Problema 17. DRAM

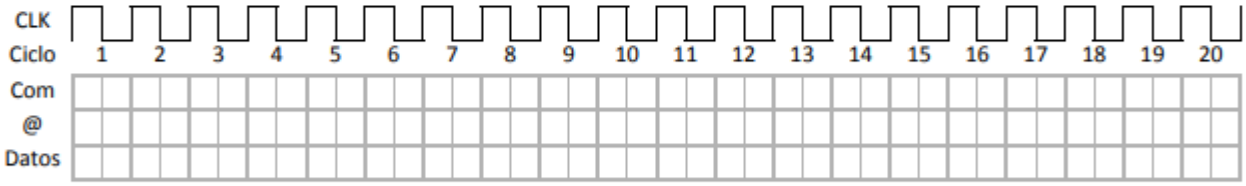
Disponemos de un DIMM de memoria DRAM síncrona (SDRAM) con las siguientes características:

- 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:

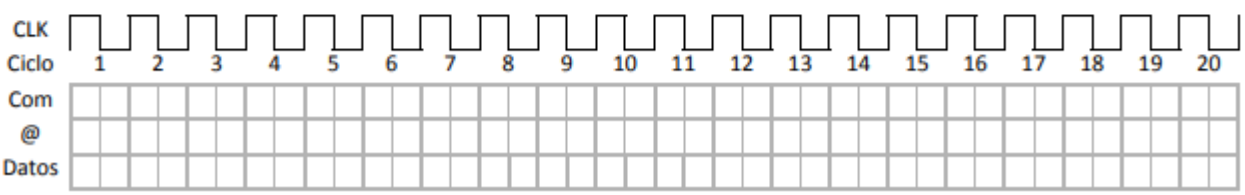
- La memoria esta inactiva -> corriente de fugas 200 mA

- 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 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 ningún 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.

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																																												