Licencia Creative Commons

Toda la documentación de Arquitectura de Computadores esta bajo licencia Creative Commons :

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

You are free:

- **to Share — to copy, distribute and transmit the work**
- **to Remix — to adapt the work**

Under the following conditions:

- **Attribution — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).**
- **Noncommercial — You may not use this work for commercial purposes.**
- **Share Alike — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.**

Consideraciones generales sobre los problemas

Representación de los resultados numéricos

Por norma general en todos los problemas de la asignatura se seguirán las siguientes convenciones para representar los resultados:

- Apartados cuyo resultado sólo puede ser un **número entero.** Se escribirá el **resultado exacto**.
- Apartados cuyo resultado puede ser un **número real.** Se escribirá el resultado usando la **notación de ingeniería normalizado y redondeado con 3 dígitos significativos**. La notación usada en ingeniería es muy parecida a la notación científica excepto que siempre se usan potencias de 10 cuyo exponente es un múltiplo de 3. La ventaja es que la conversión a unidades del sistema internacional es trivial, como microsegundos (10^{-6}segundos) o gigahercios (10^{9} ciclos/segundo).

Ejemplos:

Medida	Valor (aproximado)	Notación científica	Notación ingeniería	Sistema internacional
masa del electrón	0,00000000000000000000000000091093822 g	$9{,}11x10^{-28}$g	$911x10^{-30}$g	no hay prefijo
carga del electrón	0,0000000000000000001602 C	$1{,}60x10^{-19}$C	$160x10^{-21}$C	160 zC zepto coulombs
masa de la luna	73600000000000000000000000 g	$7{,}36x10^{25}$g	$73{,}6x10^{24}$g	73,6 Yg yotta gramos
circunferencia de la tierra	40000000 m	$4{,}00x10^{7}$m	$40{,}0x10^{6}$m	40 Mm mega metros

Usa el sentido común:

Si un programa se ejecuta en medio segundo es aceptable (e incluso recomendable) escribir *0,5 segundos* en lugar de la versión normalizada ($500x10^{3}$ segundos) ya que visualmente nos da una idea rápida de que es medio segundo. Sin embargo, si el programa se ejecuta en 4 microsegundos, debemos escribir $4x10^{-6}$ segundos (jamás escribiremos *0,000004 segundos*)

Si un programa ejecuta 2 billones de instrucciones, escribiremos $2x10^{12}$ *instrucciones*, aunque el resultado solo pueda ser un número entero (jamás escribiremos *2000000000000 instrucciones*)

Justifica las respuestas

En general, no basta con escribir el resultado, hay que **justificar la respuesta** y poner **explicitamente las unidades** del resultado. Aunque el resultado de un problema sea correcto, en los exámenes y controles, no se considera bien hecho a menos que esté debidamente justificado. La justificación no tiene porque ser larga, en el siguiente ejemplo las respuestas están adecuadamente justificadas y se ha especificado correctamente las unidades.

Problema Ejemplo:

Un fragmento de código ejecuta 37523 instrucciones, provoca 1532 fallos de cache y tarda 30 microsegundos en ejecutarse. ¿Cómo responderíamos a las siguientes preguntas?

a) ¿Cuantos fallos de cache provoca este fragmento de código? *(solo puede haber un número entero de fallos)*

según el enunciado ***1532 fallos***

b) ¿Cuantas instrucciones se ejecutan por segundo? *(el resultado puede ser un numero real)*

37523 instrucciones / 30x10^{-6} segundos = ***1,25x10^{9} instrucciones / segundo***

c) ¿Cuantos fallos por instrucción se producen? *(el resultado puede ser un numero real)*

1532 fallos / 37523 instrucciones = ***40,8x10^{-3} fallos / instrucción***

Tabla de prefijos del sistema internacional:

10^n	prefijo	Símbolo	Equivalencia decimal
10^{24}	yotta	Y	1 000 000 000 000 000 000 000 000
10^{21}	zetta	Z	1 000 000 000 000 000 000 000
10^{18}	exa	E	1 000 000 000 000 000 000
10^{25}	peta	P	1 000 000 000 000 000
10^{12}	tera	T	1 000 000 000 000
10^{9}	giga	G	1 000 000 000
10^{6}	mega	M	1 000 000
10^{3}	kilo	K	1 000
10^{0}	-	-	1
10^{-3}	mili	m	0,001
10^{-6}	micro	µ	0,000 001
10^{-9}	nano	n	0,000 000 001
10^{-12}	pico	p	0,000 000 000 001
10^{-15}	femto	f	0,000 000 000 000 001
10^{-18}	atto	a	0,000 000 000 000 000 001
10^{-21}	zepto	z	0,000 000 000 000 000 000 001
10^{-24}	yocto	y	0,000 000 000 000 000 000 000 001

Notas sobre los ejercicios

- El número entre paréntesis (si lo hay) indica la dificultad del ejercicio (1) básico, (2) medio y (3) avanzado.
- Detrás de los paréntesis aparecen los conceptos que se tratan en cada ejercicio.

Problemas Tema 1

Problema 1. (1) Texe, CPI, F, Tc, Num instrucciones dinámicas

Dados dos procesadores con las siguientes características:

	CPI	Frecuencia
Procesador A	1,2 ciclos / instrucción	2 GHz
Procesador B	1,5 ciclos / instrucción	3 GHz

a) **Calcula** el tiempo de ciclo de cada procesador

b) Suponinedo que un programa P ejecuta $2 \cdot 10^6$ instrucciones dinámicas en ambos procesadores, ¿cual es el tiempo de ejecución del programa P en cada procesador?

c) Si un programa X se compila y ejecuta en el procesador B en 1s, ¿cuántas instrucciones ejecuta?

d) Si sabemos que el procesador A ejecuta el programa X un 25% más rápido que el procesador B, ¿cuántas instrucciones dinámicas son necesarias para ejecutar el programa X en el procesador A?

e) Suponiendo que todas las instrucciones (tanto de A como de B) se codifican en 4 bytes, ¿cuánto ocupa el programa X compilado para el procesador A?

Problema 2. (2) Texe, Ley de Amdahl, SpeedUp, Capacidad de abstracción

Dado un programa que se ejecuta durante tres fases bien diferenciadas en un procesador Load/Store parecido al MIPS que funciona a 1 GHz, se mide su rendimiento en las distintas fases con los siguientes resultados:

	Instrucciones Dinámicas	Instrucciones Estáticas	CPI	Instrucciones Dinámicas de acceso a Memoria
Fase 1	10^6 instrucciones	10^6 instrucciones	2 ciclos / instrucción	10^6 instrucciones
Fase 2	10^9 instrucciones	10^6 instrucciones	3 ciclos / instrucción	10^7 instrucciones
Fase 3	10^9 instrucciones	10^6 instrucciones	4 ciclos / instrucción	10^7 instrucciones

a) ¿Cuanto tiempo tarda en ejecutarse el programa en cuestión?

b) ¿Es un programa intensivo en memoria o en cálculo?

c) Si hacemos cambios en la arquitectura de forma que las instrucciones de la Fase 3 se ejecuten un 25% más rápido (y suponemos que no afectamos al resto de instrucciones), ¿cuál es la ganancia para el conjunto del programa?

d) ¿Cual es el CPI de las instrucciones de acceso a memória (Load/Store) en la Fase 1? y ¿Cuántas veces más rápidas deberían ser las instrucciones de memòria para que la Fase 1 tardase la mitad en ejecutarse?

e) Suponiendo que el CPI de las instrucciones de acceso a memoria es el mismo que en la fase 1, ¿Cuántas veces más rápidas deberían ser las instrucciones de memòria para que el programa tardase la mitad en ejecutarse?

Problema 3. (2) Potencia, Ley de Amdahl

Dados los siguientes tiempos de ejecución de programas en dos procesadores de la misma familia:

	Procesador A	Procesador B	Instrucciones dinámicas
Programa 1	3 s	4 s	10^9 instrucciones
Programa 2	3,5 s	4,5 s	$1{,}5 \cdot 10^9$ instrucciones
Programa 3	2,5 s	1,5 s	$0{,}5 \cdot 10^9$ instrucciones

Suponiendo que el consumo de potencia del procesador A es 60W y del procesador B es 50W.

a) ¿Qué programa usa instrucciones que consumen menos energía?

b) ¿Cuál de los dos procesadores es más eficiente energéticamente si tenemos en cuenta todos los programas?

c) Si suponemos que para el procesador A conseguimos reducir el tiempo de ejecución de todos los programas en un 10% a cambio de aumentar 10W el consumo de energía ¿Esta propuesta es energéticamente eficiente?

d) ¿Cuánto deberíamos acelerar el programa 3 en el procesador A para conseguir que la eficiencia energética aumentase en un 20%? ¿Y en el procesador B?

Problema 4. (2) MIPS, MFLOPS, Texe, Energía

Dado el siguiente programa en C:

```
double v1[100000],v2[100000];
int i;
main(){
   for (i=100000;i>0;i--)
      v1[i]=v1[i]+v2[i];
}
```

Lo ejecutamos en una máquina de arquitectura x86 y otra de arquitectura MIPS con los resultados que se ven en la siguiente tabla:

	Instrucciones dinámicas	Texe	Consumo medio
x86	300.005 instrucciones	0,03 s	120 W
MIPS	500.005 instrucciones	0,04 s	80 W

a) **Calcula**, para cada máquina, los MIPS y MFLOPS al ejecutar el código de prueba. ¿Qué máquina es más rápida? ¿Cuál es más eficiente energéticamente?

A continuación cambiamos el código para que multiplique los vectores en lugar de sumarlos. Al ejecutar de nuevo el programa vemos que el tiempo de ejecución aumenta en 0,02s en el x86 y se dobla en el MIPS.

b) ¿Cómo cambian los resultados del apartado anterior con este nuevo código? ¿Cuál de las dos máquinas es mejor?

Problema 5. (2) Consumo energético, Sostenibilidad

En una oficina disponen de 11 equipos informáticos. Uno es un servidor mientras que los otros 10 son equipos de sobremesa para los trabajadores. Sus consumos y los de sus equivalentes más modernos están en la tabla que aparece a continuación:

	Encendido y Procesado	Encendido en espera	Suspendido en RAM	Apagado	Desenchufado
Servidor	200 W	100 W	10 W	2 W	0 W
Sobremesa	150 W	100 W	5 W	2 W	0 W
Pantalla	40 W	2 W	-	2 W	0 W
Servidor (nuevo)	150 W	80 W	10 W	2 W	0 W
Sobremesa (Nuevo)	100 W	80 W	5 W	2 W	0 W
Pantalla (LED)	20 W	1 W	-	1 W	0 W

Sabemos que el servidor está encendido siempre (360 días al año) y aproximadamente un 33% del tiempo está procesando. Los equipos de sobremesa en cambio están encendidos tan solo 8 horas al día los días laborables (suponed que son 220 al año). Sin embargo sistemáticamente alguno queda encendido así que en media 1 está siempre encendido y 9 sólo cuando toca. Del tiempo que están encendidos los sobremesas (incluido el que nunca se apaga) el 75% están procesando. Podéis suponer que las pantallas de éstos están activas el mismo tiempo que el

ordenador procesa y en espera el resto del tiempo. La pantalla del servidor en cambio solo está en activo unas 96 horas al año y el resto en espera. Suponiendo que el precio del kWh es de 0,10€.

a) ¿Cuál es la factura eléctrica de la empresa?
b) ¿Qué componente es el que más electricidad gasta en la empresa?
c) ¿Qué porcentaje de ahorro de energía obtendríamos si cambiáramos solo las pantallas? ¿Y si cambiáramos solo el servidor?
d) ¿Cuánto dinero podríamos ahorrar cambiando todos los equipos de la empresa?
e) ¿Cuándo dinero ahorraríamos si desenchufáramos todos los sobremesas siempre que acaba el horario laboral?
f) Si tú fueras el dueño de la empresa ¿Qué medidas propondrías para ahorrar energía?

Problema 6. (2) CPI, MIPS, MFLOPS, Amdahl

Hemos realizado un estudio sobre el comportamiento de un conjunto de programas representativo de un entorno de trabajo departamental en el procesador X. Este estudio nos dice el porcentaje de instrucciones de cada tipo que se ejecutan y su coste en ciclos de reloj. Esta información se resume en la siguiente tabla:

Tipo de instrucciones	% de uso	Coste en ciclos
Aritméticas de enteros	30%	2 ciclos
Acceso a Memoria	30%	5 ciclos
Coma Flotante	15%	7 ciclos
Saltos	15%	3 ciclos
Otras	10%	4 ciclos

Además, sabemos que en media se realizan 2 operaciones de coma flotante por cada instruccion de coma flotante ejecutada.

a) **Calcula** el CPI medio para el procesador X.
b) Suponiendo que el procesador X funciona a una frecuencia de 2 GHz calculad los MIPS y MFLOPS que obtendriamos en dicho conjunto de programas.

Un estudio más específico nos indica que una de las tareas que más tiempo consume es la gestión de procedimientos y funciones (subrutinas en general). Uno de los ingenieros de la compañía ha diseñado una modificación del procesador X (que llamaremos X1) con el objetivo de reducir el coste de la gestión de subrutinas. Los experimentos realizados con X1 revelan lo siguiente:

- La duración del ciclo de reloj del procesador X1 aumenta un 5%.
- El procesador X1 ejecuta un 25% menos de instrucciones de acceso a memoria que el procesador X.
- El procesador X1 ejecuta un 15% menos de instrucciones aritméticas con enteros que el procesador X.
- Para el resto de instrucciones el recuento queda inalterado.

c) ¿Qué procesador es más rápido? Justificad cuantitativamente la decisión.
d) **Calcula** los MIPs y MFLOPs para X1.

Problema 7. (2) CPI, MIPS, Amdahl

Un procesador trabaja a 1.5 GHz con un rendimiento de 950 MIPS ejecutando una determinada aplicación.

a) ¿Cuál es el número de ciclos por instrucción en este procesador?
b) Suponed que se mejora la frecuencia del procesador a 3 GHz, sin modificar la velocidad de la memoria (ahora todos los accesos a memoria de datos requieren 1 ciclo adicional). Si el 30% de las instrucciones necesita 1 acceso a memoria de datos y el 5% 2 accesos a memoria de datos, ¿cuál es el rendimiento en MIPS del nuevo procesador?

Problema 8. (2) Ganancia, Amdahl

Un fabricante ha sacado al mercado una nueva versión de su procesador en el que la única mejora es que permite reducir el tiempo de ejecución de las operaciones en coma flotante a un 25% del tiempo que consumían antes.

Suponiendo que en los programas que se ejecutan habitualmente en este procesador las operaciones coma flotante consumen un 20% del tiempo de ejecución, se pide lo siguiente:

a) ¿Cuál es la ganancia de velocidad que puede obtenerse ejecutándose estos programas? (Nota: ganancia = tiempo ejecución original / tiempo ejecución mejorada)

b) ¿Cuál es la ganancia máxima de velocidad que puede obtenerse debido a mejoras en la velocidad de las operaciones en coma flotante?

c) En la versión original del procesador, ¿cuál debería ser el porcentaje de tiempo de cálculo con datos en coma flotante en sus programas para obtener una ganancia de 4?

Problema 9. (2) Coste, Energía, Consumo, Sostenibilidad

Un procesador tiene una superficie de 200 mm2 y se fabrica en una oblea de silicio con una superficie útil de 63.200 mm2. El coste de una oblea y el proceso de impresión y verificación de los dados (dies) es de 23.700 €. Durante este proceso el factor de yield es del 75%.

a) ¿Cual es el coste de un dado (die)?

El coste empaquetado y testeado final es de 20€ por dado y el yield final de los circuitos integrados después del testeado final es del 92%. El fabricante quiere obtener un 50% de beneficio sobre el coste de fabricación.

b) ¿En media, cuantos circuitos integrados funcionales se producen por oblea?

c) ¿Cual será el precio de venta de los procesadores?

Se denomina “embodied energy” al coste energético de producir un producto. Este coste no solo incluye la energía consumida durante la fabricación sino también la energía consumida en producir los componentes, transporte, embalaje y comercialización del producto. Ignorar este coste energético puede conducir a mucha gente a pensar que sustituir un dispositivo más antiguo por uno más moderno con un menor consumo va a redundar en un ahorro energético y por consiguiente es beneficioso desde el punto de vista medioambiental. Sin embargo debemos tener en cuenta la “embodied energy” para determinar si dicha sustitución resulta beneficiosa.

Deseamos sustituir nuestros antiguos procesadores (que cumplen perfectamente con su tarea) por el procesador mencionado anteriormente ya que creemos que será positivo desde el punto de vista ambiental. Se ha estimado que la “embodied energy” del nuevo procesador es de 200 MJoules. En la siguiente tabla se muestra el uso medio y los consumos de ambos procesadores usados en entorno de sobremesa y servidor.

Estado	uso diario sobremesa	uso diario servidor	consumo procesador viejo	consumo procesador nuevo
Pleno rendimiento	2 horas	10 horas	50W	40W
Inactivo/bajo rendimiento	7 horas	14 horas	10W	5W
suspendido/apagado	15 horas	0 horas	0W	0W

Para un uso en entorno sobremesa:

d) Calcular el consumo anual de ambos procesadores (en Mjoules/año)

e) ¿durante cuanto tiempo debemos tener el nuevo procesador para amortizar la “embodied energy”?

Para un uso en entorno servidor:

f) Calcular el consumo anual de ambos procesadores (en Mjoules/año)

g) ¿durante cuanto tiempo debemos tener el nuevo procesador para amortizar la “embodied energy”?

h) ¿Cual crees que sería la mejor decisión en un entorno en donde tenemos computadores de los dos tipos (sobremesa y servidor)?

Los fabricantes de procesadores suelen cambiar el “socket” cada poco tiempo (2-3 años) e incluso tienen varias lineas distintas simultáneamente con lo que si nuestro procesador antiguo tiene 3 años o más es muy probable que para usar el nuevo tengamos que cambiar también la placa base y posiblemente la memoria. Se ha estimado que la

embodied energy de CPU+placa+memoria es de unos 2000 Mjoules para entorno sobremesa y 3000 Mjoules para entorno servidor. La siguiente tabla muestra el consumo del conjunto para ambos entornos.

Estado	uso diario sobremesa	consumo conjunto viejo	consumo conjunto nuevo	uso diario servidor	consumo conjunto viejo	consumo conjunto nuevo
Pleno rendimiento	2 horas	100W	80W	10 horas	120W	100W
Inactivo/bajo rendimiento	7 horas	30W	20W	14 horas	40W	30W
suspendido/apagado	15 horas	10W	5W	0 horas	10W	5W

i) ¿Porque crees que el entorno servidor tiene un mayor coste y mayores consumos a pesar de usar la misma CPU?

j) ¿Durante cuanto tiempo debemos tener el nuevo conjunto (CPU+placa+memoria) para amortizar la "embodied energy"? (para ambos sistemas)

k) Razona bajo que circunstancias se podria considerar ético o no, desde un punto de vista sostenible, un cambio de socket por parte de un fabricante.

Problema 10. (2) Ancho de banda, Latencia, Productividad, Amdhal

El tiempo total para realizar un acceso a un bloque de datos de un disco duro es:

Tiempo de acceso = Tiempo de búsqueda + Tiempo de posicionamiento + Tiempo de transferencia

Tenemos un disco duro conectado a un sistema a través de un bus SATA II que ofrece un ancho de banda teórico máximo de 300 MB/s. El Tiempo de búsqueda promedio de este disco es de 8 ms y el Tiempo de posicionamiento promedio es de 1 milisegundo. Una aplicación P1 realiza lecturas de bloques de información de 300 Kbytes (suponer 1Kbyte = 10^3 bytes) cuyos sectores de disco están consecutivos en la misma pista, por lo que el tiempo de búsqueda y el de posicionamiento solo se pierde una vez por cada bloque de 300 Kbytes (y no por cada uno de los sectores que lo componen).

a) **Calcula** cuanto tiempo tarda el bus en transferir un bloque de información de 300 Kbytes de disco (en milisegundos)

b) **Calcula** la latencia de un acceso a disco que lee un bloque de 300 Kbytes de disco (en milisegundos)

c) **Calcula** el ancho de banda real obtenido al leer un bloque de información de 300 Kbytes de este disco duro suponiendo que el disco es capaz de realizar la transferencia a la velocidad del bus (en Mbytes/s)

En este sistema ejecutamos el programa P1 que requiere 1 segundo de CPU (tiempo de usuario + tiempo de sistema). El programa P1 realiza una lectura de disco de 300 Kbytes cada 20 milisegundos. Al mismo tiempo que el programa P1 se ejecuta también otro programa P2. El programa P2 requiere 1 segundo de CPU (tiempo de usuario + tiempo de sistema) y el tiempo dedicado a entrada salida es negligible.

d) **Calcula** el tiempo total de ejecución del programa P1 (Wall Time)

e) **Calcula** el tiempo que se tarda en ejecutar concurrentemente los programas P1 y P2 y la productividad del sistema, suponiendo que mientras P1 realiza lecturas de disco podemos aprovechar para ejecutar P2 (asumir que el tiempo de cambio de contexto es negligible).

Después de cambiar los discos duros por discos nuevos, la latencia de leer un bloque de 300 Kbytes ha mejorado en un factor de 2.

f) **Calcula** el factor de mejora (speedup) para el tiempo de ejecución de P1 y para la productividad del sistema ejecutando P1 y P2.

Problema 11.

A pleno rendimiento, una CPU funciona a una frecuencia de 3 GHz y está alimentada a 1,6 V. En modo bajo consumo la CPU funciona a una frecuencia de 1 GHz y está alimentada a 1 V. Hemos medido que el consumo de la CPU en alto rendimiento es de 120W y en modo bajo consumo es de 27,5 W. En estos datos solo se considera la potencia debida a conmutación y la debida a fugas. Tanto la corriente de fugas (I) como la carga capacitiva equivalente (C) son las mismas en ambos modos.

a) **Calcula** la corriente de fugas (I) y la carga capacitiva equivalente (C) de la CPU (usar prefijo más adecuado del SI)

Problema 12. (1) fiabilidad, disponibilidad

Tenemos un sistema compuesto por los elementos mostrados en la tabla siguiente. La tabla también muestra el numero de componentes de cada tipo y el tiempo medio hasta fallo (MTTF) de cada componente.

Componente	Fuente alimentación	CPU	Placa base	DIMM	GPU	Disco duro
Nº	1	1	1	4	1	8
MTTF (horas)	125.000	1.000.000	200.000	1.000.000	500.000	100.000

a) **Calcula** el tiempo medio hasta fallos del sistema
b) El tiempo medio para reemplazar un componente que ha fallado (MTTR) es de 20 horas. Calcula el tiempo medio entre fallos (MTBF).
c) ¿Cual es la disponibilidad del sistema?

Problema 13. (2) fiabilidad

Un servidor está formado por los componentes mostrados en la tabla siguiente. La tabla también muestra el numero de componentes de cada tipo y el tiempo medio hasta fallo (MTTF) de cada componente.

Componente	Fuente alimentación	CPU	Ventilador CPU	Placa base	DIMMs	Disco duro
Nº	1	1	1	1	4	1
MTTF (horas)	100.000	1.000.000	100.000	200.000	1.000.000	100.000

El tiempo medio para reemplazar un componente que ha fallado (*mean time to repair*) es de 10 horas y la probabilidad de fallo sigue una distribución exponencial.

a) **Calcula** el tiempo medio hasta fallos del harddware (MTTF), el tiempo medio entre fallos (MTBF) y la disponibilidad del sistema.

En los càlculos anteriores se ha supuesto que solo puede haber fallos de hardware, sin embargo hay otros aspectos que pueden influir en la fiabilidad de un sistema, cómo la estabilidad del software o la propia red electrica. Sabemos que el sistema operativo usado en nuestro servidor tienen un tiempo medio entre fallos de 15000 horas y la inestabilidad de la alimentación electrica (microcortes, caidas de tensión, etc) provoca un fallo cada 745 horas en media. Los fallos provocados tanto por el sistema operativo como la red eléctrica siguen tambieén una distribución exponencial.

b) **Calcula** el tiempo medio hasta fallos del sistema (MTTF) teniendo en cuenta la combinación del hardware, el SO y la red eléctrica. ¿crees que valdria la pena gastar mucho más en un computador cuyos componentes hardware tienen el doble de MTTF (respecto la tabla anterior)? Justifica la respuesta.

Apartados para nota (no son fàciles)

Cuando se produce un fallo (sea causado por el hardware, el SO o por un fallo electrico), se pierden los datos de la aplicación que se estababa ejecutando en ese momento y hay que volverla a ejecutar una vez el computador vuelve a estar en funcionamiento. En nuestro servidor queremos ejecutar una aplicación que tarda 1000 horas en

ejecutarse, si durante su ejecución hay una fallo se deberà ejecutar de nuevo desde el principio. Por simplicidad supondremos que el tiempo de poner en marcha de nuevo el computador y la aplicación es negligible (**MTTR = 0**)

c) **Calcula** la probabilidad de que el computador acabe la ejecucón de la aplicación sin que se produzca ningún fallo. **Calcula** cuantas veces (en media) habrá que ejecutar la aplicación hasta que acabe completamente.

Sabemos que (en media) la aplicación se ha ejecutado durante MTTF horas cada vez que se produce un fallo y hay que volverla a ejecutar desde el principio.

Para evitar el problema que un fallo en el sistema puede causar en las aplicaciones con tiempos de ejecución muy largos, estas aplicaciones suelen programarse con un mecanismo de *checkpointing*, mediante el cual los datos de la aplicación se salvan periodicamente (por ejemplo cada hora). Si el sitema falla, la aplicación puede continuar a partir del último *checkpoint* con lo que el tiempo perdido debido a los fallos es practicamente despreciable.

d) **Calcula** el tiempo medio real que tardaremos en obtener los resultados de dicha aplicación (sin checkpoints). **Calcula** el speedup que obtendriamos si programamos la aplicacion con checkpoints

Problemas Tema 2

Problema 1. (1) Operadores lógicos

Suponed que `x` e `y`, variables de tamaño 1 byte, tienen los valores `0x66` y `0x93` respectivamente. Rellenad la siguiente tabla, indicando los valores resultantes de aplicar las siguientes expresiones en C:

<table>
<tr><th>Expresión</th><th>valor binario</th><th>valor hex</th><th>Expresión</th><th>valor binario</th><th>valor hex</th></tr>
<tr><td>x & y</td><td></td><td></td><td>x && y</td><td></td><td></td></tr>
<tr><td>x | y</td><td></td><td></td><td>x || y</td><td></td><td></td></tr>
<tr><td>~x | ~y</td><td></td><td></td><td>!x || !y</td><td></td><td></td></tr>
<tr><td>x & !y</td><td></td><td></td><td>x && ~y</td><td></td><td></td></tr>
</table>

Problema 2. (1) Desplazamientos

Rellenad la tabla que se muestra a continuación. El ejercicio consiste en aplicar desplazamientos lógicos y aritméticos sobre un conjunto de variables de tamaño byte.

x		x << 4		x >> 3 (lógico)		x >> 3 (aritmético)	
hex	**binario**	**hex**	**binario**	**hex**	**binario**	**hex**	**binario**
0xF0							
0x0F							
0xCC							
0x55							
0x80							
0x02							

Problema 3. (1) Endianismo

Dada la siguiente porción de memoria:

Dirección	0x200	0x201	0x202	0x203	0x204	0x205	0x206	0x207	0x208	0x209	0x20A	0x20B	0x20C
Contenido	0x34	0xA1	0xFF	0x00	0x8D	0x01	0xAE	0x00	0x00	0xE0	0x12	0xFF	0xF4

Rellenad la siguiente tabla:

Dirección	Tamaño	Contenido	
		little endian	**big endian**
0x200	byte		
0x20B	byte		
0x207	word		
0x20B	word		
0x201	longword		
0x202	longword		
0x209	longword		

Problema 4. (1) Texe, Ganancia, Num instrucciones dinámicas, Num instrucciones estáticas

Dado el siguiente fragmento de código en C:

```
for (i=10000;i!=0;i--)
   res=res+vector[i];
```

Y una posible traducción a ensamblador:

```
        movl vector, %ebx
        movl $10000, %ecx
        movl $0, %eax
bucle:  addl (%ebx,%ecx,4), %eax
        decl %ecx
        jnz bucle
        movl %eax, res
```

a) ¿Cuántas instrucciones estáticas tiene el código?
b) ¿Cuántas instrucciones dinámicas tiene el código?
c) ¿Cuántos accesos memoria se producen al ejecutar este código?
d) Suponiendo que las instrucciones tardan 3 ciclos si no acceden a la memoria de datos y 5 si acceden a la memoria de datos, ¿cuántos ciclos tarda en ejecutarse el programa anterior?
e) Si cambiamos la memoria del procesador de forma que los accesos a las instrucciones fuesen más rápidos y se ejecutaran 0,1 instrucciones más por ciclo ¿cuál sería la ganancia para este programa?

Problema 5. (2) Traducción

Dada la siguiente definición de variables globales:

```
char A[256];
char tabla[256];
```

Traducid a ensamblador de IA32 el siguiente fragmento de código:

```
for (i=0; i<256; i++)
  A[i] = tabla[A[i]];
```

Problema 6. (2) Traducción

Dado el siguiente código escrito en C:

```
int *sorpresa (int i,  int *x)
{
        if (i>-10 && i<10)
           *x = i;
        else
           x = &i;
        return x;
}
```

Teniendo en cuenta que los parámetros de la función son accesibles en las siguientes direcciones: `i` en 8`(%ebp)`, dirección de `x` en `12(%ebp)`, traducid a ensamblador del IA32 el cuerpo de la subrutina.

Problema 7. (2) Traducción

Dado el siguiente código escrito en C:

```
int PROD(int i, int j, int A[100][150], int B[150][200])
{
    int k;
    int sum = 0;
    for (k=0; k<150; k++)
        sum = sum + A[i][k] * B[k][j];
    return sum;
}
```

Teniendo en cuenta que los parámetros de la función son accesibles en las siguientes direcciones: `i` en `8(%ebp)`, `j` en `12(%ebp)`, que la dirección de inicio de `A` está en `16(%ebp)` y la dirección de inicio de `B` en `20(%ebp)`, traducid a ensamblador del IA32 el cuerpo de la subrutina. Usad registros para las variables k y sum.

Problema 8. (2) Traducción

Dado el siguiente código escrito en C:

```
int Examen(int M[50][75], int n)
{   int x, y, elem, media;
    media = 0;
    elem = 0;
    for (x=0; x<50; x++)
        for (y=0; y<75; y++)
            if (M[x][y] > n) {
                media = media + M[x][y];
                elem++;
            }
    if (elem > 0)
        media = media / elem;
    return media;
}
```

Teniendo en cuenta que los parámetros de la función son accesibles en las siguientes direcciones: `dirección de M` en `8(%ebp)` y `n` en `12(%ebp)`, traducid a ensamblador del IA32 el cuerpo de la subrutina. Usad registros para las variables **`x, y, elem, media`**.

Problema 9. (1) Structs

Dada la siguiente declaración de variables:

```
typedef struct{
        char a;
        int b[10];
} elem;
elem s[100];
```

Sabemos que la dirección de s se encuentra en %ebx y las variables i, j y x se encuentran respectivamente en los registros %esi,%edi y %dl

a) Dibujad el struct elem.

b) Determinad cómo se calcula la dirección de memoria donde se almacena el dato

```
s[i].b[j]
```

c) Escribid la secuencia de instrucciones necesaria para codificar la siguiente sentencia

```
x = s[s[i].b[j]].a
```

Problema 10. (1) Subrutinas

Dada la siguiente función escrita en C:

```
int calcula(int M[10][10], int m, int n)
{  int i, suma, fila;
   suma = 0;
   fila = 0;
   for (i = m; i<n; i++)
      suma = suma + Normaliza(M[fila][i], &fila);
   return (suma + 1);
}
```

a) Dibujad el bloque de activación de la función.

b) Traducid la función a ensamblador del IA32. Suponed que la función **`Normaliza`** ya está programada.

Problema 11. (2) Subrutinas

Dada la siguiente función escrita en C:

```
int maximo (int v[300], int i)
{  int res;
   if (i == 0) res = v[0];
   else {
      res = maximo(v, i-1);
      if (v[i] > res)
         res = v[i];
   }
   return res;
}
```

a) Dibujad el bloque de activación de la función.

b) Traducid la función a ensamblador del IA32.

Problema 12. (2) Subrurinas

Dada la siguiente función escrita en C:

```
int MULT(int a, int b)
{  int aux;
   if (b == 0)
      return 0;
   else {
      aux = MULT(a, b>>1) << 1;
      if (b & 0x0001 != 0)
         return (aux + a);
      else
         return aux;
   }
}
```

a) Dibujad el bloque de activación de la función.

b) Traducid la función a ensamblador del IA32.

Problema 13. (1) Subrutinas

Dada la siguiente función recursiva escrita en C:

```
int f (int *a, int b, int c)
{   int y[10];
    ...
    y[2] = f(&y[1], y[3], y[4]); /* 1 */
    ...
}
```

a) Dibujad el bloque de activación de la función f;

b) Traducid la sentencia {1} a ensamblador.

Problema 14.

Dada la siguiente función escrita en C:

```
void examen (int a, int b[100], int *c)
{   int d[100];
    int aux;
    ...
}
```

a) Dibujad el bloque de activación de la función.

Traducid a ensamblador del IA32 las siguientes sentencias suponiendo que se encuentran en el cuerpo de la rutina **examen**:

b) Traducid a ensamblador IA32 la siguiente sentencia suponiendo que se encuentra en el cuerpo de la rutina

```
examen(0, d, &aux);
```

c) Traducid a ensamblador IA32 la siguiente sentencia suponiendo que se encuentra en el cuerpo de la rutina

```
for (aux = 0; aux < 100; aux++)
            b[aux] = d[aux];
```

d) Traducid a ensamblador IA32 la siguiente sentencia suponiendo que se encuentra en el cuerpo de la rutina

```
examen(a, b, c);
```

Problema 15.

Dado el siguiente código escrito en C:

```
void B (int i, int v[100], int *j);
void A (int i, int v[100], int *k)
{
    int vloc[100];
    B(i, v, &i);
    if (i == 1)
        v[*k] = 400;
    else {
        vloc[*k] = 400;
        A(i-1, vloc, k)
    }
}
```

a) Dibujad el bloque de activación de la función A.

b) Codificad en lenguaje ensamblador del IA32 la función A.

Problema 16.

Dado la siguiente función recursiva escrita en C:

```
int A(int i, int *k)
{  int b;
   if (i <= 0)
      return 0;
   else {
      b = *k - 1;
      return (A(A(i-2, &b), &i) + 1);
   }
}
```

a) Dibujad el bloque de activación de la función A.
b) Traducid la función A a ensamblador del IA32:

Problema 17.

Dado el siguiente código en C:

```
int F1(int v[10], int j, int *k);

void examen(int m[10][10], int v[10], int n, int *j) {
  int vl[10];
  int i, aux;

  for (i=0; i<n; i++)
    v[i] = F1(vl, *j, j);

  aux = *j;
  for (i=0; i<n; i++)
    if (v[i] != 0)
      m[n][i] = aux / v[i];
    else
      m[n][i] = aux / 4;
}
```

Se pide:

a) Dibujad el bloque de activación de la rutina `examen`, indicando claramente a qué distancia del registro EBP se encuentran los parámetros y variables locales.
b) Traducid a ensamblador del IA32 la rutina `examen`.

Problema 18.

Considerad el siguiente código escrito en C, suponiendo que las constantes N y M han sido declaradas previamente en un `#define`.

```
int mat 1[M][N];
int mat2[N][M]

int SumaElemento(int i, int j)
{
   return mat1[i][j] + mat2[i][j];
}
```

Para esta subrutina el compilador genera el siguiente código en ensamblador:

```
SumaElemento:
      pushl %ebp
      movl %esp, %ebp

      movl 8(%ebp), %eax
      movl 12(%ebp), %ecx
      sall $2, %ecx
      leal (,%eax,8), %edx
      subl %eax, %edx
      leal (%eax, %eax, 4), %eax
      movl mat2(%ecx, %eax, 4), %eax
      addl mat1(%ecx, %edx, 4), %eax

      movl %ebp, %esp
      popl %ebp
      ret
```

a) ¿Cuánto valen las constantes `M` y `N`?
b) ¿Cuántas instrucciones estáticas tiene el código?
c) ¿Cuántas instrucciones dinámicas tiene el código?
d) ¿Cuántos accesos memoria se producen al ejecutar este código?
e) Suponiendo que cada ciclo se ejecutan 0,8 instrucciones si éstas no acceden a la memoria de datos y 0,5 si acceden a la memoria de datos, ¿cuántos ciclos tarda en ejecutarse el programa anterior?
f) Si cambiamos la memoria del procesador de forma que los accesos a las instrucciones fuesen más rápidos y se ejecutaran 0,1 instrucciones más por ciclo ¿cuál sería la ganancia para este programa?

Problema 19.

Dado el siguiente código escrito en C:

```
typedef struct {
  int i1;
  char c2[30];
  int i3;
} sx;

typedef struct {
  sx tabla[100];
  int n;
} s2;

inf F(sx *p2, int y);

int examen(s2 *p1, int *x, int y)
{ int i, j;
  sx aux;
  . . .
}
```

a) Dibujad como quedarian almacenadas en memoria las estructuras sx y s2, indicando claramente los deplazamientos respecto el inicio y el tamaño de todos los campos.
b) Dibujad el bloque de activación de la función examen, indicando claramente los desplazamientos relativos al EBP necesarios para acceder a los parámetros y las variables locales.
c) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
return(*x+aux.i3);
```

d) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
aux.i1 = F(&(*p1).tabla[j], y);
```

e) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
i = j * y;
```

f) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
aux.c2[i] = aux.c2[23];
```

g) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
for (i=0; (i<y) && (i<(*p1).n) ; i=i+5)
   (*p1).tabla[i].i1 = (*p1).tabla[i].i3 + i;
```

h) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
if (aux.i1 != y)
   aux.i3 = i;
else
   aux.i3 = j;
```

i) Traducid la siguiente sentencia a ensamblador del x86, suponiendo que está dentro de la función examen:

```
i = 0;
while (aux.c2[i] != '.') {
   aux.c2[i] = '#';
   i++;
}
```

Problemas Tema 3

Problema 1. Repaso Cache

Disponemos de un procesador de 16 bits (con bus de direcciones de 24 bits) con una memoria cache con las siguientes características:

- 2-asociativa
- Algoritmo de reemplazo LRU
- Tamaño total = 1024 bytes
- Tamaño de bloque = 16 bytes
- Política de escritura: write through + write no allocate

Rellenad la siguiente tabla:

Tipo	@ en hex	Bloque de memoria	Conjunto de MC	Acierto / Fallo	Lectura de MP			Escritura en MP		
					si / no	@	tamaño	si / no	@	tamaño
R byte	ECA930									
R word	ECB931									
W byte	ECA972									
W word	ECA933									
W byte	ECC934									
R word	ECC935									
W byte	ECB976									
W word	ECB937									
R byte	ECB938									
R word	ECA979									

Notas:

- R byte (lectura de 1 byte), R word (lectura de 2 bytes), W byte (escritura de 1 byte), W word (escritura de 2 bytes).
- El tamaño de las lecturas (y escrituras) se ha de especificar en bytes.

Problema 2. Repaso Cache

Disponemos de un procesador de 16 bits con un bus de direcciones de 16 bits. Este procesador tiene una memoria cache de datos con las siguientes características:

- Tamaño de bloque = 16 bytes
- Asociatividad = 2 (reemplazo = LRU)
- Número de líneas = 8
- Política de escritura: write through + write no allocate

Suponiendo que el contenido de la cache es el siguiente:

conjunto 0		conjunto 1		conjunto 2		conjunto 3	
EC8	1	EC5	1	EC6	0	EC7	1
AB4	0	libre	0	AB2	1	libre	0

Teniendo en cuenta que:

- En el contenido de la MC para simplificar hemos dejado el número de bloque de memoria en vez del tag.

- El bit a 1, en el contenido de la cache, indica que es la línea más recientemente referenciada.
- R-byte (lectura de 1 byte), R-word (lectura de 2 bytes), W-byte (escritura de 1 byte), W-word (escritura de 2 bytes).
- El tamaño de las lecturas (y escrituras) se ha de indicar en bytes.

Rellenad la siguiente tabla:

Tipo	@ en hex	Bloqu de memoria	Conjunto de MC	Acierto / Fallo	Lectura de MP			Escritura en MP		
					si / no	@	tamaño	si / no	@	tamaño
R byte	8890									
W word	EC51									
W byte	EC62									
W word	23D3									
W byte	ABA4									
R word	ABA5									
R byte	23D6									
W word	EC57									
R byte	EC68									
R word	8899									

Indicad también cómo queda la cache después de realizar los 10 accesos a memoria:

conjunto 0		conjunto 1		conjunto 2		conjunto 3	

Problema 3. Repaso Cache

Disponemos de un procesador de 16 bits con direcciones de 20 bits que tiene una memoria cache de datos con las siguientes características:

- 3-asociativa, con algoritmo de reemplazo LRU
- 256 bytes por bloque
- 12 bloques en la cache
- política de escritura: copy back + write allocate

El contenido inicial de la cache (por simplicidad hemos utilizado el número de bloque de memoria, en vez del tag) es el siguiente:

conjunto 0	DB	conjunto 1	DB	conjunto 2	DB	conjunto 3	DB
984	0	441	0	666	0	0A7	1
98C	1	A19	0	40A	1	45B	0
000	1	-	-	002	0	F2F	0

El DB=1 indica que el bloque correspondiente ha sido modificado. La información de reemplazo está implícita en la posición. Las posiciones inferiores corresponden a los bloques que llevan más tiempo sin utilizarse. Las posiciones superiores corresponden a los últimos bloques utilizados. Por ejemplo, en el conjunto 3, el bloque 0A7 es el último utilizado, y el bloque F2F el que lleva más tiempo sin ser utilizado.

Rellenad la siguiente tabla, indicando para cada referencia, el número de bloque de memoria que le corresponde, a qué conjunto de MC va a parar, si es acierto o fallo, si hay lectura de MP, si hay escritura en MP y el bloque reemplazado cuando proceda.

dirección (hex)	bloque memoria	conjunto MC	¿acierto o fallo?	lectura MP ¿si/no?	Escritura MP ¿si/no?	¿bloque reemplazado?
lect 00000						
lect 88801						
escr 02E02						
escr 98403						
escr 00004						
lect 40A05						
lect 99606						
lect 32C07						
lect 98408						
escr 72509						

¿Cuál es el contenido final de la Memoria Cache?

conjunto 0	DB	conjunto 1	DB	conjunto 2	DB	conjunto 3	DB

Problema 4. Repaso Cache

Se quiere diseñar la memoria cache para un determinado procesador. Se barajan dos alternativas:

- (1) Con escritura inmediata (write through) y sin carga en caso de fallo de escritura.
- (2) Con escritura cuando reemplazo (copy back) y carga en caso de fallo de escritura

Se han obtenido por simulación las siguientes medidas:

- porcentaje de escrituras: 20%
- porcentaje de bloques modificados: 33.33%
- tasa de aciertos caso (1): 0.9
- tasa de aciertos caso (2): 0.85

El tiempo de acceso a memoria cache es de 10 ns y el tiempo de memoria principal para escribir una palabra es de 80 ns. Para leer o escribir un bloque en la memoria principal se emplean 100 ns.

Se pide:

a) **Calculad** el tiempo invertido en ejecutar 1000 accesos para las dos alternativas. Detallad el número de accesos de cada tipo y el tiempo empleado para cada uno de ellos.

b)

Indicad qué alternativa sería la más rápida para un programa que sólo realizara lecturas.

c) **Indicad** qué motivos pueden existir para que la escritura de una palabra tarde ligeramente menos que la escritura de un bloque.

Problema 5. Repaso Cache

Tenim una CPU amb les següents característiques:

- CPI ideal: 1.5 cicles/instrucció
- Temps de cicle (Tc): 10 ns
- Nombre de referències per instrucció (nr): 1.6 referències/instrucció
- Cache d'instruccions i dades separades
- Cache de dades amb **copy back** i **write allocate**.

Les característiques de les dues caches son les següents:

Característica	Instruccions	Dades
Numero de referències a memòria per instrucció (nr)	1 ref/inst	0.6 ref/inst
Taxa de fallades (m)	4 %	10 %
Penalització (Tpf) al reemplaçar un bloc no modificat	10 cicles	15 cicles
Penalització (Tpf) al reemplaçar un bloc modificat	---	20 cicles
Temps de servei en cas d'encert (Tsa)	1 cicles	1 cicles
Percentatge de blocs modificats (pm)	0 %	20 %

a) **Calculeu** el temps mig d'accés a memòria en cicles (TmaI) pels accessos a instruccions?
b) **Calculeu** el temps mig d'accés a memòria en cicles (TmaD) pels accessos a dades?
c) **Calculeu** el temps mig d'accés a memòria en cicles (Tma) per tots els accessos?
d) **Calculeu** el temps d'execució en ns. (Texec) d'una instrucció?

Problema 6. Repaso Cache

L'empresa A.C.M.E. esta dissenyant un nou processador del que sabem que genera **1.3 referències** a memòria per instrucció de les que **0.3 son de dades**. Els seus dissenyadors s'han adonat de que al mateix xip queda espai suficient per posar-hi un poc de cache. Després de fer alguns càlculs resulta que poden posar-hi una cache de 16k o dues (una de 8k i una de 4k), i han optat per una de les dues configuracions següents:

- Una sola cache **unificada de 16Kb**. En aquest cas (si la cache fos ideal) tenim un **CPI_{ideal} de 1.5** cicles/instrucció degut a que quan accedim a la cache per buscar una dada no podem buscar una instrucció al mateix temps.
- Dues caches separades, una d'**instruccions de 4Kb** i una de **dades de 8Kb.** En aquest cas el **CPI_{ideal} es de 1.2** cicles/instrucció ja que podem buscar dades i instruccions simultaniament.

En qualsevol de les dues configuracions:

- Tc (temps de cicle): 10ns
- Tsa (temps de servei en cas d'encert): 1cicle.
- Tsf (temps de servei en cas de fallada): 10 cicles.

La taxa de fallades, per diferents configuracions de cache, es mostra a la següent taula:

Mida	Instruccions	Dades	Unificada
4K	8.6%	8.7%	11.2%
8K	5.8%	6.8%	8.3%
16K	3.6%	5.3%	5.9%

Es demana:

a) Quin serà el temps mig d'accés **Tma** per cada configuració (en cicles)?
b) Quin serà el temps d'execució **Texec** de 1 instrucció real en cada cas?
c) Per quina opció optaríeu i perquè? (en una línia)
d) Creus que es pot trobar alguna opció millor en base a les **dades de que disposem**? En cas afirmatiu, digues quina i perquè.

Problema 7. Repaso Memoria Virtual

Dado el siguiente código escrito en ensamblador x86:

```
          movl $0, %ebx
          movl $0, %esi
    for:  cmpl $512*1000, %esi
          jge end
    (a)   movl (%ebx, %esi, 4), %eax
    (b)   addl %eax, 8*1024(%ebx, %esi, 4)
    (c)   movl %eax, 16*1024(%ebx, %esi, 4)
          addl $512, %esi
          jmp for
    end:
```

Suponiendo que la memoria utiliza **páginas de tamaño 8KB** y que utilizamos un **TLB de 4 entradas (reemplazo LRU)**, responde a las siguientes preguntas:

a) Para cada uno de los accesos (etiquetas a, b, c), indica a qué página de la memoria virtual se accede en cada una de las 17 primeras iteraciones.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a																	
b																	
c																	

b) **Calcula** la cantidad de **aciertos de TLB**, en todo el bucle: . . . .
c) **Calcula** la cantidad de **fallos de TLB**, en todo el bucle: . . . .

Suponiendo que la memoria utiliza **páginas de tamaño 4KB** y que utilizamos un **TLB de 4 entradas (reemplazo LRU)**, responde a las siguientes preguntas:

d) Para cada uno de los accesos (etiquetas a, b, c), indica a qué página de la memoria virtual se accede en cada una de las 17 primeras iteraciones.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
a																	
b																	
c																	

e) **Calcula** la cantidad de **aciertos de TLB**, en todo el bucle: . . . .
f) **Calcula** la cantidad de **fallos de TLB**, en todo el bucle: . . . .

Problema 8. Repaso Memoria Virtual

Tenim un processador amb memòria virtual basada en paginació. El sistema de memòria virtual te les següents característiques:

- 16 bits d'adreça lògica
- 15 bits d'adreça física
- mida de pàgina 8 KB
- reemplaçament LRU

El contingut de la taula de pàgines es mostra a la figura 1, on VPN = número de pàgina lògica, P=bit de presencia, M=pàgina modificada i PPN=número de pàgina física. El contingut de la memòria física es mostra a la figura 2. En cas que la memòria física s'empleni i faci falta reemplaçar una pàgina es segueix un algorisme LRU. De les tres pàgines inicialment presents a memòria, la pàgina física 1 (lògica 3) es la que fa menys temps que ha estat accedida, la següent es la 0 (lògica 2) y, finalment, la 2 (lògica 4) es la que fa més temps que ha estat accedida.

1 Contingut **inicial** de la **Taula de Pàgines**

VPN	P	M	PPN
0	0	0	-
1	0	0	-
2	1	0	0
3	1	0	1
4	1	0	2
5	0	0	-
6	0	0	-
7	0	0	-

2 Contingut **inicial** de **Memòria**

pàgina física	pàgina lògica
0	2
1	3
2	4
3	-

3 Contingut **final** de la **Taula de Pàgines**

VPN	P	M	PPN
0			
1			
2			
3			
4			
5			
6			
7			

4 Contingut **final** de **Memòria**

pàgina física	pàgina lògica
0	
1	
2	
3	

a) **Empleneu** la següent taula indicant, per cada referència, la pàgina lògica (VPN), el desplaçament, l'adreça física resultant de la traducció. Indiqueu amb una creu (X) quan es produeix un fallo de pàgina, quan es llegeix de disc dur, quan s'escriu a disc dur i, en cas de reemplaçar una pàgina, indiqueu el VPN i PPN. Indiqueu també el contingut final de la taula de pàgines i de la memòria física (figures 3 i 4)

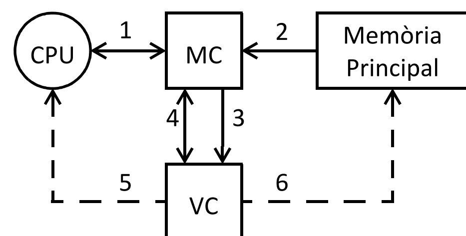
adreça lògica (hexa)		VPN (hexa)	desplaçament (hexa)	adreça física (hexa)	fallo de pàgina	lectura disc	escriptura disc	Pàgina reemplaçada	
								VPN	PPN
escriptura	F458								
escriptura	8666								
lectura	1BBF								
escriptura	5C44								
lectura	6600								
lectura	4000								

Problema 9. Caches petites y simples

Hem vist a teoria que una cache d'emplaçament directe te un temps d'accés inferior a una associativa. Una cache directa també sol tenir una taxa de fallades més elevada degut als conflictes. Una possible sol.lució al problema es tenir el que s'anomena una cache de víctimes (*Victim Cache*) que permet reduir les fallades degudes a conflictes. La idea es una aplicació del concepte de cache petita i simple.

Una cache de víctimes (VC) es una cache totalment associativa, però molt petita (4-8 blocs) que treballa en paral.lel a la memòria cache (MC), que habitualment es d'emplaçament directe. La VC emmagatzema aquells blocs que han estat expulsats de la MC. Encara que la VC es totalment associativa el seu temps d'accés es similar o inferior al de la MC (directa) perquè es molt petita.

La figura il·lustra el funcionament de un sistema de memòria amb cache de víctimes. Quan hi ha un accés a memòria es poden donar les següents situacions:



- *hit* a MC: es serveix la dada de MC (1) (no penalització).
- *miss* a MC (i també a VC): es porta el bloc corresponent de memòria principal (2) a MC, i el bloc expulsat de la MC s'emmagatzema a VC (3) (penalització molt elevada).
- *miss* a MC però *hit* a VC: donat que el bloc demanat es troba a VC no es necessari accedir a memòria principal sinó que l'obtenim directament de VC i el bloc expulsat de MC

l'emmagatzemem a VC, es a dir intercanviem el bloc demanat i el expulsat entre MC i VC (4) (penalització molt baixa).

Donat que la VC i la MC es poden consultar en paral.lel, una possible implementació podria permetre que en el darrer cas (*miss* a MC i *hit* a VC) la dada es servis a la CPU directament per la VC (5) amb el que es podria aconseguir que no hi hagués cap penalització, tot i que la circuiteria necessària per poder-ho fer (multiplexors, comprovació en paral.lel de les dos caches, ...) podria incrementar lleugerament el temps d'accés. Una altra consideració a tenir en compte es el dels blocs modificats en una cache *copy back*. Donat que quan un bloc es expulsat de MC no desapareix del sistema, sinó que es copia a VC, només serà necessari actualitzar la memòria principal quan un bloc modificat sigui expulsat definitivament de VC (6).

Aquest problema ens permetrà veure el funcionament de les caches de víctimes i els seus avantatges. Per construir una memòria cache s'han considerat tres possibilitats:

- Una memòria cache d'emplaçament directe amb 8 blocs.
- Una memòria cache associativa per conjunts amb 4 conjunts de 2 blocs cadascun i amb reemplaçament LRU.
- Una memòria cache d'emplaçament directe amb 8 blocs a la que s'ha afegit una *victim cache* amb reemplaçament FIFO de 2 blocs de capacitat.

a) **Indiqueu** quins accessos seran *hit* (amb una X) per cada una de les tres possibilitats per la següent seqüència de **referències a bloc** (en octal) on tots els accessos son lectures. En el cas de la directa + VC es considerarà *miss* si el bloc referenciat no es troba ni a MC ni a VC.

Bloc de memôria	**73**	**55**	**43**	**45**	**73**	**45**	**13**	**43**	**73**	**55**	**45**	**73**	**15**	**43**
Directa														
2-associativa														
Directa + VC														

b) Creus que hi hauria cap diferencia si la VC fes servir un reemplaçament LRU? Perquè?

Volem estudiar la implementació d'aquestes 3 caches com a cache de dades en un processador. Per un programa P que només fa lectures executat en aquest processador amb memòria ideal (on tots els accessos a memòria tarden 1 cicle) sabem que ha executat $10x10^9$ instruccions en $12x10^9$ cicles i s'han fet $3x10^9$ accessos a memòria (dades).

c) **Calculeu** el CPI amb memòria ideal (CPI_{ideal}).

d) **Calculeu** el ratio *nr* (accessos a memòria per instrucció).

La cache es troba al camí crític del processador i volem que en cas d'encert es pugui llegir la dada en 1 sol cicle, de forma que el temps d'accés a la cache ens determinarà el temps de cicle del processador en totes les implementacions.

Cache d'**emplaçament directe**: El temps de cicle (Tc) es de 10 ns/cicle, la taxa de fallades (m) es de 0.1 fallades/accés i el temps de penalització en cas de fallada (Tpf) es de 10 cicles.

e) Quants cicles tarda en executar-se el programa P?

f) Quin es el temps d'execució de P? (en segons)

Cache **2-associativa**: El temps de cicle (Tc) es de 12 ns/cicle, la taxa de fallades (m) es de 0.05 fallades/accés i el temps de penalització en cas de fallada (Tpf) es de 9 cicles.

g) Perquè creus que el temps de penalització en cas de fallada es de 9 cicles mentre que en el cas d'emplaçament directe era de 10 cicles si sabem que la memòria principal es la mateixa?

h) Quants cicles tarda en executar-se el programa P?

i) Quin es el temps d'execució de P? (en segons)

Cache **emplaçament directe + victim cache** amb accés simultani: A pesar que la victim cache es més ràpida que la d'emplaçament directe, la lògica i el multiplexor necessaris per controlar de quina cache obtindrem la dada fa que el temps d'accés sigui lleugerament més alt que en el cas de la cache directa. El temps de cicle (Tc) es de 11 ns/cicle, la taxa de fallades (m) global del conjunt MC+VC es de 0.06 fallades/accés i el temps de penalització en cas de fallada (Tpf) es de 10 cicles.

j) Quants cicles tarda en executar-se el programa P?

k) Quin es el temps d'execució de P? (en segons)

Cache **emplaçament directe + victim cache** amb accés seqüencial: En aquesta segona implementació, els accessos que s'han de fer a la victim cache tenen una penalització addicional de un cicle, però el temps de cicle es el de la cache d'emplaçament directe. El temps de cicle (Tc) es de 10 ns/cicle, la taxa de fallades (m) global del conjunt MC+VC es de 0.06 fallades/accés, el temps de penalització en cas que fallem a MC però encertem a VC (Tpvc) es de 1 cicle i el temps de penalització en cas que fallem a totes dues (Tpf) es de 11 cicles.

l) Perquè creus que el temps de penalització en cas de fallada es de 11 cicles mentre que en el cas d'emplaçament directe era de 10 cicles si sabem que la memòria principal es la mateixa?
m) Calcular la probabilitat que un accés falli a MC però encerti a VC? (pista: es pot deduir a partir de la taxa de fallades global i la taxa de fallades que tenim quant només hi ha la cache d'emplaçament directe)
n) Quants cicles tarda en executar-se el programa P?
o) Quin es el temps d'execució de P? (en segons)

Problema 10. Predicción de vía

Una CPU funciona a un voltaje de 1,2 V y una frecuencia de 2GHz. Se ha determinado que esta CPU tiene una corriente de fugas de 3 A y que a pleno rendimiento tiene una carga capacitiva equivalente de 5 nF (nanoFaradios).

a) **Calculad** la potencia media dinámica (debida a conmutación), la potencia media estática (debida a fugas) y la potencia media total.

Se desea integrar esta CPU con una memoria cache de datos 2-asociativa de 128 KB de capacidad y un tamaño de bloque de cache de 64 bytes. Las direcciones generadas por la CPU son de 48 bits. La corriente de fugas de la memoria RAM estática es de 3 μA (microAmperios) por bit. La energía consumida durante un acceso a la memoria de etiquetas es de 5 nJ (nanoJoules) por vía y la consumida durante un acceso a la memoria de datos es de 25 nJ por vía.

b) **Calculad** el numero de conjuntos, el de bloques de cache, el de vías y el de bloques por vía.
c) **Dibujad** una dirección indicando claramente los campos usados para seleccionar el byte dentro del bloque, seleccionar el conjunto de la cache y los bits usados como etiqueta.
d) **Calculad** el tamaño **en bits** de la memoria de datos y el de la memoria de etiquetas de una vía (por simplicidad ignoraremos el bit de validez y otros bits de control).
e) **Calculad** la potencia media estática (debida a fugas) de la cache.

Se desean comparar diversas implementaciones alternativas de cache de datos 2-asociativa: **paralela**, **serie**, y con **predictor de vía**. Para compararlas se usa un *benchmark* con $4x10^9$ instrucciones dinámicas que realiza 10^9 accesos de datos a memoria y $2x10^9$ operaciones aritméticas de punto flotante. Este *benchmark* tiene un 10% de fallos en la cache descrita anteriormente.

En la implementación **paralela**, se accede simultáneamente tanto a las memorias de etiquetas como las de datos de ambas vías. Un acceso a cache se realiza en 1 ciclo y la penalización media por fallo de cache es de 20 ciclos. El *benchmark* ejecutado con la implementación **paralela** de la cache ha tardado 5 segundos.

f) **Calculad** los MFLOPS de la implementación **paralela.**
g) **Calculad** el CPI de la implementación **paralela** y el CPI que obtendríamos con una memoria ideal (CPI_{ideal}) en donde todos los accesos tardan 1 ciclo.
h) **Calculad** la energía dinámica consumida por un acceso a la cache. Para simplificar asumiremos que todos los accesos consumen lo mismo sean acierto o fallo, la energía extra consumida en acceder a memoria principal en caso de fallo está fuera de los objetivos de este problema.
i) **Calculad** la potencia (dinámica) media consumida en acceder a la cache
j) **Calculad** la potencia media total (estática+dinámica) consumida por el sistema CPU-cache.
k) **Calculad** la energía total consumida para ejecutar el *benchmark* y la eficiencia en MFLOPS/Watt.

En la implementación **serie** un acceso tarda 2 ciclos. En el primer ciclo se accede a las memorias de etiquetas de ambas vías. Una vez determinada la vía que contiene el dato, en el segundo ciclo se accede solo a la memoria de datos de dicha vía. La penalización en caso de fallo sigue siendo de 20 ciclos ya que en el primer ciclo del acceso ya se puede determinar si es acierto o fallo. Obsérvese que respecto la implementación **paralela**, los aciertos tienen una penalización de 1 ciclo.

l) **Calculad** el tiempo de ejecución y los MFLOPS de la implementación **serie.**
m) **Calculad** la energía consumida por un acceso a la cache.

n) **Calculad** la potencia (dinámica) media consumida en acceder a la cache
o) **Calculad** la potencia media total consumida por el sistema CPU-cache.
p) **Calculad** la energía total consumida para ejecutar el *benchmark* y la eficiencia en MFLOPS/Watt.

En la implementación con **predictor de vía**, este predice la vía probable en que se encuentra el dato y se accede solo a las memorias de etiquetas y de datos de esa vía. En caso de fallo del predictor hay que acceder a la otra vía (memorias de etiquetas y datos). El predictor usado consiste en una memoria de 8k x 1 bits indexado con los bits bajos del PC de las instrucciones de acceso a memoria. Este predictor tiene una tasa de aciertos del 80% del total de accesos a memoria y cada acceso al predictor consume 1nJ. En esta implementación se pueden dar las siguientes situaciones:

- El predictor acierta: se accede a 1 sola vía (datos+etiquetas) en 1 ciclo (no hay penalización) y al comprobar los tags se comprueba que es acierto de cache.
- El predictor falla pero es acierto de cache: El acceso tarda 2 ciclos, en el primero se accede a la vía probable (aunque equivocada) al comprobar los tags se descubre que es fallo (de via) y en el segundo se accede a la vía correcta y se descubre que es acierto de cache (1 ciclo de penalización).
- El predictor falla y además es fallo de cache: En el primer ciclo se accede a la vía probable (aunque incorrecta), en el segundo ciclo se accede a la otra vía y se descubre que es fallo de cache (con lo que hay que acceder a memoria principal). Obsérvese que en este caso la penalización es de 21 ciclos ya que no se descubre el fallo de cache hasta que se accede a la 2a vía.

q) ¿Puede darse al caso de que un acierto del predictor de vía sea fallo de cache? ¿porqué?
r) **Calculad** la potencia media estática (debida a fugas) del predictor y compárala con la de la cache (se calcula de la misma forma ya que se ha empleado el mismo tipo de memoria estática).
s) **Calculad** el tiempo de ejecución y los MFLOPS de la implementación con **predictor de vía.**
t) **Calculad** la energía consumida por un acceso en que el predictor acierta y uno en que el predictor falla (tener en cuenta la energía consumida por el acceso al predictor). Calcular también la energia media consumida por acceso.
u) **Calculad** la potencia (dinámica) media consumida en acceder a la cache
v) **Calculad** la potencia media total consumida por el sistema CPU-cache (acuérdate de las fugas del predictor).
w) **Calculad** la energía total consumida para ejecutar el *benchmark* y la eficiencia en MFLOPS/Watt.
x) **Calculad** la ganancia en eficiencia energética de la implementación serie sobre la paralela y la de predicción de vía sobre la serie.

Problema 11. Caches segmentadas

En un procesador X el camino crítico, y por tanto el tiempo de ciclo, está limitado por la memoria cache de datos. El tiempo de los componentes de la memoria cache de datos se desglosa de la siguiente forma:

Componente	Tiempo
Memoria de etiquetas	0,30 ns
Selección de vía	0,15 ns
Memoria de datos	0,45 ns
Mux/Driver de datos	0,10 ns
Registro de desacoplo (en caso que se use)	0,05 ns

Queremos evaluar el rendimiento de 4 posibles implementaciones de la memoria cache para el procesador X:

- Procesador **X1**: La cache de datos tiene una implementación paralela y el tiempo de acceso a la cache es de 1 ciclo de procesador
- Procesador **X2**: La cache de datos está segmentada en 2 etapas y el tiempo de acceso a la cache es de 2 ciclos de procesador
- Procesador **X3**: La cache de datos está segmentada en 3 etapas y el tiempo de acceso a la cache es de 3 ciclos de procesador

- Procesador **X4**: La cache de datos está segmentada en 4 etapas y el tiempo de acceso a la cache es de 4 ciclos de procesador

a) **Calculad** el tiempo de ciclo de la cache de datos y el tiempo total de un acceso para los procesadores X1, X2, X3 y X4, usando la distribución mas adecuada de los componentes por etapas.

b) **Razonad** porque descartamos las opciones X2 y X4 para el resto del problema

c) **Calculad** la frecuencia de reloj de los procesadores X1 y X3

Un programa P que ejecuta $2x10^9$ instrucciones tiene un 60% de instrucciones aritméticas, un 20% de instrucciones de salto y un 20% de instrucciones de acceso a memoria (Load/Store). Las instrucciones aritméticas tardan 5 ciclos, las de salto 4 y las de memoria 4 ciclos + los ciclos del acceso a la cache.

d) **Calculad** el CPI del programa P para los procesadores X1 y X3 suponiendo que nunca hay fallos en la cache de datos.

e) **Calculad** el speedup de X3 sobre X1 en % suponiendo que nunca hay fallos en la cache de datos

Sabemos que el programa P tiene un 10% de fallos en la cache de datos utilizada y que el tiempo de penalización en ambos casos es de 60 ciclos.

f) **Calculad** el speedup real de X3 sobre X1 en % teniendo en cuenta la jerarquía de memoria completa.

Problema 12. Cache no bloqueante.

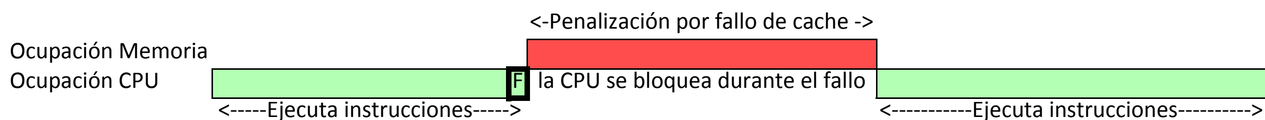
Nota: Conviene desenpolvar los apuntes de Probabilidad y Estadistica y repasar las distribuciónes de probabilidad geometrica y uniforme discreta.

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, por lo que el número de ciclos entre fallos sigue una ***distribución geométrica***.

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 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).

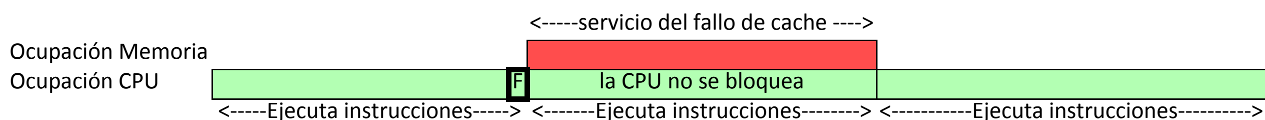


El programa P se ha ejecutado en el procesador con cache bloqueante (que llamaremos procesador B) en 4 segundos. Este procesador B funciona a una frecuencia de 2 GHz.

c) **Calculad** el CPI de P en el procesador B (CPI_B)

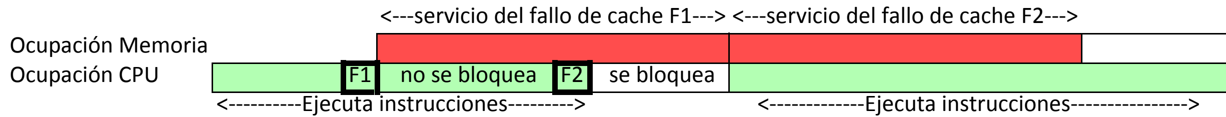
d) **Calculad** el tiempo de penalización por fallo de cache (Tpf) en ciclos.

El rendimiento del procesador se puede mejorar implementando una cache no bloqueante (con las mismas características de tamaño de bloque, asociatividad, reemplazo, etc) de forma que cuando se produce un fallo de cache el procesador siga ejecutando instrucciones tal como muestra la siguiente figura.



En la implementación elegida (que denominamos procesador N) dispondremos de un único MSHR (Miss Holding Status Register) que nos permite tener como máximo un fallo pendiente. Si durante el servicio de este fallo (F1) se

produce un segundo fallo (F2), hay que esperar a que la jerarquía de memoria complete el fallo en servicio antes de que pueda servir el siguiente fallo, con lo que el procesador se bloqueará por unos ciclos, tal como muestra la siguiente figura. Este mecanismo en que se permite un único fallo pendiente se denomina "hit under miss".



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 entre F1 y F2 será el mismo.

e) **Calculad** la probabilidad de que se produzca un segundo fallo durante el servicio de un fallo anterior
f) ¿Puede producirse un tercer fallo?

Si se produce un segundo fallo durante el intervalo de servicio de un fallo anterior, este se puede producir en cualquiera de los ciclos que dura el servicio, con la misma probabilidad. Es decir, se trata de una distribución de probabilidad ***uniforme discreta*** (dado de 60 caras).

g) **Calculad** cuantos ciclos se pierden como máximo y como mínimo en función de en que ciclo del intervalo se produce el segundo fallo.
h) **Calculad** el número medio de ciclos perdidos debido al segundo fallo (repasa cual es el valor medio esperado en una distribución de probabilidad uniforme discreta, o sea un dado numerado de 0 a 59)
i) **Calculad** el numero de ciclos necesario para ejecutar P en el procesador N (con cache no bloqueante)

Debido a la complejidad añadida de la cache no bloqueante, el procesador N funciona a un frecuencia ligeramente inferior de 1,9 GHz

j) **Calculad** la ganancia (speedup) del procesador N sobre el B

Problema 13. Continuación anticipada, Transferencia en desorden

Un procesador tiene una cache de primer nivel (que llamaremos L1) con bloques de 32 bytes y está conectado a un segundo nivel de jerarquía de memoria (que llamaremos L2) mediante un bus de 8 bytes de ancho. El primer nivel (L1) tiene un tiempo de acceso de 1 ciclo cuando se produce un acierto. Cuando fallamos en el primer nivel accedemos al segundo nivel (L2) para leer un bloque de datos. Suponemos que solo se realizan lecturas y que nunca hay fallos en L2. El siguiente cronograma ilustra un fallo en L1: se necesitan 5 ciclos de latencia y 4 para transferir los datos (T0-T3). Los datos se cargan en L1 mientras se transfieren (car L1), y una vez tenemos todo el bloque en L1, hay que realizar una lectura en L1 (Lect) para enviar el dato a la CPU (DATO), cosa que ocurre dentro del mismo ciclo.

CLK																
Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU											DATO					
L1	MISS						car L1	car L1	car L1	car L1	Lect					
L2		Latencia					T0	T1	T2	T3						

Al ejecutar un programa en un simulador, hemos obtenido que, a una frecuencia de 2GHz el programa tardaría 2 segundos en el caso ideal de que no haya fallos de cache, que se han realizado 10^9 accesos a memoria y que el 20% de los accesos provocarían un fallo en L1.

a) **Calculad** el tiempo de ciclo y los ciclos que tarda el programa en el caso ideal.
b)
Calculad los ciclos de penalización de un fallo de L1 y el tiempo de ejecución del programa teniendo en cuenta la penalización debida a los fallos de L1.

Para mejorar el rendimiento, permitimos que el procesador pueda captar el dato en el mismo ciclo que se transfiere de L2 a L1 (continuación anticipada o *early restart*). Mediante simulación sabemos que cuando se produce un fallo en L1 en nuestro programa, en el 70% de los casos el dato deseado se corresponde al que se transfiere en el ciclo T0,

mientras que hay una probabilidad del 10% de que se transfiera en cada uno de los ciclos restantes (T1-T3). Sabemos además que nunca se produce un fallo mientras se acaba de transferir el resto del bloque.

c) **Completad** el siguiente cronograma en donde se ilustran las acciones a realizar en caso de fallo en L1, suponiendo que tenemos continuación anticipada y que el dato solicitado se corresponde al byte 12 del bloque.

CLK																
Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU																
L1	MISS															
L2																

d) **Calculad** el tiempo medio de penalización (en ciclos) de un fallo en L1 y el tiempo de ejecución del programa para el procesador con continuación anticipada.

Para mejorar más el rendimiento hemos modificado también L2 de forma que se transfiera el dato solicitado en el primer ciclo (T0) de la transferencia (transferencia en desorden). Sabemos además que nunca se produce un fallo mientras se acaba de transferir el resto del bloque.

e) **Completad** el siguiente cronograma en donde se ilustran las acciones a realizar en caso de fallo en L1, suponiendo que tenemos transferencia en desorden y que el dato solicitado se corresponde al byte 12 del bloque.

CLK																
Ciclo	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
CPU																
L1	MISS															
L2																

f) **Calculad** el tiempo medio de penalización (en ciclos) de un fallo en L1 y el tiempo de ejecución del programa para el procesador con continuación anticipada.

g)

Calculad la ganancia (speedup) del sistema con continuación anticipada y del sistema con transferencia en desorden respecto el sistema sin ninguna mejora.

Problema 14.
Prefetch

Disponemos de un procesador de 16 bits con direcciones de 16 bits que tiene una memoria cache de datos con las siguientes características: 3-asociativa, con algoritmo de reemplazo LRU, 12 bloques y 64 bytes por bloque, política de escritura: *copy back + write allocate*

El contenido inicial de la memoria de etiquetas (tags) es el siguiente:

conjunto 0	DB	conjunto 1	DB	conjunto 2	DB	conjunto 3	DB
13	1	13	1	13	0	13	0
43	1	43	1	43	0	43	0
AC	0	AC	0	AC	1	AC	1

El DB=1 indica que el bloque correspondiente ha sido modificado. La información de reemplazo está implícita en la posición. Las posiciones inferiores corresponden a los bloques que llevan más tiempo sin utilizarse. Las posiciones superiores corresponden a los últimos bloques utilizados. Por ejemplo, en el conjunto 3, el bloque con tag 13 es el último utilizado, y el bloque con tag AC el que lleva más tiempo sin ser utilizado.

a) **Rellenad** la siguiente tabla, indicando para cada referencia, el número de bloque de memoria que le corresponde, la etiqueta (TAG), a qué conjunto de MC va a parar, si es acierto o fallo (A/F), el bloque

reemplazado cuando proceda, el número de bytes leídos de MP (si se lee de MP) y el número de bytes escritos en MP (si se escribe en MP)

tipo	dirección (hex)	bloque de memoria (hex)	TAG (hex)	conjunto MC	¿acierto o fallo? (A/F)	bloque reemplazado	bytes escritura MP	bytes lectura MP
LECT	B12B							
LECT	B145							
LECT	B1AF							
LECT	B1C4							
ESCR	4387							
LECT	1108							
ESCR	1199							
LECT	11AA							

A la cache anterior le añadimos un ***buffer de prefetch*** de una entrada. En este *buffer* se hace prebúsqueda hardware del bloque **i+1** cuando se accede (tanto en acierto como en fallo) al bloque **i**, siempre que el **i+1** no esté ya en la cache o en el *buffer*. En este último caso, no se realiza *prefetch*.

b) **Rellenad** la siguiente tabla (mismas referencias que la anterior) indicando, para cada referencia, el número de bloque de memoria que le corresponde, la etiqueta (TAG), a qué conjunto de MC va a parar, si se produce acierto o fallo en la cache (A/F), el número de bytes leídos de MP (si se lee de MP), el número de bytes escritos en MP (si se escribe en MP), el bloque de MP que se encuentra en el *buffer* (si procede), si se produce acierto o fallo (A/F) en el *buffer* y el bloque que se prebusca de MP (si procede).

tipo	dirección (hex)	bloque de memoria (hex)	TAG (hex)	conjunto MC	Cache ¿acierto o fallo?	bytes escritura MP	bytes lectura MP	bloque actual buffer	Buffer ¿acierto o fallo?	bloque prefetch buffer
LECT	B12B									
LECT	B145									
LECT	B1AF									
LECT	B1C4									
ESCR	4387									
LECT	1108									
ESCR	1199									
LECT	11AA									

Problema 15. Prefetch

Tenim un processador de 8 bits amb adreces de 16 bits que té una memòria cache de 256 bytes amb emplaçament directe, 16 bytes per bloc de i política d'escriptura: *copy back + write allocate*.

Aquesta cache implementa *prefetch hardware* amb un *buffer de prefetch* d'un bloc de capacitat. El funcionament del *buffer de prefetch* és el següent: Per cada fallada de cache, a més a més del bloc que s'està accedint, es cerca també el següent bloc de memòria principal i es posa al *buffer*. En cas que un accés falli a la cache i encerti al *buffer*, es passa el bloc del *buffer* a la cache i es porta al *buffer* el bloc següent de memòria principal.

Suposant que la cache està inicialment buida, empleneu la següent taula indicant per cada referència, el bloc de memòria principal que s'està accedint (#bloc de memòria), el bloc de memòria cache on s'emplaça (#bloc MC), el

TAG (etiqueta) corresponent, si és encert (*hit*) o fallo (*miss*) a la cache, el bloc actual que hi ha emmagatzemat al *buffer de prefetch* (#bloc BP) i si hi ha *hit* o *miss* al *buffer de prefetch*.

	adreça (hex)	**#bloc de memòria (hex)**	**#bloc MC (hex)**	**TAG (hex)**	**hit o miss**	**#bloc BP (hex)**	**hit o miss BP**
0	escr 1540						
1	lect 1548						
2	lect 1648						
3	escr 1650						
4	lect 1658						
5	lect 1540						
6	lect 1658						
7	escr 1550						
8	lect 1560						

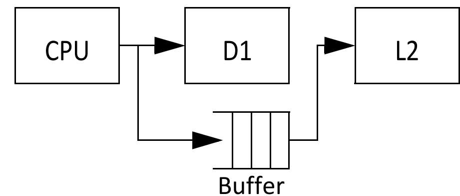
Problema 16. Buffers de escritura

En un procesador que interpreta el lenguaje máquina x86 ejecutamos el siguiente código:

```
        movl $0, %esi
A:      movl %eax, a(,%esi,4)  ; store a[i]
B:      movl %eax, b(,%esi,4)  ; store b[i]
i:      incl %esi              ; i++
c:      cmpl $N, %esi          ; i<N ?
j:      jl A
```

En este código, que inicializa 2 vectores a un valor, se han etiquetado las instrucciones del bucle. Las instrucciones A y B realizan *stores* en los vectores a y b cuyas direcciones de inicio son 0x10000000 y 0x20000000, respectivamente.

Este procesador tiene una cache de datos de primer nivel (D1) con escritura a inmediata (Write Trough) y sin asignación en caso de fallo en escritura (Write NO Allocate). El tiempo de escribir una palabra en el siguiente nivel de la jerarquía (L2) es de 5 ciclos, por lo que se ha incorporado un buffer de escritura (write buffer), como se muestra en la figura de la derecha, para que el procesador no necesite esperar estos cinco ciclos en cada *store* que realiza. Un **buffer** de escritura funciona como una **cola FIFO**, los datos entran en la cola y van avanzando hasta la cabeza, el dato en la cabeza (en el ejercicio se le llama entrada 0 del buffer o Buffer[0]) es el que se escribe en memoria.



Todas las instrucciones se ejecutan en 1 ciclo (suponemos que nunca tendremos fallos en la cache de instrucciones). Las instrucciones de *store* (A y B en nuestro bucle) también se ejecutan en un ciclo si hay espacio en el buffer para almacenar el dato a escribir en L2. En caso que el buffer este lleno, el procesador se espera los ciclos necesarios, hasta que tenemos espacio en el buffer para ejecutar una instrucción de *store*.

En el **Cronograma 1:** (ver apendice al final de tema 3) se muestra un cronograma de la ejecución de las 2 primeras iteraciones del bucle en donde podemos ver el funcionamiento con un buffer de 1 sola entrada. En este cronograma se ha etiquetado el ciclo en que se ejecuta cada una de las instrucciones con la etiqueta usada en el código anterior. La fila "# Buffer" indica el numero de entradas ocupadas del buffer y la fila "Buffer[0]" indica el contenido de la entrada 0 del buffer. En este caso solo hay una entrada, en los cronogramas 2 y 3 se usan buffers con 2 y 3 entradas respectivamente. En nuestros cronogramas, dado que no hay lecturas de memoria que puedan interferir con las escrituras del buffer, la entrada Buffer[0] siempre se corresponde con los datos que se están escribiendo en L2 (ocupación bus).

En el ciclo 01 se ha ejecutado el store A, por lo que la entrada Buffer[0] estará ocupada por el dato a[0] durante los siguientes 5 ciclos. En el ciclo 02 el procesador intentaría ejecutar el store B, pero debe esperarse al ciclo 07 a que el buffer tenga entradas disponibles (Buffer[0] estará ocupado con el dato b[0] durante los siguientes 5 ciclos). En los ciclos 08, 09 y 10 se ejecutan las siguientes 3 instrucciones (que no realizan accesos a datos). En el ciclo 11 se inicia la iteración 1, pero dado que el buffer está ocupado, el procesador no puede ejecutar la instrucción A hasta el ciclo 13 (en donde escribe el dato a[1] en el buffer). Igualmente, B debe esperarse al ciclo 19 (en que escribe el dato b[1]).

a) **Completa** el **Cronograma 1:** hasta el ciclo 44.
   Calcula el CPI para N=1.000.000 iteraciones.
   Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

En el **Cronograma 2:** suponemos que el buffer dispone de 2 entradas (nótese que ahora es posible escribir en el buffer siempre que haya menos de 2 entradas ocupadas).

b) **Rellena** el **Cronograma 2:** hasta el ciclo 44.
   Calcula el CPI para N=1.000.000 iteraciones.
   Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

En el **Cronograma 3:** suponemos que el buffer dispone de 3 entradas (nótese que ahora es posible escribir en el buffer siempre que haya menos de 3 entradas ocupadas).

c) **Rellena** el **Cronograma 3:** hasta el ciclo 44.
   Calcula el CPI para N=1.000.000 iteraciones.
   Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

Como se puede observar, el tener 3 entradas en el buffer no representa una mejora sustancial para este código.

d) **Razona** ¿A que crees qué es debido?

Supongamos ahora que disponemos de un Merge Buffer de 3 entradas, en donde cada entrada del buffer puede almacenar un bloque de datos de 8 bytes debidamente alineados. En nuestro ejemplo se correspondería a una pareja de palabras consecutivas debidamente alineadas, es decir una entrada podría almacenar las parejas de datos a[0:1] o b[8:9] pero no las a[1:2] o b[3:4] dado que no se corresponden a un bloque de 8 bytes alineado.

Con el Merge Buffer, cuando se ejecuta una instrucción de *store*, primero se comprueba si el dato se puede combinar con una entrada existente, y si no es así se usa una nueva entrada. Sólo en caso que se necesite una nueva entrada y el buffer esté lleno será necesario bloquear el procesador, pero no si la nueva escritura se puede combinar con una entrada existente. Si la entrada 0 del buffer ya ha iniciado su escritura en L2, entonces no será posible la combinación con futuras escritura. Nótese que en nuestros cronogramas, dado que no hay lecturas de memoria que puedan interferir con las escrituras del buffer, la entrada Buffer[0] siempre se corresponde con datos que se están escribiendo en L2. Escribir un bloque de 8 bytes sigue tardando 5 ciclos. Para indicar que una entrada del buffer tiene una pareja de datos lo podemos escribir como: a[0:1].

e) **Rellena** el **Cronograma 4:** hasta el ciclo 44 en donde suponemos que tenemos un merge buffer de 3 entradas y 2 palabras (8 bytes) por entrada.
Calcula el CPI para N=1.000.000 iteraciones.
Calcula el ancho de banda (en bytes por ciclo) entre el buffer y L2 para N=1.000.000 iteraciones.

Problema 17. DRAM

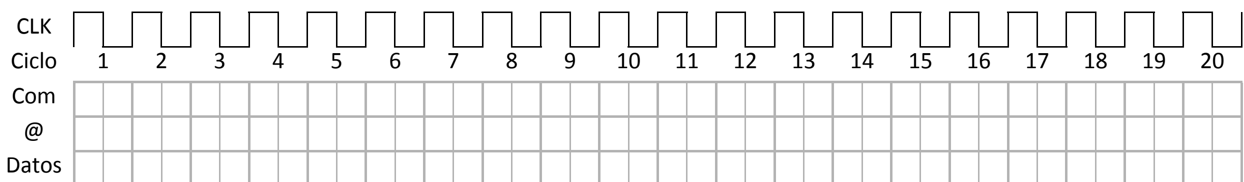
Disponemos de un DIMM de memoria DRAM síncrona (SDRAM) con las siguientes características:

- 8 chips de 1 byte cada uno por DIMM
- Latencia de fila: 4 ciclos
- Latencia de columna: 3 ciclos
- Latencia de precarga: 2 ciclos
- Frecuencia de reloj: 200 MHz

A esta memoria realizamos un acceso en lectura en el que leemos un bloque de 64 bytes. Para indicar la ocupación de los distintos recursos utilizaremos la siguiente nomenclatura:

- ACT: comando ACTIVE
- RD: comando READ
- PRE: comando PRECHARGE
- @F: ciclo en que se envía la dirección de fila
- @C: ciclo en que se envía la dirección de columna
- Di: ciclo en que se transmite el paquete de datos i (D0, D1, D2, ...)

a) **Rellenad** el siguiente cronograma indicando la ocupación de los distintos recursos para una operación de lectura de 64 bytes.



b) **Calculad** el tiempo de ciclo de la memoria en ns.

c) **Calculad** el ancho de banda teórico máximo suponiendo que el bus de datos está transfiriendo datos continuamente.

d) **Calculad** el ancho de banda real suponiendo que somos capaces de iniciar un nuevo acceso a un bloque de 64 bytes tan pronto hemos completado el acceso anterior.

La tensión de alimentación de esta memoria es de 1.5 voltios, mientras que la corriente consumida depende de la actividad:

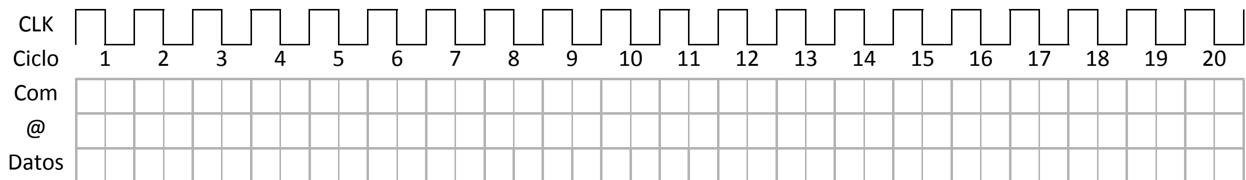
- La memoria esta inactiva -> corriente de fugas 200 mA

- Durante toda la operación de lectura (desde que se envía el comando ACTIVE hasta que se completa el PRECHARGE), se consumen 100 mA adicionales debidos al funcionamiento de los componentes internos (además de la corriente de fugas que sigue existiendo)
- Durante la transferencia de datos, además de la corriente de fugas y los componentes internos, hay que alimentar los drivers de entrada salida, con lo que se consumen otros 500 mA adicionales.

e) **Calculad** la energía consumida (en julios) y la potencia media consumida (en vatios) en la memoria durante un intervalo de 100 ciclos suponiendo que iniciamos un acceso cada 25 ciclos.

Después de unos años este DIMM de memoria SDRAM es sustituido por un DIMM DRAM DDR (Double Data Rate) manteniendo el resto de características iguales.

f) **Rellenad** el siguiente cronograma indicando la ocupación de los distintos recursos para una operación de lectura de 64 bytes en la nueva memoria DDR.



Problema 18. Cache Multinivell, DRAM

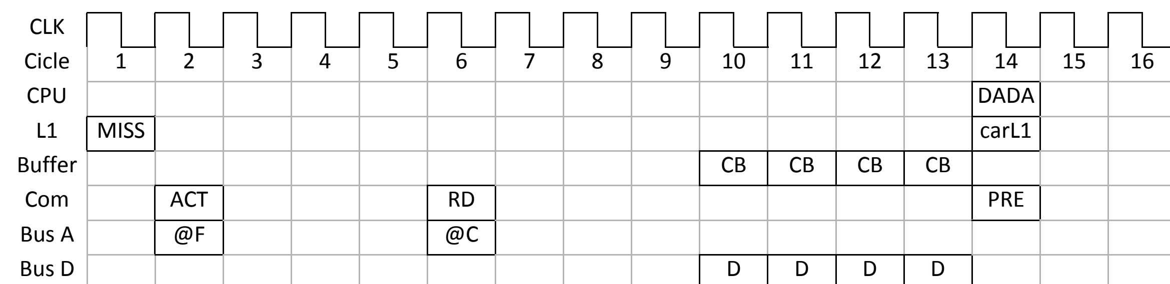
Tenim el disseny de una CPU que tindrà un temps de cicle (Tc) de 10 ns. A l'executar un programa P (que executa $5x10^9$ instruccions) en un simulador on tots els accessos a memòria tarden 1 cicle s'ha mesurat un CPI de 1,8 cicles/instrucció (que anomenarem CPI_{ideal}).

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 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%. La memòria principal esta formada per un DIMM SDRAM de 8 bytes d'amplada amb una latència de fila de 4 cicles, una latència de columna també de 4 cicles i un temps per la comanda PRECHARGE de 1 cicle.

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

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 de penalització d'una fallada (en cicles).

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

e) **Calculeu** el CPI amb aquesta jerarquia de memòria.

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

A aquest sistema afegim un segon nivell de cache (L2) entre la cache d'instruccions (L1) i la memòria principal (SDRAM) 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.

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

h) **Calculeu** 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											

i) **Calculeu** el temps de penalització en cas de fallar a L1 i encertar a L2 (en cicles).

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		

j) **Calculeu** el temps de penalització en cas de fallar a L1 i a L2 (en cicles).

k) **Calculeu** el temps mig d'accés a memòria (T_{mam}) pels accesos a instruccions (en nanosegons).

l) **Calculeu** el CPI amb aquesta jerarquia de memòria.

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

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

Problema 19. Detección y corrección de errores

Una forma habitual de expresar la fiabilidad es reportando fallos por tiempo o **FIT** (del inglés *failures in time*, ver 1.7 de Hennessy & Pattersson). **FIT** suele expresarse como fallos por cada 10^9 horas. En el caso de las memorias suele citarse fallos por 10^9 horas por Mbit. Un reciente estudio de google ha medido alrededor de 25.000 fallos por 10^9 horas por Mbit de memoria instalado en sus máquinas.

a) **Calculad** el MTTF (tiempo medio hasta fallo) por Mbit.

b) **Calculad** el MTTF de 1 bit individual (en horas y en millones de años).

c) **Calculad** el MTTF de la memoria de un computador con 16 GB de memoria instalados.

Sin ningún mecanismo de protección, cada uno de estos fallos podría hacer que la aplicación en que se producen genere resultados incorrectos, abortar la aplicación o incluso forzar que se tenga que reiniciar el sistema. Por ese motivo usan memorias **ECC** que permiten corregir 1 bit (incluso varios bits si se usa un mecanismo como el llamado ChipKill de IBM), en cuyo caso se dice que se ha producido un error recuperable. En caso de que el número de bits

erróneos sea superior a lo que es capaz de corregir el sistema **ECC** usado se dice que se ha producido un error no recuperable. Si se usan memorias **ECC**, sólo los errores no recuperables pueden tener consecuencias negativas para el sistema. En ese mismo estudio google estimó que aproximadamente 1 de cada 20.000 fallos era no recuperable.

d) **Calculad** el MTTF de la memoria de un computador con 16 GB de memoria **ECC** instalados.

En el caso de google, consideran que un error no recuperable es suficientemente grave como para reemplazar el DIMM que lo ha provocado. Se estima que google podría tener unos 500.000 servidores (en unos 40-60 datacenters repartidos por todo el mundo). Suponiendo que todos los servidores tuviesen 16 GB de memoria (en realidad no son todos iguales ni tenemos la más remota idea de la memoria que tienen).

e) **Calculad** cuantos DIMM cambia google cada día (a nivel mundial).

Se calcula que para fabricar un chip de memoria se consumen unos 70 MJoules de energía. Supongamos que cada DIMM tiene 18 chips de memoria (asumimos DIMMS con 2 rangos lo que hace 16 datos + 2 paridad). Sabemos además que para producir un Mjoule se emiten en media 50 g de CO_2

f) **Calculad** la energía equivalente consumida en fabricar los DIMMs que se reemplazan cada año y el CO_2 equivalente emitido (en toneladas).

Nota: Algunos datos de este problema son estimaciones o inventados, el resultado final no tiene porque guardar níngún parecido con los datos reales de google.

Problema 20. DRAM avanzada, prefetch

En un procesador que interpreta el lenguaje máquina x86 ejecutamos el siguiente código:

```
        movl $0, %esi
        movl $0, %eax
L:      movl a(,%esi,8), %ecx ; load a[2*i]
a:      addl %ecx, %eax       ; suma += a[2*i]
i:      incl %esi             ; i++
c:      cmpl $N, %esi         ; i<N ?
j:      jl L
```

En este código, que suma los elementos de un vector, se han etiquetado las instrucciones del bucle. La instrucción L realiza un *Load* del elemento a[2*i], siendo la dirección de inicio de a 0x20000000 y suponiendo que N=64.000.000 ($64x10^6$).

Este procesador funciona a una frecuencia de 2GHz y tiene una cache de datos de primer nivel (D1) con bloques de 32 bytes. Todas las instrucciones se ejecutan en 1 ciclo (suponemos que nunca tendremos fallos en la cache de instrucciones). Las instrucciones de *load* (L en nuestro bucle) también se ejecutan en un ciclo si se produce acierto en D1, sin embargo el procesador se bloquea en caso de fallo hasta que llegue el dato. En caso de fallo los bloques son leídos de una memoria DRAM síncrona. La SDRAM está formada por 1 DIMM (con 8 chips de 1 byte cada uno) con las siguientes latencias: latencia de fila 9 ciclos, latencia de columna 9 ciclos, latencia de precarga 8 ciclos.

En el **Cronograma 5:** se muestra un cronograma de la ejecución de las 2 primeras iteraciones del bucle en donde podemos ver los pasos a realizar en caso de fallo. En este cronograma se ha etiquetado el ciclo en que se ejecuta cada una de las instrucciones con la etiqueta usada en el código anterior. Las tres filas inferiores indican la ocupación del sistema de memoria, para lo que usamos la siguiente nomenclatura:

- Cache: **H** ciclo en que se produce un acierto, **M** ciclo en que se detecta un fallo, **D** ciclo en que la CPU recibe el dato después de servir un fallo.
- Comando SDRAM: **Ac** ciclo en que se inicia el comando ACTIVE, **Rd** ciclo en que se inicia el comando READ, **Pr** ciclo en que se inicia comando PRECHARGE.
- Datos SDRAM: **di** indica el ciclo en que se transmite el paquete de datos i (d0, d1, d2, ...).

En el ciclo 01 se inicia la ejecución de L (iteración 0), que provoca un fallo, por lo que la ejecución de esta instrucción se prolonga hasta el ciclo 24 en que recibe el dato. Debido a este fallo, en el ciclo 02 el controlador de memoria inicia la lectura de un bloque de datos enviando el comando Ac, 9 ciclos después envía el comando Rd (ciclo 11). En el ciclo 20 la memoria empieza a transmitir los datos que se escriben en la cache durante el mismo ciclo que se transfieren. Finalmente, en el ciclo 24, una vez completada la transmisión del bloque, se envía el dato a la CPU y se cierra el banco con el comando Pr. En los ciclos 25 a 28 se ejecutan el resto de instrucciones del bucle (a,i,c,j) que no realizan accesos a memoria. En el ciclo 29 se inicia la iteración 1 del bucle en donde se muestra el caso en que al ejecutar la instrucción L se produce un acierto en D1, por lo que la instrucción se puede ejecutar en un solo ciclo. Nótese que cuando se inicia la ejecución de L en el ciclo 29 (iteración 1), la memoria aún no ha completado el comando PRECHARGE

anterior, por lo que si el acceso hubiese sido un fallo, el comando ACTIVE no se podría lanzar hasta el ciclo 32 (aumentando aun más la penalización por fallo).

a) **Calcular** la tasa de fallos de la cache de datos (D1) al ejecutar el bucle.
b) **Completar** el **Cronograma 5:**
c) **Calcular** el CPI al ejecutar el bucle y el tiempo de ejecución del bucle

Para mejorar el rendimiento añadimos prefetch a la cache D1. Cuando accedemos a un bloque de datos (i), se lanza un prefecth del bloque siguiente (i+1) siempre que el siguiente (i+1) no esté ya en la cache o se esté sirviendo ya un prefecth del mismo. El acceso a SDRAM para realizar el prefetch de (i+1) lo lanzaremos, si es posible, el ciclo siguiente al acceso (i). Es posible que al lanzar un prefetch la memoria SDRAM esté ocupada por un acceso anterior (debido a un fallo o a otro prefecth), en este caso, el prefecth se iniciará tan pronto la memoria esté disponible. Nótese que en caso de fallo al acceder un bloque (i) es posible que se inicie un prefetch al siguiente (i+1), naturalmente serviremos primero el fallo (el prefetch se iniciará justo al completar el fallo). Por ejemplo, en el **Cronograma 5:**, el acceso a memoria realizado por L en la iteración 0 provoca un fallo que ocupa la SDRAM hasta el ciclo 31 y, en caso de tener prefectch, también provocaría un prefetch del bloque siguiente que no se iniciaría hasta el ciclo 32. En caso de que se produzca un fallo al acceder a un bloque del que se está realizando un prefecth no se accede a la SDRAM para servir el fallo, sino que se espera a que se complete el prefetch para servir el dato a la CPU, de esta forma no es necesario esperar todos los ciclos necesarios para servir un fallo (que ahora llamaremos fallo completo) sino solo los que quedaban para completar el prefetch (en este caso se dice que hemos tenido un fallo parcial).

En el **Cronograma 6:** se muestra el cronograma a partir de la iteración 1. En la parte correspondiente a la ocupación del sistema de memoria puede verse el acierto a cache de la instrucción L y la ocupación de la memoria debida al fallo (completo) de L en la iteración 0.

d) **Completar** el **Cronograma 6:** con el mecanismo de prefecth descrito.
e) **Calcular** el numero de fallos completos y el número de fallos parciales. ¿crees que, para calcular el CPI, vale la pena tener en cuenta los fallos completos, dado el numero de iteraciones que ejecutamos?

f) **Calcular** los ciclos perdidos en caso de tener un fallo parcial.
g) **Calcular** el CPI, el tiempo de ejecución del bucle y el speed-up respecto al procesador original.

Otra posible forma de mejorar el rendimiento consiste en aprovechar la localidad espacial de las filas de memoria (que los fabricantes llaman páginas). Como sabemos, el comando ACTIVE abre una página y el comando PRECHARGE cierra la página activa. En los dos procesadores considerados anteriormente todos los accesos a SDRAM abrían la página al inicio del acceso y la cerraban al final. Supongamos que los bloques de memoria se almacenan consecutivos dentro de una página y que una página de un chip SDRAM tiene 256 bytes (mira en las transparencias como se distribuye un bloque entre los 8 chips). Notese que el comando ACTIVE abre simultaneamente 8 páginas en los 8 chips del DIMM. Para este ejercicio supondremos un procesador (sin prefetch) con un controlador de memoria avanzado que no cierra la página (PRECHARGE) después de cada acceso. En caso de que un acceso se realiza sobre la página abierta no es necesario abrirla (ACTIVE). Sin embargo si el acceso se realiza sobre una página distinta tenemos que cerrar la anterior (PRECHARGE) y abrir (ACTIVE) la que se desea acceder (este caso incluye el primer acceso del bucle).

h) **Calcular** el numero de bloques que almacena una página, cuantos accesos a SDRAM tienen que abrir página y cuantos pueden reusar la página que está abierta. ¿crees que vale la pena tener en cuenta el hecho de que el primer acceso (y ninguno más) no necesita cerrar página (aunque si abrir la nueva)?
i) **Dibujar** en el **Cronograma 7:** el cronograma de un fallo de cache que reusa la página abierta.
j) **Dibujar** en el **Cronograma 8:** el cronograma de un fallo de cache que accede a una página distinta de la que está abierta.
k) **Calcular** el tiempo de penalización de los fallos que abren página y el de los que reusan una página abierta.
l) **Calcular** el CPI, el tiempo de ejecución del bucle y el speed-up respecto al procesador original.

Podríamos mejorar, aun más, el rendimiento combinando prefetch con el controlador de memoria avanzado. Como hemos visto anteriormente, cuando hay prefetch, solo el primer acceso (iteración 0) es un fallo completo y que podemos ignorarlo, por lo que solo tendremos en cuenta los fallos parciales. Sin embargo, con el controlador avanzado tenemos algunos prefetch que abren una página nueva, mientras que otros reusan una página abierta (la proporción debería ser la misma que en el apartado h)).

m) **Dibujar** en el **Cronograma 9:** el cronograma de ejecución del bucle a partir de la **iteración 4**, donde se inicia un prefecth que reusa la página abierta (por simplicidad, se han numerado los ciclos a partir de 01, aunque la iteración 4 no empieza en el ciclo 01)

n) **Dibujar** en el **Cronograma 10:** el cronograma de ejecución del bucle a partir de la **iteración 252** (en la iteración 252=4*63 se accede al bloque de memoria 63 y se inicia el prefetch del bloque de memoria 64), donde se inicia un prefecth que accede a una página distinta de la que está abierta (por simplicidad, se han numerado los ciclos a partir de 01, aunque la iteración 252 no empieza en el ciclo 01)

o) **Calcular** los ciclos perdidos por fallo parcial en ambos casos (si procede). ¿Se produce fallo parcial con los dos tipos de prefetch?

p) **Calcular** el CPI, el tiempo de ejecución del bucle y el speed-up respecto al procesador original.

Sabemos que las SDRAM actuales tienen múltiples bancos. Supongamos que nuestra SDRAM tiene 2 bancos y que las direcciones de memoria están entrelazadas entre bancos a nivel de página. Es decir los primeros 2k bytes se almacenan en el banco 0, los siguientes 2k en el 1 y los siguientes 2k nuevamente en el 0. Este hecho lo aprovecharemos modificando ligeramente el controlador avanzado de SDRAM. Dado que tenemos dos bancos, no es necesario cerrar la página abierta en un banco para abrir una nueva página en el otro banco. De hecho es posible solapar parcialmente el comando PRECHARGE que cierra la página de un banco con el comando ACTIVE que abre una página en el otro banco e incluso con el acceso READ correspondiente. Supongamos de momento que no tenemos prefetch.

q) **Dibujar** en el **Cronograma 11:** el cronograma de un fallo de cache que accede a una página distinta de la que está abierta. Calcula el orden más adecuado de los comandos de SDRAM para que la CPU reciba el dato en el número mínimo de ciclos y que todo el proceso tarde lo menos posible.

r) **Calcular** el tiempo de penalización de los fallos que abren página y el de los que reusan una página abierta.

s) **Calcular** el CPI, el tiempo de ejecución del bucle y el speed-up respecto al procesador original.

Como seguramente habréis imaginado, esta última mejora también la podemos aplicar al procesador con prefecth. Teniendo en cuenta la experiencia de los apartados anteriores.

t) **Calcular** el CPI, el tiempo de ejecución del bucle y el speed-up respecto al procesador original.

Apéndice: Cronogramas Tema 3

Problema 16

Cronograma 1: Buffer de 1 entrada.

Iteración	<---------------Iteración 0 ------------->										<-------------------Iteración 1 ------------------->																																								
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)	A										-	-	A																															
movl %eax, b(,%esi,4)		-	-	-	-	-	B							-	-	-	-	-	B																									
incl %esi								i												i																								
cmpl $N, %esi									c												c																							
jl A										j												j																						
Ocupación bus				a[0]						b[0]						a[1]						b[1]																						
# Buffer	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1	0	1	1	1	1	1																				
Buffer[0]				a[0]						b[0]						a[1]						b[1]																						

CPI = ... Ancho de banda = ...

Cronograma 2: Buffer de 2 entradas

Iteración																																												
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)																																												
movl %eax, b(,%esi,4)																																												
incl %esi																																												
cmpl $N, %esi																																												
jl A																																												
Ocupación bus																																												
# Buffer																																												
Buffer[0]																																												
Buffer[1]																																												

CPI = ... Ancho de banda = ...

Cronograma 3: Buffer de 3 entradas

Iteración																																												
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)																																												
movl %eax, b(,%esi,4)																																												
incl %esi																																												
cmpl $N, %esi																																												
jl A																																												
Ocupación bus																																												
# Buffer																																												
Buffer[0]																																												
Buffer[1]																																												
Buffer[2]																																												

CPI = ... Ancho de banda = ...

Cronograma 4: Merge buffer de 3 entradas

Iteración																																												
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl %eax, a(,%esi,4)																																												
movl %eax, b(,%esi,4)																																												
incl %esi																																												
cmpl $N, %esi																																												
jl A																																												
Ocupación bus																																												
# Buffer																																												
Buffer[0]																																												
Buffer[1]																																												
Buffer[2]																																												

CPI = ... Ancho de banda = ...

Problema 20

Cronograma 5: SIN prefetch

Iteración	<---------------------------Iteración 0--------------------------->																												<-- Iteración 1 -->															
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl a(,%esi,8), %ecx	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L	L					L															
addl %ecx, %eax																									a					a														
incl %esi																										i					i													
cmpl $N, %esi																											c					c												
jl L																												j					j											
Cache	M																							D					H															
Comando SDRAM		Ac									Rd													Pr																				
Datos SDRAM																				d0	d1	d2	d3																					

Cronograma 6: CON prefetch

Iteración	<-- Iteración 1 -->																																											
Ciclo	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72
movl a(,%esi,8), %ecx	L																																											
addl %ecx, %eax		a																																										
incl %esi			i																																									
cmpl $N, %esi				c																																								
jl L					j																																							
Cache	H																																											
Comando SDRAM																																												
Datos SDRAM																																												

Cronograma 7: Fallo que NO abre página.

Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
Cache	M																																											
Comando SDRAM																																												
Datos SDRAM																																												

Cronograma 8: Fallo que SI abre página.

Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
Cache	M																																											
Comando SDRAM																																												
Datos SDRAM																																												

Cronograma 9: Prefecth que NO abre página.

Iteración																																												
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl a(,%esi,8), %ecx																																												
addl %ecx, %eax																																												
incl %esi																																												
cmpl $N, %esi																																												
jl L																																												
Cache																																												
Comando SDRAM																																												
Datos SDRAM																																												

Cronograma 10: Prefecth que SI abre página.

Iteración																																												
Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
movl a(,%esi,8), %ecx																																												
addl %ecx, %eax																																												
incl %esi																																												
cmpl $N, %esi																																												
jl L																																												
Cache																																												
Comando SDRAM																																												
Datos SDRAM																																												

Cronograma 11: Fallo que abre página en la SDRAM con dos bancos.

Ciclo	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44
Cache	M																																											
Comando SDRAM																																												
Datos SDRAM																																												

Problemas Tema 4

Problema 1. Discos, Ancho de banda, RAIDs

Disponemos de discos con las siguientes características:

- Capacidad 1 Tbyte.
- Seek time medio (situar el cabezal en el cilindro) 8 ms.
- Latencia media (tiempo que tarda en pasar el sector deseado) 2 ms.
- Transfer Rate (ancho de banda durante la transferencia de datos) 256 Mbytes/s.
- MTTF 50.000 horas.
- Tamaño de sector 512 bytes.

a) **Calcula** cuanto tiempo tarda en transferirse un bloque de datos de 5000 sectores.

Si los 5000 sectores se encuentran almacenados de forma consecutiva en la misma pista, solo se pierde tiempo en situar el cabezal (seek time + latencia) en el primero de ellos ya que el disco es capaz de transferir los siguientes sectores a medida que gira el disco sin necesidad de situar el cabezal (suponemos que la velocidad de rotación es la adecuada para al Transfer Rate).

b) **Calcula** el tiempo total necesario para leer un bloque de datos de 5000 sectores consecutivos desde que se envía la petición al disco hasta que los datos están en memoria (suponemos que el procesado de las peticiones por el disco, calculo de CRCs, DMA, etc introducen un tiempo negligible).

c) **Calcula** el ancho de banda efectivo al leer un bloque de datos de 5000 sectores consecutivos.

En un computador con uno de estos discos ejecutamos una aplicación formada por 3 fases:

- La fase 1 lee 8 bloques de datos (de 5000 sectores consecutivos cada uno) de disco.
- La fase 2 realiza los calculos y representa el 40% del tiempo de la aplicación (ejecutada con un solo disco).
- La fase 3 escribe 4 bloques de datos (de 5000 sectores consecutivos cada uno) a disco. Escribir un bloque de datos tarda lo mismo que leerlo.

d) **Calcula** el tiempo que tarda la fase 2.

Aunque los datos caben en un solo disco, para mejorar el rendimiento de la aplicación, instalamos un RAID 0 con 8 discos iguales con tiras de 5000 sectores de forma que los 8 bloques de datos de la fase 1 se encuentran en 8 discos distintos. Igualmente, los 4 bloques de la fase 3, también se encuentran en 4 discos distintos. Suponemos que el resto del sistema es capaz de soportar el ancho de banda necesario.

e) **Calcula** el ancho de banda efectivo al leer los 8 bloques de datos del RAID 0

f) **Calcula** el ancho de banda efectivo al escribir los 4 bloques de datos al RAID 0

g) **Calcula** el speed-up de la fase 1

h) **Calcula** el speed-up de la fase 3

i) **Calcula** el speed-up de la aplicación

Problema 2. RAIDS, ancho de banda

Disponemos de 60 discos físicos de 300 Gbytes de capacidad por disco, que ofrecen un ancho de banda efectivo de 100 Mbytes/s por disco. Con estos discos deseamos montar un disco lógico en donde consideramos las siguientes 4 opciones:

- RAID 6
- RAID 10 (mirror doble con 30 grupos de 2 discos)
- RAID 50 (con 6 grupos de 10 discos)
- RAID 51 (mirror doble con 2 grupos de 30 discos)

a) **Calcular** la cantidad de información útil que puede almacenar cada uno de los RAIDS considerados.

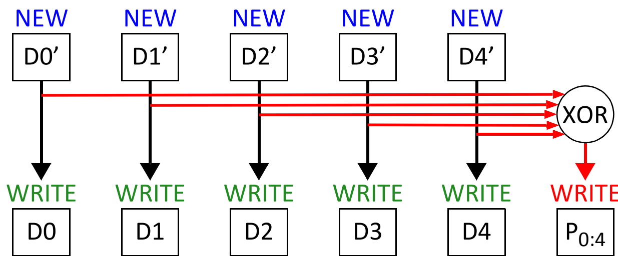
Para analizar el ancho de banda consideraremos por separado el caso en que realizamos accesos a tiras consecutivas y el caso en que realizamos accesos a tiras aleatorias. En el caso de accesos aleatorios, el controlador del RAID ordena las peticiones de acceso para aprovechar al máximo la concurrencia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

que disponemos de suficientes peticiones de lectura de forma que el controlador del RAID siempre puede aprovechar el ancho de banda de todos los discos físicos.

b) **Calcular** el ancho de banda efectivo, si hacemos **lecturas** secuenciales, para cada uno de los RAIDS considerados.

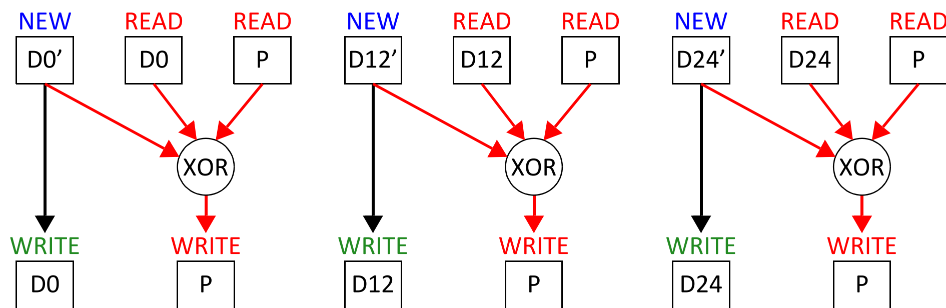
c) **Calcular** el ancho de banda efectivo, si hacemos **lecturas** aleatorias, para cada uno de los RAIDS considerados.

Para aquellos RAIDS que tienen algún tipo de paridad (como 6, 50 o 51), 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. La siguiente figura muestra la escritura de 5 tiras de datos consecutivas D0-D4 en lo que podria ser un RAID 4 o 5 con 6 discos. Obsérvese que podemos aprovechar el ancho de banda de 5 discos para datos ya que el sexto es usado para escribir la paridad.



d) **Calcular** el ancho de banda efectivo, si hacemos **escrituras** secuenciales, para cada uno de los RAIDS considerados.

En el caso de escrituras aleatorias (para RAIDS con algún tipo de paridad), se ha visto en teoría que es necesario leer los datos antiguos y la tira (o tiras) de paridad correspondiente para calcular la nueva paridad y además escribir tanto datos como paridad por cada escritura que se desea realizar. La siguiente figura muestra la escritura en paralelo de 3 tiras de datos independientes en lo que podría ser un RAID 5 con 6 discos (para acabarlo de entender es recomendable dibujar como estarían distribuidas las tiras de la 0 a la 24 y las paridades correspondientes en un RAID 5 con 6 discos). Obsérvese que para escribir 3 tiras de datos es necesario realizar 12 operaciones de disco realizando una lectura y una escritura en cada uno de los 6 discos, con lo que en la práctica el ancho de banda efectivo (de datos) es equivalente a 1,5 discos.



e) **Calcular** el ancho de banda efectivo, si hacemos **escrituras** aleatorias, para cada uno de los RAIDS considerados.

Problema 3. RAIDS, Fiabilidad

En clase de teoría se ha deducido la expresión para el tiempo medio entre fallos (**$MTTF_{RAID}$**) de los RAID 0, 1, 3, 4 y 5. En este ejercicio vamos a deducir la expresión general del tiempo medio entre fallos del RAID completo (**$MTTF_{RAID}$**) para RAID 6, 10, 50 y 51 donde el tiempo medio entre fallos de un disco es **$MTTF_d$**, el tiempo de recuperación de un disco es **MTTR**, el número total de discos es **N** y, si es el caso, el número de discos por grupo es **G**.

En un RAID 10 (mirror doble organizado N/2 grupos de 2 discos), una vez ha fallado un disco cualquiera, el sistema solo falla si falla el mirror de este. Cualquier otro disco que falle, el sistema puede seguir funcionando.

a) **Deducir** la expresión general del $MTTF_{RAID}$ para un RAID 10 de N discos.

En un RAID 50 organizado en N/G grupos de G discos, una vez ha fallado un disco, el sistema solo falla si falla un segundo disco del mismo grupo. Si falla un disco de otro grupo el sistema puede seguir funcionando.

b) **Deducir** la expresión general del $MTTF_{RAID}$ para un RAID 50 de N discos con grupos de G discos.

En un RAID 6 de N discos, una vez ha fallado un disco, el sistema se comporta (a efectos de fiabilidad) como un RAID 5 con N-1 discos. El sistema puede soportar el fallo de un segundo disco y solo falla en caso de que falle un tercer disco.

c) **Deducir** la expresión general del $MTTF_{RAID}$ para un RAID 6 de N discos.

En un RAID 51 (mirror doble organizado 2 grupos de N/2 discos), el sistema solo fallará si fallan 2 discos dentro de un grupo (mirror) y fallan otros dos discos en el otro grupo (mirror). Es decir puede tolerar un primer fallo en cualquier disco, un segundo fallo en el mismo grupo, un tercer fallo en cualquiera de los discos del otro grupo y finalmente el sistema fallara si falla un cuarto disco del otro grupo.

d) **Deducir** la expresión general del $MTTF_{RAID}$ para un RAID 51 de N discos.

Disponemos de 60 discos físicos con los que deseamos montar un disco lógico en donde consideramos las siguientes 4 opciones:

- RAID 6
- RAID 10 (mirror doble con 30 grupos de 2 discos)
- RAID 50 (con 6 grupos de 10 discos)
- RAID 51 (mirror doble con 2 grupos de 30 discos)

El tiempo medio entre fallos MTTF de un disco es de 60.000 horas, mientras que el tiempo de cambiar un disco y reconstruir la información (MTTR) es de 30 horas.

e) **Calcular** el $MTTF_{RAID}$ para cada uno de los RAIDS considerados.

Problemas Tema 5

Problema 1. Tipos de máquinas

Dada la siguiente expresión donde R, A,B,C y D son variables globales:

```
R = C-(A-B)/(C-D)
```

a) **Programa** una secuencia de código que la evalue en una arquitectura de tipo pila con las siguientes operaciones:

```
add/sub/mul/div   #pila[top+1]=pila[top] op pila[top+1]; top=top+1
push @            #top=top-1; pila[top]=M[@]
pop @             #M[@]=pila[top]; top=top+1
```

b) **Programa** una secuencia de código que la evalue en una arquitectura de tipo acumulador con las siguientes operaciones:

```
add/sub/mul/div @   #ACC=ACC op M[@]
load @              #ACC=M[@]
store @             #M[@]=ACC
```

Problema 2. RISC-CISC

Un cachivache electrónico (llamado i-Cachivache) ejecuta repetidamente un programa que no puede tardar más de 2,5 segundos. Para ello el i-Cachivache incorpora un procesador CISC registro-memoria. Al ejecutar el programa compilado para dicho procesador hemos obtenido los siguientes datos:

- 10^9 instrucciones dinámicas ejecutadas
- CPI 2,5 ciclos/instrucción
- el 30% de las instrucciones tienen un operando en memoria
- el 10% de las instrucciones tienen dos operandos en memoria
- el 70% de los accesos a memoria usan el modo base+desplazamiento
- el 20% de los desplazamientos se codifican con más de 16 bits
- el 20% de los accesos a memoria usan el modo registro indirecto (equivalente a base + desplazamiento cero)
- el 20% de las instrucciones tienen un operando inmediato
- el 15% de los inmediatos se codifican con más de 16 bits.

a) ¿Cuantos accesos a datos se realizan durante la ejecución del programa?

b) ¿A que frecuencia debemos hacer funcionar el procesador para que el programa se ejecute en el tiempo previsto?

En la segunda generación del cachivache (el i-Cachivache 2) deseamos sustituir el procesador CISC por un RISC que esperamos tenga un menor consumo y aumente la duración de la batería. Para ello disponemos de un traductor que traduce del lenguaje máquina CISC al RISC. Al traducir el programa hemos obtenido que:

- El 90% de las instrucciones CISC se han podido mapear sobre una instrucción RISC equivalente. El 10% restante ha necesitado 2 instrucciones RISC.
- Los accesos a memoria han necesitado una instrucción adicional de Load/Store.
- Los accesos a memoria que no se han podido implementar usando el modo base+desplazamiento han necesitado otra instrucción adicional para calcular la dirección.
- Los load/store con desplazamientos de más de 16 bits han necesitado una instrucción adicional ya que el RISC sólo dispone de 16 bits para codificarlos.
- Los inmediatos que necesitan más de 16 bits también han necesitado una instrucción adicional ya que los inmediatos también están limitados a 16 bits.

c) ¿Cuantas instrucciones dinámicas ejecutará el procesador RISC?

Al ejecutar el programa traducido en el procesador RISC hemos obtenido un CPI de 1,2 ciclos/instrucción.

d) ¿A que frecuencia deberíamos hacer funcionar el procesador RISC?

El procesador RISC es ligeramente más pequeño que el CISC por lo que su intensidad de fuga y su carga capacitiva son también menores, 10 A y 50 nF en el CISC por solo 8 A y 40 nF para el RISC. Ambos procesadores funcionan con una tensión de alimentación de 1 V. En ambos procesadores la potencia de cortocircuito es despreciable, por lo que solo tendremos en cuenta la potencia de fuga y la de conmutación.

e) Calcular la energía consumida por ambos procesadores al ejecutar el programa.

f) Calcular la ganancia en duración de batería del RISC sobre el CISC

Al cabo de un tiempo se dispone de un compilador que nos permite compilar el programa directamente para el procesador RISC. Al ejecutar el programa compilado directamente para el RISC, vemos que el número de instrucciones dinámicas se ha reducido a $1{,}5x10^9$ instrucciones, pero el CPI ha aumentado a 1,3 ciclos/instrucción. Esto nos permitirá sacar al mercado el i-Cachivache 3 que, a parte de un nuevo diseño más “cool” de la carcasa, es idéntico al i-Cachivache 2, pero con la nueva versión del código).

g) ¿A que frecuencia debería funcionar la CPU RISC con el nuevo programa compilado?

h) ¿Cual será la ganancia en duración de batería con el nuevo código?

Problema 3. Microoperaciones

Los actuales procesadores CISC de la familia x86 (tanto de Intel como AMD) traducen internamente las instrucciones de lenguaje máquina x86 a microperaciones (que denominan uops). En una implementación de x86 tenemos las siguientes uops:

Tipo	Descripción	Ejemplos
LOAD	Rd <- M[Rb+ Ri*s+dspl]	LOAD %r1 <- M[%ebx+%edx*4+36]
STORE	M[Rb+ Ri*s+dspl] <- Rf	STORE M[%ebx+%edx*4+36] <- %r3
MOV	Rd <- Rf1 Rd <- inmediato	MOVL %eax <- $3 MOVL %r1 <- %ebx
Aritmética	Rd <- Rf1 (op) Rf2 Rd <- Rf1 (op) inmediato	ADDL %r1 <- %r2 + %eax IMULL %eax <- %r4 * $20
Comparación	flags <- Rf1 - Rf2 flags <- Rf1 - inmediato	CMPL %eax , %r5 CMPL %eax , $100
Salto	mismos saltos que x86	JGE loop

Solo las instrucciones de LOAD y STORE pueden acceder a memoria, el resto sólo pueden tener registros o inmediatos como operandos. Ademas de los registros visibles en x86 disponemos de registros adicionales para almacenar valores intermedios (%r1 ... %r8). Los modos de direccionamiento de LOAD y STORE son los mismos que x86.

a) **Traducir** las siguiente secuencia de instrucciones en x86 a la correspondiente secuencia de microoperaciones

```
        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:
```

b) **Calcular** cuantas instrucciones dinámicas y cuantas uops dinámicas se ejecutan

Al ejecutar este código nuestra CPU obtiene un UPC (uops por ciclo) de 1,3

c) **Calcular** el CPI de este código

Suponiendo una frecuencia de 3GHz

d) **Calcular** el tiempo de ejecución de este código

Supongamos que las instrucciones x86 ocupan:

- 1 byte de código de operación (OpCode)
- 1 byte adicional si tiene 2 o más operandos (Modo), las de 1 sólo operando lo codifican dentro del OpCode
- 4 bytes adicionales si alguno de los operandos es un inmediato (incluidos los destinos de los saltos)
- 1 byte adicional (SIB) si accede a memoria
- 4 bytes adicionales si el modo de direccionamiento incluye desplazamiento

Las uops ocupan todas 6 bytes.

e) **Calcular** el tamaño del código x86 y el que ocuparían las uops equivalentes.

La familia *Sandy Bridge* de Intel (2011) incluye, ademas de una cache de instrucciones de nivel 1, una cache de micro-ops (que denominan de nivel 0) 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.

f) **Calcular** el numero de bytes leidos y el ancho de banda efectivo de la cache de instrucciones al ejecutar este código (sin uop cache)

g) **Calcular** el numero de bytes leidos y el ancho de banda efectivo de la cache de uops al ejecutar este código (con uop cache)

Una de las ventajas de la cache de micro-uops es que se produce un ahorro de energía ya que no es necesario descodificar todas las instrucciones dinámicas. En nuestro procesador, un acceso, tanto a la cache de instrucciones como a la cache de uops, consume 1 nJ (nanoJulio) por byte leido. Descodificar una instrucción consume 10 nJ. El código ejemplo cabe tanto a la cache de instrucciones como en la cache de uops, que inicialmente están vacias.

h) **Calcular** la energía consumida por nuestro programa en un procesador sin cache de uops (las instrucciones se leen de la cache de instrucciones y se decodifican cada vez) , en uno con cache de uops (las instrucciones se leeen ya decodificadas de la cache de uops, la cache de datos y el decodificador solo se usan cuando es fallo en la cache de uops) y la ganancia en consumo de energía debida a la cache de uops durante la búsqueda y descodificación de instrucciones.

Problemas Tema 6

Problema 1. Segmentación, riesgos de control.

Uno de los problemas de los procesadores segmentados (y también superescalares y VLIW) es la penalización introducida por los saltos. En los procesadores actuales con muchas etapas 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*).

Como ejemplo supongamos un procesador superescalar segmentado ideal (sin penalizaciones por saltos ni fallos de cache, ni bloqueos debidos a riesgos de datos o estructurales) que es capaz de ejecutar 4 instrucciones por ciclo.

a) **Calcular** el CPI del procesador ideal.

Supongamos ahora que los saltos introducen una penalización de 20 ciclos (manteniendo ideal el resto del procesador).

b) ¿Cuantas instrucciones se dejan de ejecutar cada vez que se ejecuta una instrucción de salto?

La cantidad de saltos que ejecutan los programas no es nada despreciable, por ejemplo, el conjunto de benchmarks SpecInt ejecutan en una arquitectura x86 un 20% de instrucciones de salto. Supongamos que en nuestro procesador ejecutamos un programa con un 20% de saltos.

c) **Calcular** el CPI teniendo encuentra las penalizaciones por salto.
d) **Calcular** cuantas veces más lento se ejecuta el programa por culpa de los saltos respecto el procesador ideal.

Para solventar el problema de los saltos, los procesadores actuales incorporan predictores de saltos en la fase de búsqueda de instrucciones que les permite predecir cual será la siguiente instrucción a ejecutar después de un salto. El procesador ejecuta instrucciones de forma *especulativa* de acuerdo con el predictor de saltos. Cuando finalmente se conoce realmente si el salto salta o no (es decir, se comprueba si el predictor ha acertado o no), es caso de acierto no se ha perdido ningún ciclo (ya se han ejecutado las instrucciones que tocaba), mientras que en caso de fallo se han perdido los ciclos de penalización por salto (ejecutando instrucciones que NO tocaba) y se deben descartar los resultados de las instrucciones ejecutadas de forma *especulativa*.

Supongamos que a nuestro procesador le incorporamos un predictor de saltos que en nuestro programa tienen una tasa de aciertos del 95%.

e) **Calcular** el CPI con el procesador con predictor de saltos.
f) **Calcular** el speedup respecto el procesador sin predictor de saltos.

Problema 2. VLIW, Jerarquia de memória.

Tenemos un procesador VLIW segmentado en el que cada instrucción VLIW puede codificar un máximo de 6 operaciones independientes (cada operación realiza cálculos equivalentes a una instrucción RISC). Dado que en un VLIW una sola instrucción equivale a muchas operaciones, se define una métrica adicional denominada OPC (operaciones por ciclo). Este VLIW sigue el clásico pipeline de 5 etapas:

- F: Búsqueda de la instrucción (*fetch*)
- D: Descodificación y lectura de registros
- E: Ejecución y calculo de direcciones
- M: Acceso a memoria
- W: Escritura de registros (*write*)

Un programa compilado para este procesador VLIW se ha interpretado en un simulador con memoria ideal (entendiendo por ideal el que los accesos a instrucciones se pueden completar durante la etapa F y los accesos a datos se pueden completar durante la etapa M sin necesitar ciclos adicionales). Se han ejecutado 10^9 instrucciones VLIW en 10^9 ciclos con una media de 4 operaciones por instrucción.

a) **Calcular** el IPC y el OPC del programa
b) ¿Cual debería ser el IPC de un procesador RISC superescalar que funciona a la misma frecuencia para obtener el mismo rendimiento.

Analizando los accesos a datos, hemos comprobado que el 20% de las instrucciones VLIW no realizan ningún acceso a memoria, el 40% realiza 1 acceso a memoria y el 40% realiza 2 accesos a memoria simultáneos. Nótese que para obtener el rendimiento anterior es necesario que la cache de datos permita dos accesos por ciclo (además de caches de datos e instrucciones separadas para que no haya riesgos estructurales). Una memoria cache con dos puertos tiene mayor coste (área de chip), mayor tiempo de acceso (que podría repercutir en el tiempo de ciclo) y mayor consumo que una equivalente con un solo puerto. Por ello se desea evaluar el impacto en el rendimiento de usar memorias con un solo puerto (menor coste, tiempo de acceso y consumo). De momento supondremos que no hay fallos de cache.

Memoria cache con un solo puerto y un solo banco. En caso que una misma instrucción VLIW realize 2 accesos a datos simultaneos (esto seria un buen ejemplo de riesgo estructural), el procesador se bloquea durante un ciclo y realiza los accesos en secuencia.

c) **Calcular** los ciclos que tarda el VLIW con un solo banco y un solo puerto
d) **Calcular** el IPC y el OPC

Una alternativa mejor es partir la cache en bancos de un solo puerto de forma que cuando una instrucción VLIW realice 2 accesos a datos simultaneos, solo será necesario bloquear el procesador si ambos accesos acceden al mismo banco. Suponemos que cualquier acceso puede acceder a cualquier banco con la misma probabilidad.

e) **Calcular** la probabilidad de que 2 accesos simultáneos vayan al mismo banco en una cache con 4 bancos.
f) **Calcular** los ciclos que tarda el VLIW con una cache con 4 bancos.
g) **Calcular** el IPC y el OPC.

Problema 3. (3 puntos)

Se ha medido la ejecución de un programa en un sistema con un solo procesador y un solo disco (un PC de sobremesa) y se ha visto que su tiempo de ejecución es de 200 horas.

Dado que el coste en tiempo es muy alto, deseamos ejecutar el programa (que en parte es paralelizable) en un sistema multiprocesador con procesadores idénticos al de nuestro PC. Para poder estimar el rendimiento del programa hemos medido (en el PC) que el programa se ejecuta en 3 fases bien diferenciadas:

Fase 1: Código SECUENCIAL que no puede paralelizarse, ocupa el 5% del tiempo en la ejecución en el PC.

Fase 2: Código TOTALMENTE PARALELIZABLE, ocupa el 85% del tiempo si se ejecuta el programa en el PC.

Fase 3: Código de E/S que no puede paralelizarse, ocupa el 10% del tiempo en la ejecución en el PC.

a) **Calcular** la ganancia máxima en velocidad que se puede conseguir en este programa si dispusiéramos para ejecutarlo de un supercomputador compuesto por un número infinito de procesadores y 1 disco.

En realidad, aunque la Fase 2 sea muy paralelizable, no es posible realizar una paralelización completa debido al tiempo de sincronización entre los distintos procesos. Además se calcula que el tiempo de sincronización aumenta conforme aumenta el número de procesos por lo que hay un límite a la cantidad de procesadores que podemos usar ganando velocidad. Suponiendo que el tiempo de cálculo de la Fase 2 se puede dividir de forma perfecta entre los procesadores que se usan pero que el tiempo de sincronización aumenta en un 0,5% del tiempo de ejecución en el PC por cada procesador.

b) **Calcular** la expresión matemática del tiempo de ejecución total del programa en función del número de procesadores (N).

A partir de la ecuación resultado del apartado anterior nos interesa encontrar la cantidad de procesadores (N) que resulta en un tiempo de ejecución mínimo. Para ello podemos derivar la ecuación e igualar a 0 para obtener sus mínimos y máximos. A continuación basta con probar para estos valores de N cual es el que da el tiempo total mínimo en la ecuación original.

c) Deriva la ecuación anterior, igualala a cero y calcula el número de procesadores (N) en los que el tiempo de ejecución total del programa es mínimo.

Del resultado anterior sabemos que el tiempo de ejecución de la Fase 2 en cada procesador (sincronización incluida) es de aproximadamente 26 h.

d) **Calcular** cual es la ganancia en tiempo real del programa cuando se ejecuta en un entorno con el número ideal de procesadores y un disco.

Nuestro supercomputador, además de poseer muchos procesadores también dispone de un RAID de discos. Esto nos permite suponer que la Fase 3 se realizará más rápidamente ya que en esta fase hay suficientes accesos como para saturar el ancho de banda de los discos. Sabemos además que todos los accesos son aleatorios.

e) **Calcular** la ganancia en tiempo con respecto a un PC de sobremesa que obtendríamos en el programa si lo ejecutaramos en un entorno con un solo procesador y un RAID 0 formado por 10 discos.

Nuestro sistema supercomputador está compuesto por un RAID 5 de 10 discos, cada uno de los cuales es igual al del PC de sobremesa. Sabemos que en nuestro programa el tiempo de la Fase 3 (es decir, el tiempo donde se trabaja con los discos) se gasta la mitad en escrituras y la mitad en lecturas. A partir de aquí suponemos que siempre que hablamos del supercomputador nos referimos a la configuración con el número ideal de procesadores y un RAID 5 de 10 discos.

f) **Calcular** cuanto tiempo tardará la Fase 3 de nuestro programa ejecutada en el supercomputador.

g) **Calcular** la ganancia en tiempo del supercomputador con respecto al PC de sobremesa.

Sabemos que el programa contiene $648 * 10^{13}$ instrucciones dinámicas de las cuales $216 * 10^{13}$ son instrucciones de coma flotante que realizan un total de $72 * 10^{13}$ operaciones de coma flotante.

h) **Calcular** a cuantos MIPS y MFLOPs se ejecuta el programa en el PC de sobremesa .

Al pasar el programa al supercomputador medimos que las instrucciones dinámicas de sincronización son $13 * 10^{13}$ en total.

i) **Calcular** a cuantos MIPS y MFLOPs se ejecuta el programa en el supercomputador.

Sabemos que la potencia consumida por cada procesador (tanto del PC de sobremesa como del supercomputador, que son iguales) es de 90 W, mientras que la potencia consumida por cada disco (también son todos iguales) es de 30 W. Suponemos que estos son los únicos elementos con consumo significativo y que se encuentran funcionando durante toda la ejecución del programa

j) **Calcular** los MFLOPS/W tanto del PC de sobremesa como del supercomputador.

k) **Calcular** el incremento de eficiencia energética que podríamos obtener para el supercomputador si consiguiéramos apagar completamente los elementos que no se utilizan (es decir, 12 de los 13 procesadores durante las etapas de cálculo secuencial y los 10 discos durante todas las etapas sin E/S).