Tema-3.pdf

roro_pocha

Arquitectura de Computadores

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Tema 3: Arquitectura con paralelismo a nivel de thread (TLP)

Lección 7. Arquitecturas TLP

Clasificación de arquitecturas con TLP explícito y una instancia del SO

Hay 3 tipos de paralelismo:

Tipo de Paralelismo	¿Qué ejecuta en paralelo?
DLP (Data-Level Parallelism)	Operaciones sobre datos de **una misma instrucción**
ILP (Instruction-Level Parallelism)	**Varias instrucciones** de un mismo thread
TLP (Thread-Level Parallelism)	**Varios hilos (threads)** en paralelo

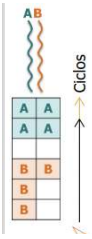
En DLP e ILP se trabaja con un solo hilo, optimizando su rendimiento.
En TLP se introducen **varios hilos**, cada uno con su propio flujo de instrucciones.

Además se distingue entre:

- **TLP con una instancia del SO:** múltiples hilos corriendo sobre un mismo sistema operativo.
- **TLP con múltiples instancias del SO:** threads ejecutándose en múltiples computadores.

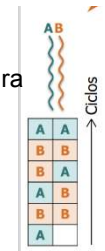
Cores Multithread

La estructura ILP tiene ciertas limitaciones que producen pérdidas verticales y horizontales, ya que sólo tiene un hilo para extraer instrucciones.
La solución es implementar cores multithread (TLP) que permitan cambiar de hilo cuando uno se bloquea o incluso emitir instrucciones de varios hilos en paralelo.
Para ello realizaremos distintos cambios en la arquitectura ILP.

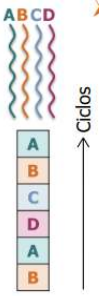


Clasificación

← **Temporal Multithreading (TMT):** Un solo hilo emite instrucciones por ciclo. El hardware decide cuándo cambiar de hilo. Ejecuta un hilo a la vez, pero cambia frecuentemente para evitar tiempos muertos.
Simultaneous Multithreading (SMT) →: Permite que varios hilos emitan instrucciones en el mismo ciclo. Se ejecutan en paralelo dentro del mismo core aprovechando su arquitectura superescalar.

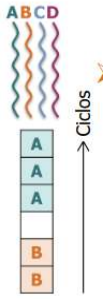


Clasificación de cores con TMT



← **Grano fino (FGMT):** El hardware cambia de hilo en cada ciclo. Lo hace sin coste (0 ciclos perdidos) porque sigue siendo un turno rotatorio (round-robin), responde a eventos de latencia. El objetivo es mantener el pipeline ocupado todo el tiempo, aunque ningún hilo se ejecute muy rápido.

Grano grueso (CGMT) →: El hardware ejecuta un hilo durante varios ciclos seguidos. Cambia de hijo solo cuando ocurre un evento de alta latencia (ej. fallo de caché) o después de un intervalo determinado. Hay un pequeño coste de cambio de contexto (puede ser de 0 a varios ciclos).



Estrategias de conmutación con CGMT

Estática:

La decisión de cambiar de hilo está predefinida:

- Por instrucciones **explícitas**, añadidas en el repertorio de instrucciones, que indican "cambia de hilo".
- O por instrucciones **implícitas** que provocan cambio, como saltos, cargas o accesos a memoria.

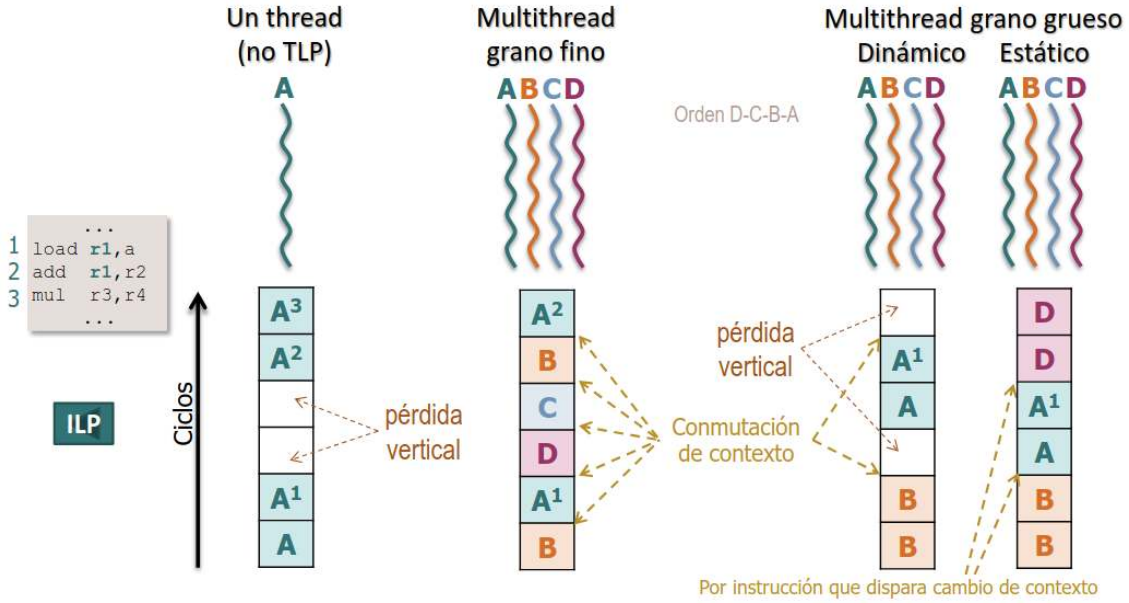
Ventaja: Cambio de contexto con **muy bajo coste** (0 o 1 ciclo).
Desventaja: Puede hacer **cambios innecesarios** si el hilo no está bloqueado.

Dinámica:

El hardware detecta eventos en tiempo real y decide si conviene cambiar de hilo.
Ventaja: Evita cambios de contexto innecesarios → **más eficiente.**
Desventaja: Requiere **más lógica de control** y puede tener **mayor sobrecarga.**

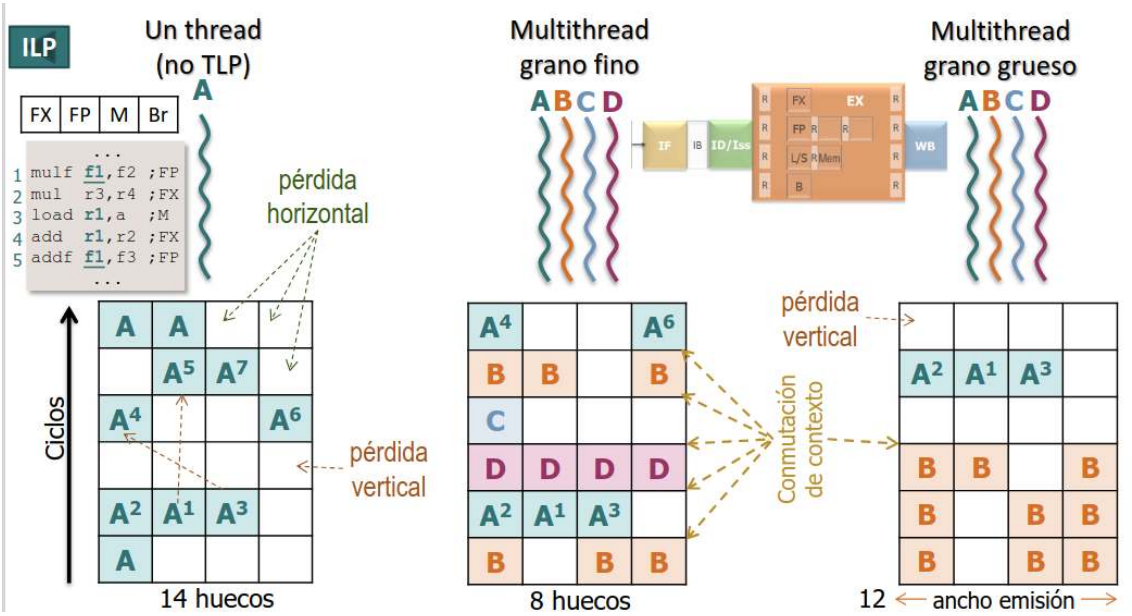
Alternativas en un core escalar segmentado

Solo se emite una instrucción por ciclo.

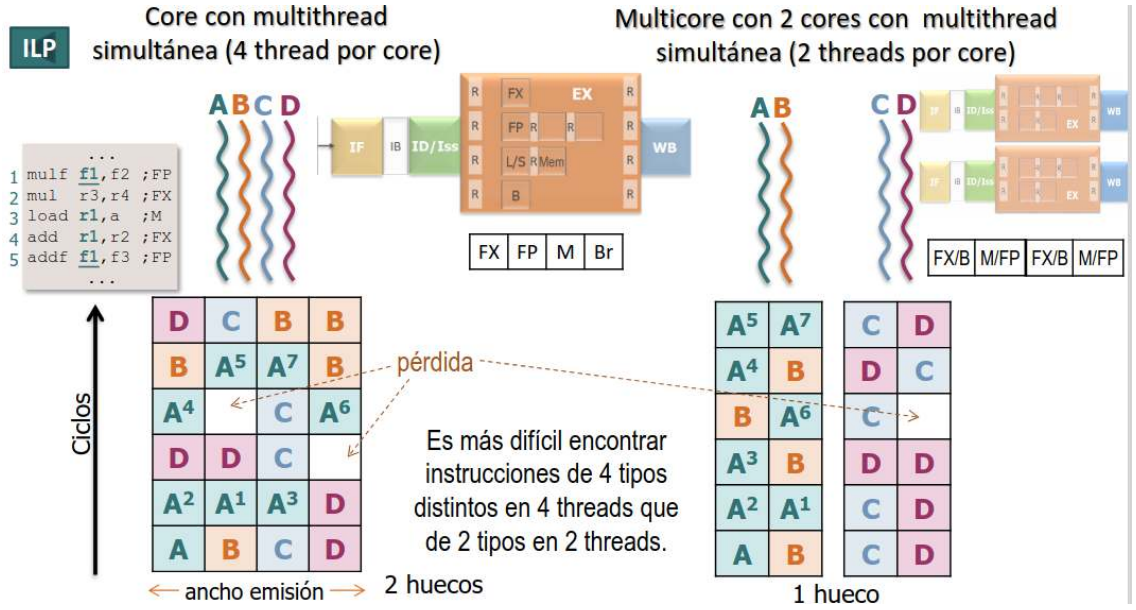


Alternativas en un core con emisión múltiple de instrucciones de un thread

Se emiten más de una instrucción cada ciclo de reloj.



Core multithread simultánea y multicore



En core no se puede ejecutar instrucciones del mismo tipo (FX con FX, FP con Fp…) en el mismo ciclo.

En multicore tampoco se puede M con FP ni FX con B en el mismo ciclo y mismo bloque. Si son distintos bloques sí se puede.

Lección 8. Coherencia del sistema de memoria

Concepto de coherencia en el sistema de memoria. Situaciones de incoherencia

Se produce incoherencia cuando las copias de un mismo dato en diferentes cachés no están sincronizadas, es decir, tienen valores distintos.
Tipos de fallos de coherencia:

- **Caché-caché:** dos cachés tienen copias distintas de un mismo dato.
- **Caché-memoria principal:** la caché tiene un valor diferente del almacenado en RAM.

Métodos de actualización de memoria principal implementados en cachés

Escritura inmediata (Write-through): Cada vez que se escribe un dato en la caché, también se actualiza inmediatamente la memoria principal. Así siempre la memoria está actualizada pero consume muchos recursos, por lo que produce menor rendimiento.

Posescritura (write-back): Se escribe en la caché, pero la memoria principal solo se actualiza cuando la necesite. Es más eficiente pero se corre riesgo de incoherencia si no se controla.

Alternativas para propagar una escritura en protocolos de coherencia de caché

Escritura con actualización (write-update): Cuando se escribe en una caché, se modifica en el resto de cachés. Esto genera mucho tráfico.

Escritura con invalidación (write-invalidate): Cuando se escribe en una caché, el resto de copias se invalidan, y solo se actualizan cuando estas las necesiten. No quita el tráfico pero sí lo disminuye.

Protocolos de mantenimiento de coherencia: clasificación y diseño

Protocolos de espionaje (snoopy): El bus es compartido por todos los procesadores. Cuando uno hace una lectura o escritura, lo anuncia en el bus. Todas las cachés lo vigilan. Si otra caché tiene una copia del dato, puede invalidarla, responder con su copia o actualizarla, según el protocolo.
Protocolos basados en directorios: Se basa en mantener un registro de qué procesadores tienen copias de cada bloque de memoria y en qué estado están esas copias.
Esquemas jerárquicos: Si tengo varios niveles puedo tener distintos protocolos en cada nivel.

Protocolo MSI de espionaje

Estados de un bloque en caché:

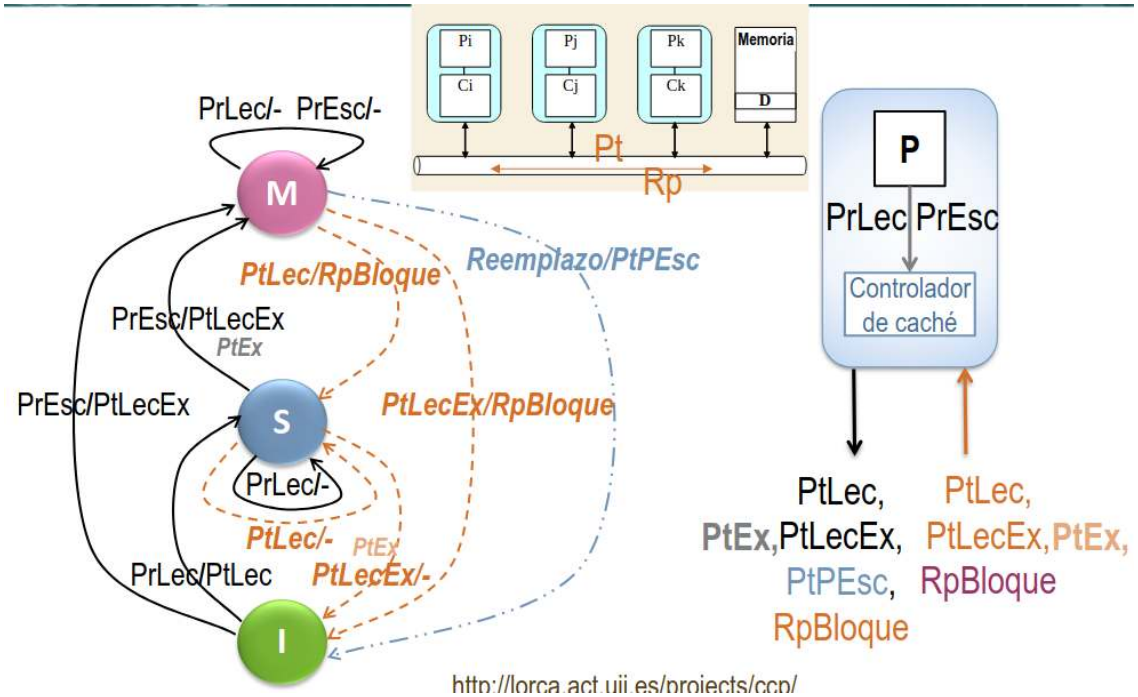
- **Modificado (M):** es la única copia del bloque válida en todo el sistema.
- **Compartido (C,S):** está válido, también válido en memoria y puede que haya copia válida en otras cachés.
- **Inválido (I):** se ha invalidado o no está físicamente.

Estados de un bloque en memoria:

- **Válido:** puede haber copia válida en una o varias cachés.
- **Inválido:** habŕa copia válida en una caché.

Tipos de transferencias generadas por un nodo con caché:

- **PtLec** (Petición de lectura): Se genera cuando hay un fallo de caché en lectura (PrLec). Se solicita el bloque a otros nodos o a memoria.
- **PtLecEx** (Petición de lectura exclusiva): Se produce cuando el procesador quiere escribir en un bloque que está en estado compartido o inválido.
- **PtPEsc** (Petición de posescritura): Cuando un bloque en estado Modificado (M) se va a reemplazar (por política de caché), se debe escribir en memoria o enviar a otro nodo.
- **RpBloque** (Respuesta con bloque): Se envía un bloque modificado como respuesta a una petición de lectura exclusiva



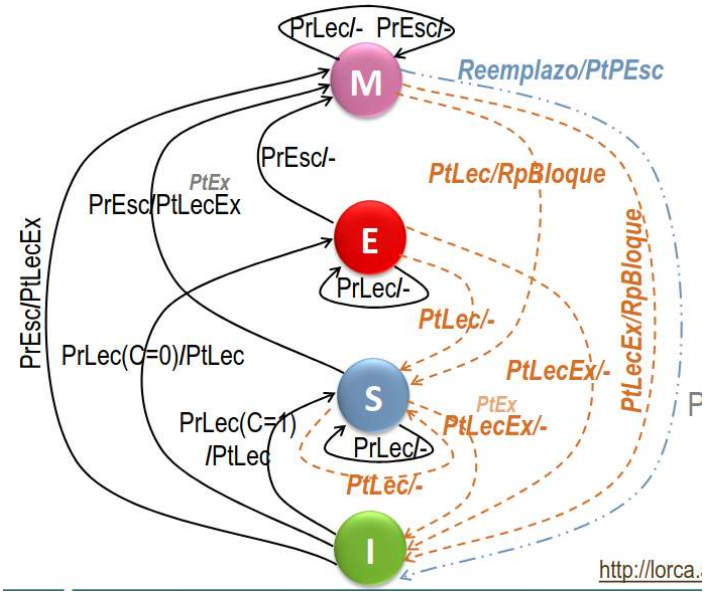
Protocolo MESI de espionaje

Se añade un nuevo estado de bloque en caché:
Exclusivo (E): Es la única copia del bloque válida en cachés, la memoria también está actualizada.

¿DÍA DE CLASES infinitas?

WRIGLEY'S Orbit

masca y fluye



Concepto de coherencia en el sistema de memoria. Requisitos para evitar problemas por incoherencia.

Puede haber distintos tiempos de propagación dependiendo de que nodo a que nodo se manda. Por lo que una actualización puede llegar antes de la que se quería. Esto produce incoherencia.

Requisitos del SM para evitar problemas por incoherencia

Propagar las escrituras en una dirección. De esta manera todos los procesadores verán la misma versión del dato. Para conseguir mayor escalabilidad, se debería enviar paquetes de actualización/invalidación sólo a cachés (nodos) con copia del bloque. Para ello se debe mantener en un directorio, para cada bloque, los nodos con copia del mismo.

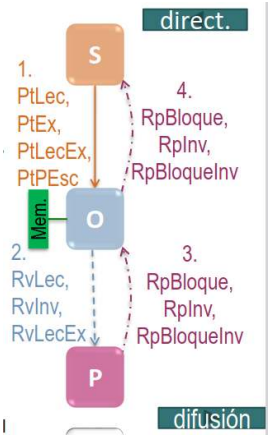
Serializar las escrituras en una dirección. Todos los procesadores deben ver el mismo orden global. Se manda la petición de escritura a un nodo home o nodo origen y desde él se mandan las copias en orden de llegada.

Alternativas para implementar el directorio

Centralizado: Compartido por todos los nodos. Contiene información de los bloques de todos los módulos de memoria.

Distribuido: El más común. Se divide el directorio entre los nodos. Típicamente el directorio de un nodo contiene información de los bloques de sus módulos de memoria.

Protocolo MSI basado en directorios con o sin difusión

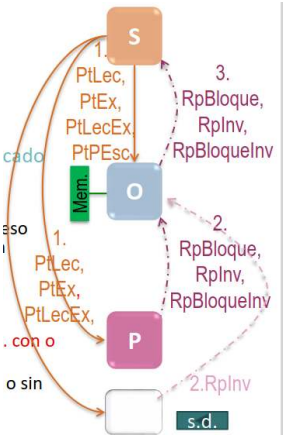


← Sin difusión

Tipos de nodos: solicitante (S), origen (O), propietario (P).
El nodo solicitante manda una petición al nodo origen y el nodo origen manda esa petición al resto de nodos, estos le responden y el nodo origen manda la respuesta al nodo solicitante.

Con difusión →

Ahora el nodo solicitante envía la petición a todos los nodos, y estos mandan la respuesta al nodo origen, que a su vez manda esta respuesta al solicitante.



Con difusión VS sin difusión

La latencia (tiempo de respuesta) media de sin difusión podría ser peor (compartición de datos modificable).
Mientras que el ancho de banda (productividad) medio, podría ser peor con difusión (poca compartición de datos modificables).

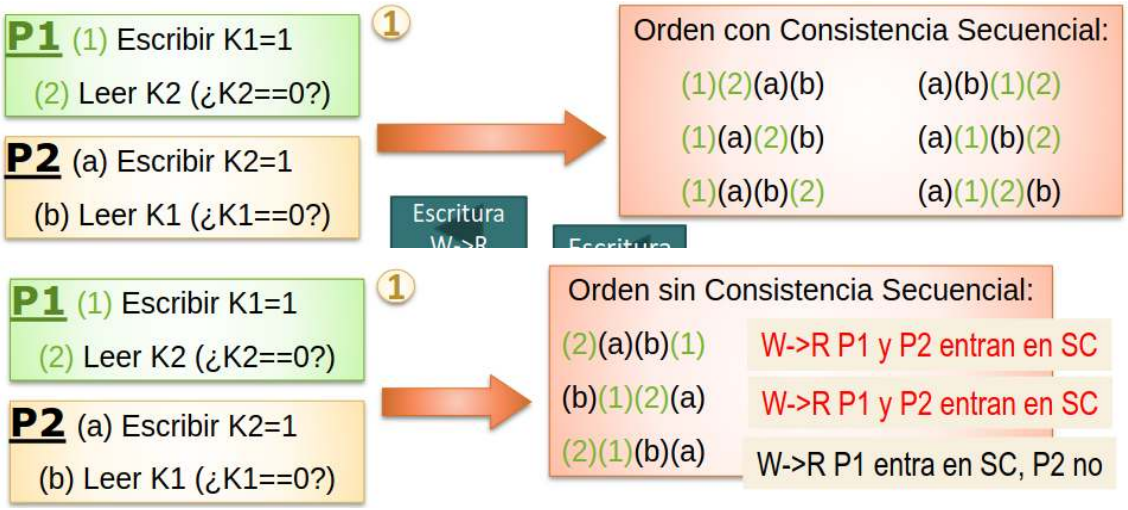
Lección 9. Consistencia del sistema de memoria

Concepto de consistencia de memoria

Especifica el orden en el cual las operaciones de memoria deben parecer haberse realizado.

Consistencia secuencial (SC)

Todas las operaciones de un flujo de instrucciones deben parecer haberse ejecutado en el orden descrito por el programador.
Con SC puede que las instrucciones de distintos programas se entrelacen pero el orden interno del programa se mantiene.



Modelos de consistencia relajados

No cumplen los requisitos para garantizar SC que relajan.

Modelo	Orden del programa relajado W→R	W→W	R→RW	Orden global Lec. anticipada propia	de otro	Instrucciones para garantizar los órdenes relajados por el modelo
Sparc-TSO, x86-TSO	Si			Si		l-m-e (instruc. lectura-modificación-escritura atómica)
ARMv7	Si	Si	Si	Si	Si	DMB (*Data Memory Barrier*)

Orden del programa: No cumple con el orden en los casos que pone Sí.
Orden global de otro: Un tercer procesador puede ver la actualización de otro antes que un 2º.
DMB de ARMv7 es una barrera en la que la parte de arriba de la barrera no puede cambiar el orden con la parte de abajo.

Lección 10. Sincronización

Comunicación en multiprocesadores y necesidad de usar código de sincronización

La sincronización en multiprocesadores es necesaria porque hay recursos compartidos. Así evitamos condiciones de carrera, protegemos las secciones críticas, mantenemos la coherencia y consistencia de memoria y coordinamos tareas concurrentes.

Cerrojos

Cerrojos simples

Permiten sincronizar mediante dos operaciones:

- **Cierre del cerrojo o lock(k)**: intenta adquirir el derecho a acceder a una nueva sección crítica. Mientras uno está dentro de la SC, el resto esperan a que termine y pasa el siguiente.
- **Apertura del cerrojo o unlock(k):** el que estaba dentro de la SC sale y deja pasar a la siguiente hebra.

Cerrojos con etiqueta

Se le asigna un número a una hebra. Luego espera su turno, solo entra cuando el turno es el mismo número que el suyo. Al salir de la SC incrementa el turno para la siguiente hebra.

Barreras

Una barrera es un punto del programa donde todos los hilos deben esperar hasta que todos hayan llegado antes de continuar.
Se usa para sincronizar fases y se implementa mediante variables compartidas.

- count: número de hilos que han llegado
- flag: indica si todos han llegado

Apoyo hardware a primitivas software

Test&Set (x)

```
Test&Set (x) {
   temp = x ;
   x = 1 ;
   return (temp) ;
}
/* x compartida */
```

Fetch&Oper(x,a)

```
Fetch&Add(x,a) {
    temp = x ;
    x = x + a ;
    return (temp);
}/* x compartida,
a local */
```

Compare&Swap(a,b,x)

```
Compare&Swap(a,b,x){
    if (a==x) {
        temp=x ;
        x=b; b=temp ; }
}/* x compartida,
   a y b  locales */
```