• **Fase 1:** LeerDatos(), únicamente se hacen lecturas desde disco. Ocupa el 25% de toda la ejecución en el PC1.

• **Fase 2:** ProcesarDatos(), parte intensiva en cálculo que es totalmente paralelizable. Ocupa el 60% de la ejecución en el PC1.

• **Fase 3:** EscribirResultados(), los datos recién calculados se escriben a disco. Ocupa el 15% de la ejecución en el PC1.

El tiempo de ejecución de la aplicación en el PC1 es de 120 horas. Queremos estudiar diferentes alternativas que nos permitan reducir este tiempo.

a) **Calcula** el Speedup máximo que podríamos obtener si ejecutáramos este programa en un supercomputador con un número infinito de procesadores y un único disco.

S = 1 / (1-0.6) = 2,5

b) **Calcula** con cuántos procesadores tenemos que ejecutar el programa para obtener un Speedup total de 2.

El número de procesadores es igual al speed up que se obtendrá en la fase 2
S = 1 / (1-0.6+0,6/p) = 2 --> p = 6

Dado que gran parte de la aplicación puede paralelizarse completamente, ofrecemos a la empresa reemplazar el ordenador PC1 por un nuevo ordenador PC6 con un procesador multi-core de 6 núcleos.

Hemos comprobado que el 80% de la Fase 2 se dedica a hacer operaciones matemáticas entre varias matrices de enteros de 4 bytes en las que cada cálculo es independiente de los demás, así que decidimos transformar el código en esas regiones para poder utilizar instrucciones SIMD. La arquitectura de los núcleos del procesador nos permite utilizar registros xmm de 128 bits.

c) **Calcula** el Speedup de la Fase 2 después de aplicar la transformación en el código para usar instrucciones SIMD y de ejecutarlo en el PC6. **Calcula** el tiempo de ejecución de la aplicación después de estas mejoras.

$S_{fase2+SIMD}$ = 1 / (0.2/6+0.8/(6*4)) = 15

S = 1 / (1-0.6+0.6/15) = 2,27
T = 120h / 2,27 = 52,86 h

Para reducir el tiempo de las fases de lectura/escritura de datos necesitaremos un sistema de almacenamiento que nos proporcione mayor ancho de banda que el actual. Además, queremos añadir algún mecanismo de tolerancia a fallos en disco. Valoramos utilizar los sistemas de almacenamiento RAID 10 (con mirror doble), RAID 5 y RAID 6. La aplicación necesita un total de 24 terabytes de almacenamiento.

Para implementar los diversos sistemas RAID disponemos de un único modelo de disco duro DD de 4TB con un ancho de banda de 100MB/s, tanto en lectura como en escritura.

d) **Calcula** cuántos discos duros DD necesitaríamos para implementar cada uno de los niveles de RAID 10 (mirror doble), RAID 5 y RAID 6 si queremos almacenar al menos 24TB útiles

RAID 10 -> 24 TB * 2 mirrors / 4 TB/disco = 12 discos
RAID 5 -> 24 TB / 4 TB/disco + 1 paridad = 7 discos
RAID 6 -> 24 TB / 4 TB/disco + 2 paridad = 8 discos

e) Después de explicar a la empresa ACME las diferencias entre los tres niveles de RAID, la empresa se decanta por montar un RAID5 con 8 discos duros DD. Dibuja el esquema del RAID 5 que nos ha pedido la empresa indicando claramente el tipo de entrelazado, cómo se distribuye la paridad entre los discos, y el ancho de banda máximo que se conseguirá en el RAID5 en lectura secuencial, en escritura secuencial y en escritura aleatoria.

Entrelazado a nivel de tira,
AB lectura secuencial = 8 discos * 100 MB/s /disco = 800 MB/s
AB escritura secuencial = 7 discos * 100 MB/s /disco = 700 MB/s
AB escritura aleatoria = 8 discos * 100 MB/s /disco / 4 accesos/escritura= 200 MB/s

f) **Calcula** el Speedup y el tiempo total que tarda la aplicación respecto a ejecutarla en el PC1 al ejecutarla en el PC6 + SIMD con un RAID 5 de 8 discos), teniendo en cuenta que todos los accesos a disco son secuenciales.

S_{f1} = 800 / 100 = 8

S_{f2} =15

S_{f3} = 700 / 100 = 7

S = 1 / (0.25/8 + 0.6/15 + 0.15/7) = 10,79

T = 120h / 10,79 = 11,12h