Laboratorio Sesión 07: Investigando la jerarquía de memoria

Objetivo

El objetivo de esta sesión es desarrollar una serie de herramientas que nos permitirán averiguar alguno de los parámetros fundamentales de la memoria cache de un computador.

Introducción

La memoria cache es una memoria de alta velocidad y pequeño tamaño que pretende almacenar, de forma transparente al programador, aquella información que será referenciada en un futuro próximo. Las memorias cache son efectivas porque se basan en las propiedades de localidad, tanto de datos como de instrucciones, de los programas. Algunos de los parámetros fundamentales que caracterizan una memoria cache son:

- Tamaño de la cache.
- Tamaño de la línea de cache.
- Grado de asociatividad de la cache.

En esta sesión vamos a desarrollar unos programas que nos permitirán averiguar los valores de estos tres parámetros en un computador cualquiera[1].

Metodología de trabajo

La herramienta ideal para desarrollar este tipo de aplicaciones son los contadores hardware. Los contadores hardware son circuitos internos del microprocesador cuyo objetivo es permitir que el usuario tenga conocimiento de ciertos parámetros relevantes en la ejecución de programas. Los contadores hardware, una vez programados, se incrementan cada vez que se produce un determinado evento: acceso a memoria, fallos en la cache datos L1, etc. Estos contadores nos permiten averiguar cuántos eventos de un determinado tipo se producen en una porción de código. En el siguiente código se muestra cómo se obtiene esta información:

```
...
InitCounters(#Referencias, #FallosL1Datos); // Inicializar contadores
ReadCounters(&referencias1, &fallos1); // Leer contadores

// CODIGO A EVALUAR
ReadCounters(&referencias2, &fallos2); // Leer contadores
referencias = referencias2-referencias1; // Calcular referencias totales
fallos = fallos2 - fallos1; // Calcular fallos totales
m = fallos / referencias;
```

Otra forma de calcular estos parámetros es utilizar las medidas de tiempo (como las que se han hecho en las prácticas anteriores). Dado que los fallos de cache son más costosos (en tiempo) que un acierto, midiendo los tiempos de ejecución de distintos programas pueden deducirse el valor de algunos parámetros de la cache.

Sin embargo, cuando se utilizan medidas de tiempo, los resultados obtenidos suelen ser bastante imprecisos porque muchos otros eventos, y no sólo los fallos de cache, afectan al tiempo de ejecución.

En nuestro caso, para evitar las variaciones propias de los sistemas con varios procesadores, no vamos a utilizar ninguna de estas posibilidades. Vamos a utilizar un simulador, que para un programa determinado nos dirá cuántos fallos de cache se producen. Un ejemplo de cómo usar el simulador podría ser el siguiente código:

[1] La mayoría de los procesadores actuales disponen de una cache de datos y otra de instrucciones de primer nivel dentro del procesador. Nosotros estudiaremos sólo la cache de datos.

```
// 1) CODIGO A EVALUAR:    // 1) CODIGO EVALUADOR:

sum = 0;                   InitCache(137);          // Inicializar Cache #137
for (i=0; i<1000; i++)     for (i=0; i<1000; i++)
   sum = sum + v[i]           Referencia(&v[i]);    // Evaluar acceso v[i]
                           misses = Fallos();       // Obtener los fallos de cache


// 2) CODIGO A EVALUAR:    // 2) CODIGO EVALUADOR:

for (i=0; i<1000; i++)     InitCache(71);           // Inicializar Cache #71
   v[i] = v[i] + 44        for (i=0; i<1000; i++){
                              Referencia(&v[i]);    // Evaluar acceso lect v[i]
                              Referencia(&v[i]);    // Evaluar acceso escr v[i]
                           }
                           misses = Fallos();       // Obtener los fallos de cache
```

En este código, el número que se pasa como parámetro a la función InitCache indica el tipo de cache que se usará para hacer las mediciones (hay muchas caches diferentes programadas con distintos tamaños de cache, tamaños de línea, asociatividad, políticas de escritura, etc., por lo que los resultados del experimento dependerán del parámetro que se pase a esta función).

Estudio Previo

Antes de explicar en qué consiste la sesión, plantearemos el trabajo previo. Todos los puntos del trabajo previo están relacionados con la práctica a realizar.

1. Suponiendo que cada elemento de `v[i]` ocupa 1 byte, calculad cuántos fallos de cache provoca el acceso `v[i]` en los siguientes casos:

Código	Memoria Cache	stepA	stepB	stepC	stepD
`for (j=0, i=0; j<10000; j++) { sum = sum + v[i]; i = i + step; }`	Cache Directa Tamaño: 4KB Tamaño línea: 8B	step = 1	step = 4	step = 8	step = 16
`for (j=0, i=0; j<10000; j++) { sum = sum + v[i]; i = i + step; }`	Cache 2-asociativa Tamaño: 4KB Tamaño línea: 16B	step = 1	step = 4	step = 8	step = 16

2. Dado el siguiente código y suponiendo que cada elemento de `v[i]` ocupa 1 byte:

```
for (j=0, i=0; j<10000; j++) {
   sum = sum + v[i]; // cada elemento de v[i] ocupa 1 byte
   i = i + step;
}
```

   Suponiendo que la cache es directa (16KB) con líneas de 8B, dibujad una gráfica donde se represente el número de fallos que se producen (eje y) variando la variable step de 1 a 16 (eje x).

3. Suponiendo que cada elemento de `v[i]` ocupa 1 byte, calculad cuántos fallos de cache provoca el acceso `v[i]` en los siguientes casos:

Código	Memoria Cache	Valores de `limite`

`for (i=0, j=0; j<32; j++) {`
`  sum = sum + v[i];`
`  i = i + 8;`
`  if (i >= limite) i = 0;`
`}`
	Cache Directa
	Tamaño: 4 líneas
	Tamaño línea: 8B
		limite= 16, 32, 40, 48, 64 y 128
		o lo que es lo mismo
		limite=16B, 32B, 40B, 48B, 64B y 128B

`for (i=0, j=0; j<32; j++) {`
`  sum = sum + v[i];`
`  i = i + 8;`
`  if (i >= limite) i = 0;`
`}`
	Cache 2-asociativa
	Tamaño: 4 líneas
	Tamaño línea: 8B
		limite=16B, 32B, 40B, 48B, 64B y 128B

`for (i=0, j=0; j<32; j++) {`
`  sum = sum + v[i];`
`  i = i + 8;`
`  if (i >= limite) i = 0;`
`}`
	Cache 4-asociativa
	Tamaño: 4 líneas
	Tamaño línea: 8B
		limite=16B, 32B, 40B, 48B, 64B y 128B

4. Dado el siguiente código:

```
for (i=0,j=0; j<M; j++) { // M vale 1024*1000
   sum = sum+ v[i];       // cada elemento de v[i] ocupa 1 byte
   i = i + 32;
   if (i >= limite) i = 0;
}
```

   Suponiendo que la cache es directa (2KB) con líneas de 32B, dibujad una gráfica con los fallos que se producen (eje y) variando la variable `limite` de 512 a 8192 con incrementos de 512 (eje x).

5. Estudiaremos ahora la influencia de la asociatividad en el número de fallos. Dado el siguiente código:

```
for (limite = 1; limite <= 32; limite++) {
    for (i=0, j=0; j<M; j++) {  // M vale 256*1000
      if ((j % limite)==0) i=0;
      sum = sum+ v[i];          // cada elemento de v[i] ocupa 1 byte
      i=i+cache_size;
    }
}
```

   Suponiendo que la cache es de 3KB con líneas de 8 bytes, dibujad una gráfica con los fallos que se producen (eje y) respecto a la variable `limite` (eje x) en cada uno de los siguientes casos (no es necesario calcular todos los puntos):

   a) asociatividad = 4

   b) asociatividad = 6

   ¿Cuál es la relación entre el número de fallos, la variable límite y la asociatividad de la cache?

Trabajo a realizar durante la Práctica

Durante la sesión, el profesor de laboratorio os dará una hoja que deberéis rellenar. En esa hoja estará indicado el código de dos caches (será el parámetro que deberéis pasarle a la rutina `InitCache`). Para cada una de las caches deberéis averiguar:

- Tamaño de línea (rango posible de valores: entre 16 y 128 bytes).
- Tamaño de cache (rango posible de valores: entre 1 y 2048 Kbytes).
- Asociatividad (rango posible de valores: entre directa y 8-asociativa).

En las páginas siguientes se explica cómo utilizar los programas de la práctica para averiguar estos valores y se muestra el modelo de cuestionario que os entregará el profesor de laboratorio y que deberéis rellenar. Tendréis que repetir el cuestionario para 2 caches diferentes. El trabajo que habéis realizado en el previo está orientado a que entendáis cómo se puede descubrir el valor de ciertos parámetros de la cache (tamaño de línea, tamaño y asociatividad) a partir del análisis del número de fallos cuando se usan ciertos patrones de acceso a la cache.

Tamaño de la línea de la cache de datos

El primer parámetro que vamos a averiguar es el tamaño de la línea de cache. La línea de cache es la unidad de transferencia entre la memoria cache y la memoria principal (o el siguiente nivel de la jerarquía de memoria). Recordad que, cada vez que se produce un fallo de cache, se trae del nivel superior de la jerarquía el bloque de memoria que contiene el dato solicitado.

En todos los apartados de la práctica, al igual que hemos hecho en el previo, un char $V[i]$ ocupa 1 byte.

Si revisamos los 2 primeros apartados del trabajo previo, el código era similar a éste:

```
char v[N];
i = 0;
for (j = 0; j < M; j++) {
   sum = sum + v[i];
   i = i + step;
   if (i >= N) i = 0;
}
```

Había que calcular el número de fallos en la cache de datos de este programa. Vamos a suponer lo siguiente:

- El vector `v` es varias veces más grande que la cache de datos.
- Los elementos de `v` ocupan 1 byte
- `M` es un valor grande (p.e. 1000*1024)
- El tamaño de línea es 16 bytes.
- Sólo se contabilizan los accesos en lectura a `v[i]`.
- Hay que calcular los fallos teniendo en cuenta diferentes valores de la variable `step`.

Estudiando el código, podemos llegar a las siguientes conclusiones:

- Se hace un número constante de accesos a memoria: `M`.
- Como el vector `v` es mucho más grande que la memoria cache, sólo tenemos localidad temporal mientras N es inferior al tamaño de la cache. Por ejemplo, Para un valor de N suficientemente grande, con `step=1`, si empezamos con `i=0` y recorremos el vector hasta `N-1` , cuando volvamos a `i=0` volveremos a fallar porque no quedará nada de los primeros elementos del vector en la cache.
- Tenemos sin embargo localidad espacial que depende del valor de la variable `step`. La variable step determina el patrón de acceso, la distancia entre dos accesos consecutivos. Cuanto menor sea, por lo tanto, más se aprovechará la localidad espacial.

El comportamiento del algoritmo para líneas de 8 bytes se muestra de forma gráfica en las siguientes figuras:

step=1, 1 fallo y 7 aciertos por línea, M/8 fallos en total

F	-	A	-	A	-	A	-	**F**	-	A	-	A	-	A	-	**F**	-	A	-	A	-	A	-	**F**	-	A	-	A	-	A	-

step=2, 1 fallo y 3 aciertos por línea, M/4 fallos en total

F	-	-	-	A	-	-	-	**F**	-	-	-	A	-	-	-	**F**	-	-	-	A	-	-	-	**F**	-	-	-	A	-	-	-

step=4, 1 fallo y 1 acierto por línea, M/2 fallos en total

F	-	-	-	-	-	-	-	**F**	-	-	-	-	-	-	-	**F**	-	-	-	-	-	-	-	**F**	-	-	-	-	-	-	-

step=8, 1 fallo por línea, M fallos en total

En definitiva, podemos concluir que cuando el `step` sea mayor o igual que el tamaño de línea, todos los accesos serán fallos. El número de fallos irá creciendo a medida que aumenta la variable `step` mientras `step` < `tam`.

Utilizando como base el código del ejercicio teórico anterior, es fácil escribir un programa en C que permita averiguar el tamaño de línea de nuestro computador.
Ese código podría ser el siguiente:

```
// CODIGO A EVALUAR:

for (step=1; step<=12; step++) {
   i = 0;
   sum = 0;
   for (j=0; j<M; j++) {
      sum = sum + v[i];
      i = i + step;
      if (i >= tam) i = 0;
   }
}
```

```
// CODIGO EVALUADOR:

for (step=1; step<=12; step++) {
   InitCache(COD);
   i = 0;
   for (j=0; j<M; j++) {
      Referencia(&v[i]);
      i = i + step;
      if (i >= tam) i = 0;
   }
   misses = Fallos();
}
```

Este programa (incompleto) lo podéis encontrar en el fichero `LineSize.c`. La variable `COD` de la función `InitCache` identifica el tipo de cache que usaréis para hacer la práctica. El valor de la variable os lo diremos durante la clase de laboratorio. Se pide que hagáis lo siguiente:

- Completad, Compilad y Ejecutad el programa. Para compilar, debéis utilizar el siguiente comando:

  ```
  $> gcc -m32 LineSize.c Simulador.o -o LINE
  ```

  En `Simulador.o` están las rutinas que simulan el comportamiento de la cache.

- Deducid el tamaño de línea de la cache de datos que estáis simulando . Es posible que sea necesario modificar el valor máximo que puede alcanzar la variable `step`.

Tamaño de la memoria cache de datos

Un parámetro fundamental de la memoria cache es su tamaño. Vamos a diseñar un programa escrito en C que nos permita averiguar el tamaño de la memoria cache de datos.

Igual que antes, si revisamos el apartado 3 del trabajo previo, teníamos un código similar a éste:

```
char v[N];
i = 0;
step = tamaño_línea;
for (j = 0; j < M; j++) {
   sum = sum + v[i];
   i = i + step;
   if (i >= limit) i = 0;
}
```

Hay que calcular el número de fallos en la cache de datos de este programa, teniendo en cuenta lo siguiente:

- El vector `v` es varias veces más grande que la cache de datos.
- `M` es un valor muy grande (p.e. 10*1000*1024).
- Si la variable `STEP` se inicializa con el tamaño de la línea se hace un sólo acceso a cada línea, por lo que en el código sólo se producirán fallos de carga. Esto facilitará los cálculos. Por ello, debéis usar en este apartado el tamaño de línea que habéis calculado en el apartado anterior para la variable `STEP`.
- Sólo se contabilizarán los accesos en lectura a `v[i]`.

Estudiando el código, podemos llegar a las siguientes conclusiones:

- Se hace un número constante de accesos a memoria: `M`.
- El programa se comportará de forma muy diferente en función del valor de `limit`:
  - Si `limit` es menor o igual que el tamaño de la cache, estaremos accediendo repetidamente a un conjunto de líneas que cabe en la cache. Sólo tendremos los fallos iniciales de carga, y una vez se han cargado las líneas en la cache, todos los accesos serán aciertos.
  - Cuando `limit` sea un poco mayor que el tamaño de la cache se producirán algunos fallos de capacidad, ya que las primeras líneas de la cache se reescribirán y se perderá la localidad temporal cuando se acceda de nuevo a los primeros elementos del vector.
  - A partir de un cierto valor de `limit`, todos los accesos serán fallos.

Utilizando como base el código del ejercicio teórico anterior, es fácil escribir un programa en C que permita averiguar el tamaño de la cache de datos de primer nivel.
Ese código podría ser el siguiente:

```
// CODIGO A EVALUAR:

for (limit=256; limit<=512; limit+=256) {
   LimpiarCache();
   i = 0;
   for (j=0; j<M; j++) {
      sum = sum + v[i];
      i=i+LINE_SIZE;
      if (i >= limit) i=0;
   }
}
```

```
// CODIGO EVALUADOR:

for (limit=256; limit<=512; limit+=256) {
   InitCache(COD);
   i = 0;
   for (j=0; j<M; j++) {
      Referencia(&v[i]);
       i=i+LINE_SIZE;
       if (i >= limit) i=0;
   }
   misses = Fallos();
}
```

Este programa lo podéis encontrar en el fichero `CacheSize.c`.
Se pide que hagáis lo siguiente:

- Actualizad en el código la constante LINE_SIZE con el tamaño de línea que habéis encontrado en el apartado anterior
- Compilad y Ejecutad el programa. Para compilar debéis utilizar el siguiente comando:

  ```
  $> gcc -m32 CacheSize.c Simulador.o -o CACHE
  ```

- Deducid el tamaño de la cache de datos. Una forma sencilla de hacerlo es calcular el tamaño de la cache en función de la cantidad de líneas que contiene. Para realizar este apartado es posible que sea necesario modificar el bucle:

  ```
  for (limite=256; limite<= 512; limite+=256)
  ```

  Para encontrar el tamaño de la cache es conveniente que el límite inferior del bucle (for limite=256;) sea múltiplo del incremento (limite+=256).

  Modificad estos valores teniendo en cuenta los posibles tamaños que puede tomar la memoria cache.

Asociatividad de la memoria cache de datos

Finalmente, una vez conocidos el tamaño de línea y el tamaño de cache de nuestra memoria de datos, trataremos de descubrir su asociatividad. En el ejercicio previo habéis analizado los fallos de una cache de 3 KB con líneas de 8 Bytes en función de su asociatividad. Para ello habéis usado el siguiente código:

```
for (limite = 1; limite <= 2; limite++) {
    for (i=0, j=0; j<M; j++) {  // M vale 256*1000
      if ((j % limite)==0) i=0;
      sum = sum+ v[i];          // cada elemento de v[i] ocupa 1 byte
      i=i+CACHE_SIZE;
    }
}
```

Podemos ver que los accesos al vector son múltiplo del tamaño de la cache. Por lo tanto, todos los accesos irán a parar al conjunto 0 de la cache. Si la cache es directa, los accesos se machacarán unos a otros y todos los accesos serán fallos. Si la cache es 2-asociativa y límite vale 2, los dos primeros accesos serán fallos de carga, pero el resto serán todo aciertos. Sin embargo, si límite vale 3 o más se producirán fallos de conflicto, y todos los accesos serán fallos. Partiendo de esta idea, podemos diseñar un código que calcule la asociatividad de una cache si conocemos su tamaño.

```
// CODIGO A EVALUAR:

for (limite = 1; limite <= 2; limite++) {
    LimpiarCache();
    for (i=0, j=0; j<M; j++) {
      if ((j % limite)==0) i=0;
      sum = sum+ v[i];
      i=i+CACHE_SIZE;
    }
}
```

```
// CODIGO EVALUADOR:

  for (limite = 1; limite <= 2; limite++) {
      InitCache(COD);
      for (i=0, j=0; j<M; j++) {
         if ((j % limite)==0) i=0;
         Referencia(&v[i]);     // acceso a v[i]
         i=i+CACHE_SIZE;
      }
      refs = Referencias();
      misses = Fallos();
   }
```

Este programa lo podéis encontrar en el fichero `Associativity.c`.
Se pide que hagáis lo siguiente:

- Actualizad en el código la variable CACHE_SIZE con el tamaño de cache que habéis encontrado en el apartado anterior
- Completad, Compilad y Ejecutad el programa. Atención!, es posible que haya que cambiar los parámetros del bucle. Para compilar debéis utilizar el siguiente comando:

  ```
  $> gcc -m32 Associativity.c Simulador.o -o ASSOC
  ```

- Deducid la asociatividad de la cache de datos.

Nombre:_____ Grupo:_____

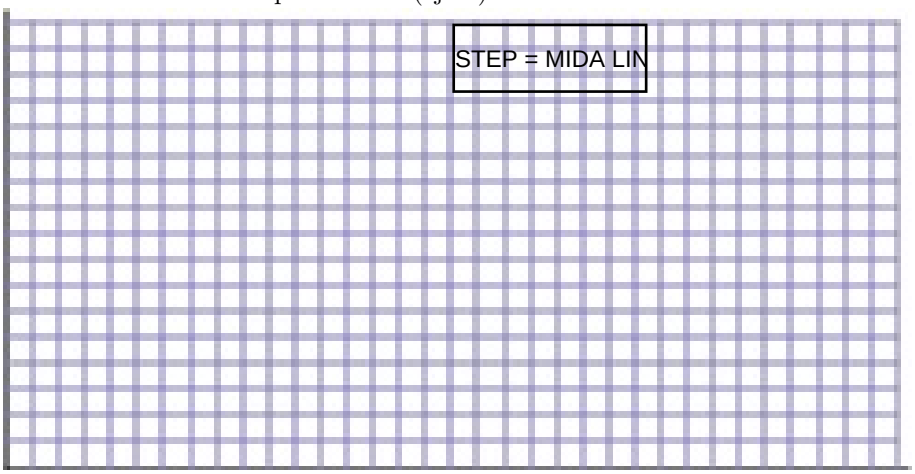
Nombre:_____

Hoja de respuesta al Estudio Previo

1. Fallos del acceso a `v[i]`:

Código	Memoria Cache	stepA	stepB	stepC	stepD
`for (j=0, i=0; j<10000; j++) { sum = sum + v[i]; i = i + step; }`	Cache Directa Tamaño: 4KB Tamaño línea: 8B	10k/8=	10k/2=	10k	10k
`for (j=0, i=0; j<10000; j++) { sum = sum + v[i]; i = i + step; }`	Cache 2-asociativa Tamaño: 4KB Tamaño línea: 16B	10k/16	10k/4=2	10k/2=5	10k

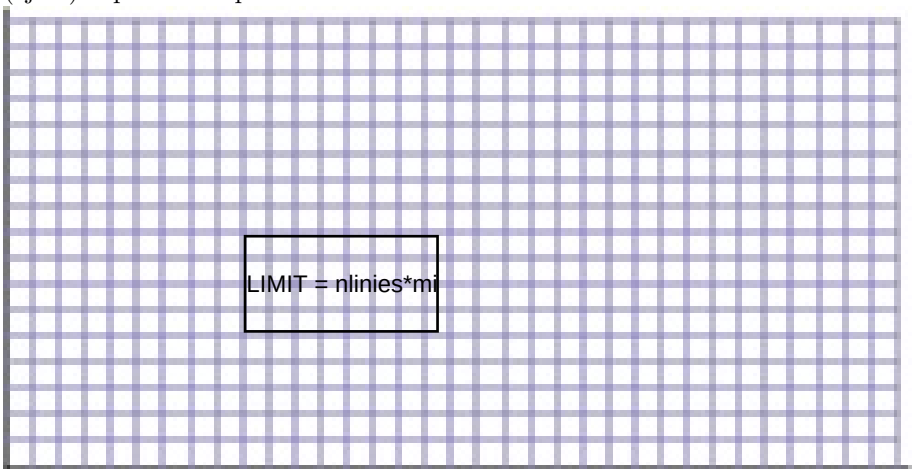
2. Dibujad una gráfica donde se represente el número de fallos que se producen (eje y) variando la variable step de 1 a 16 (eje x):



3. Fallos de cache que provoca el acceso `v[i]` en los siguientes casos:

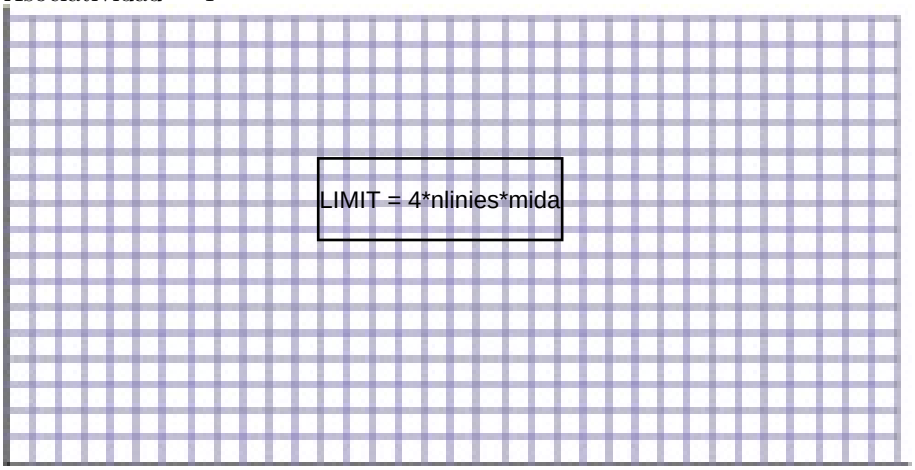
Código	Memoria Cache	Valores de limite 16B	32B	40B	48B	64B	128B
`for (i=0, j=0; j<32; j++) { sum = sum + v[i]; i = i + 8; if (i >= limite) i = 0; }`	Cache Directa Tamaño: 4 líneas Tamaño línea: 8B	2	4	32	32	32	32
`for (i=0, j=0; j<32; j++) { sum = sum + v[i]; i = i + 8; if (i >= limite) i = 0; }`	Cache 2-asociativa Tamaño: 4 líneas Tamaño línea: 8B	2	4	5	6	32	32
`for (i=0, j=0; j<32; j++) { sum = sum + v[i]; i = i + 8; if (i >= limite) i = 0; }`	Cache 4-asociativa Tamaño: 4 líneas Tamaño línea: 8B	2	4	5	6	8	32

4. Dibujad una gráfica con los fallos que se producen (eje y) repecto a la variable `limite` (eje x) suponiendo que la cache es directa.

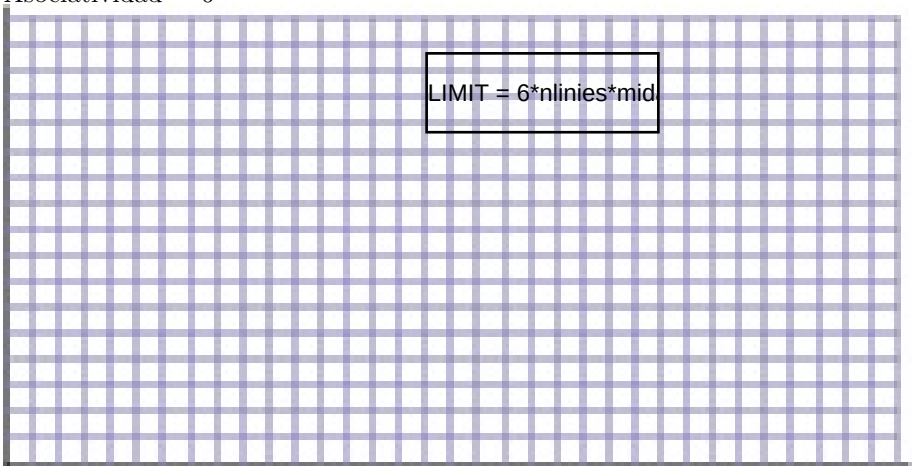


5. Dibujad una gráfica con los fallos que se producen (eje y) repecto a la variable `limite` (eje x) suponiendo que el grado de asociatividad de la cache es:

Asociatividad = 4



Asociatividad = 6



¿Cuál es la relación entre el número de fallos, la variable límite y la asociatividad de la cache?

+limit -> +fallos, -limit -> -fallos+associ -> -fallos, -associ -> +fallos

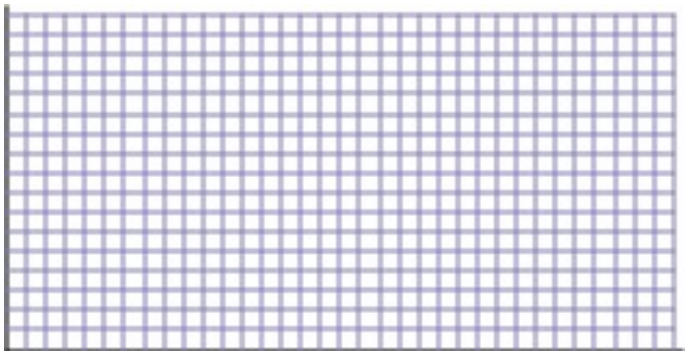
Nombre:_____ Grupo:_____

Nombre:_____

Hoja de respuestas de la práctica

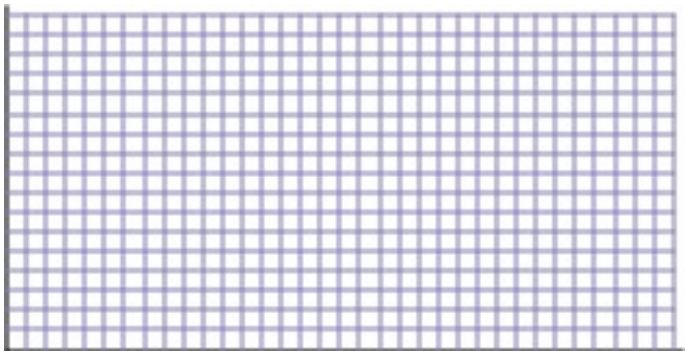
Código Cache:

1. Rellenad la siguiente gráfica donde se represente el número de fallos que se producen (eje y) en función de la variable step (eje x). Esta gráfica es similar a la del apartado 2) del trabajo previo.



2. Tamaño de línea (Justificad la respuesta a partir de la gráfica anterior):

3. Rellenad la siguiente gráfica donde se represente el número de fallos que se producen (eje y) en función de la variable limit (eje x). Esta gráfica es similar a la del apartado 4) del trabajo previo.



4. Tamaño de cache (Justificad la respuesta a partir de la gráfica anterior):

5. Asociatividad (Revisad el apartado 5) del trabajo previo. Justificad la respuesta):

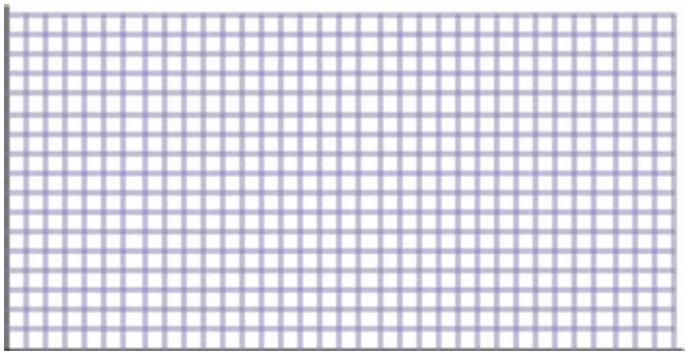
Nombre:_____ Grupo:_____

Nombre:_____

Hoja de respuestas de la práctica

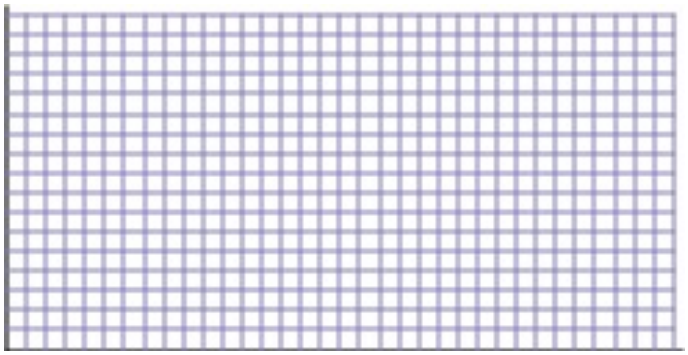
Código Cache:

1. Rellenad la siguiente gráfica donde se represente el número de fallos que se producen (eje y) en función de la variable step (eje x). Esta gráfica es similar a la del apartado 2) del trabajo previo.



2. Tamaño de línea (Justificad la respuesta a partir de la gráfica anterior):

3. Rellenad la siguiente gráfica donde se represente el número de fallos que se producen (eje y) en función de la variable limit (eje x). Esta gráfica es similar a la del apartado 4) del trabajo previo.



4. Tamaño de cache (Justificad la respuesta a partir de la gráfica anterior):

5. Asociatividad (Revisad el apartado 5) del trabajo previo. Justificad la respuesta):