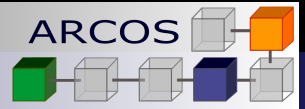
Modelos de consistencia de memoria

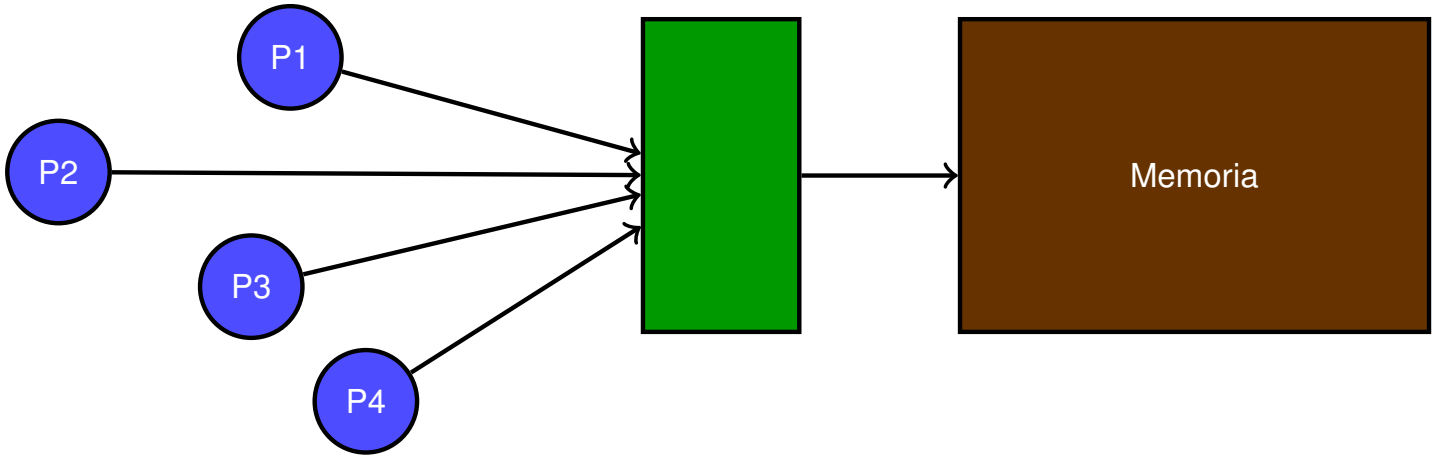
Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid



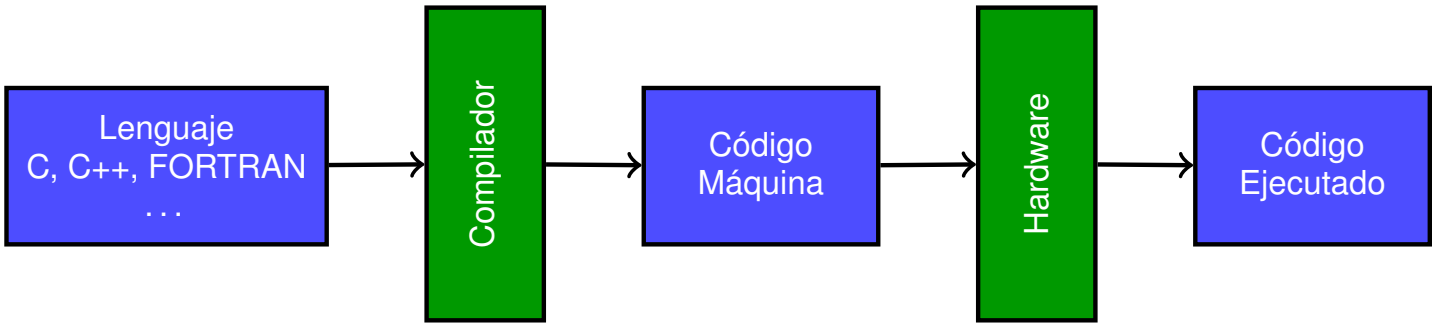
Consistencia de memoria



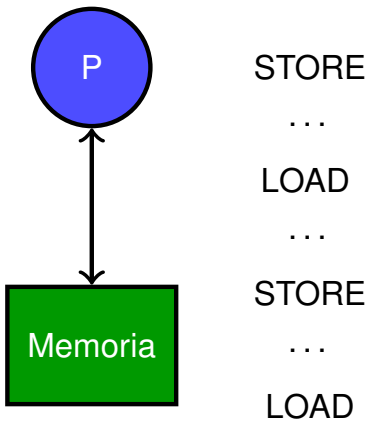
- **Modelo de consistencia de memoria**:
  - Conjunto de reglas que define como procesa el **sistema de memoria** operaciones de memoria de **múltiples procesadores**.
  - **Contrato** entre el programador y el sistema.
  - Determina qué **optimizaciones son válidas** sobre programas correctos.

Modelo de memoria

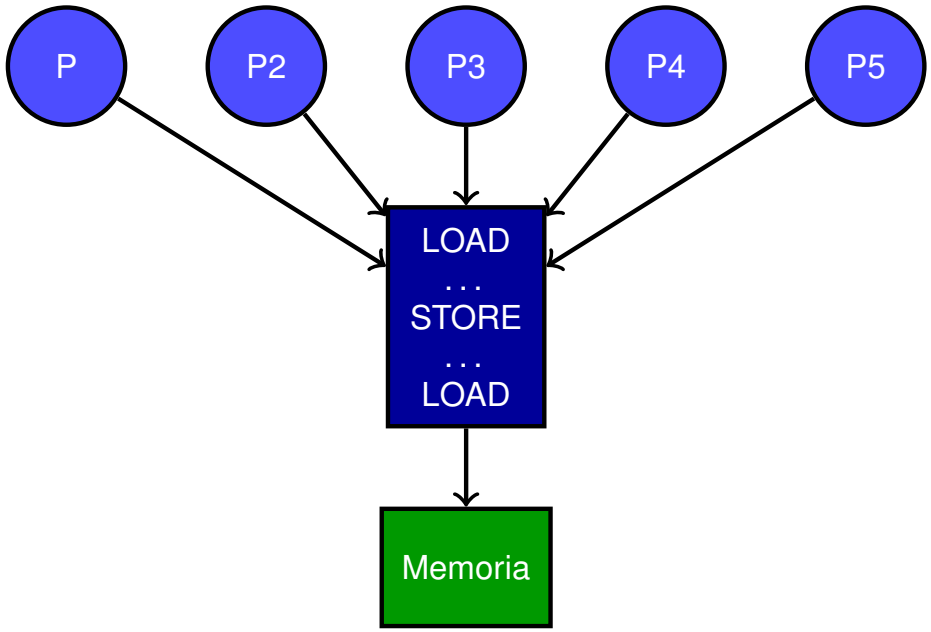
- Interfaz entre el programa y sus transformadores.
  - Define que valores puede devolver una lectura.
- El modelo de memoria del lenguaje tiene implicaciones para el hardware.



Modelo de memoria monoprocesador



- **Modelo de comportamiento de memoria**:
  - Las operaciones de memoria ocurren en **orden de programa**.
    - Una lectura devuelve el valor de la última escritura en orden de programa.
- Semántica definida por **orden de programa secuencial**:
  - Razonamiento **simple** pero **restringido**.
    - Resolver **dependencias de datos y control**.
  - Las operaciones **independientes** pueden ejecutarse en **paralelo**.
  - Las optimizaciones **preservan la semántica**.



Un sistema multiprocesador es **secuencialmente consistente** si el resultado de cualquier ejecución es el mismo que se obtendría si las operaciones de todos los procesadores se ejecutase en algún orden secuencial, y las operaciones de cada procesador individual aparecen en esa secuencia en el orden indicado por el programa.

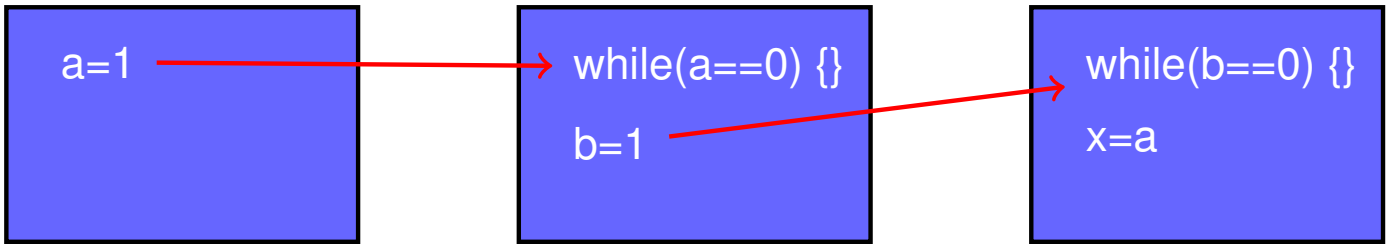
Leslie Lamport, 1979

Restricciones de la consistencia secuencial

- **Orden de programa**.
  - Las operaciones de memoria de un programa deben hacerse visibles a **todos los procesos** en el **orden de programa**.

- **Atomicidad**.
  - El orden total de ejecución entre procesos debe ser **consistente** requiriendo que todas las operaciones sean **atómicas**.
    - Nada que un procesador haga después de que haya visto el nuevo valor de una escritura se hace visible a otros procesos antes de que hayan visto el valor de esa escritura.

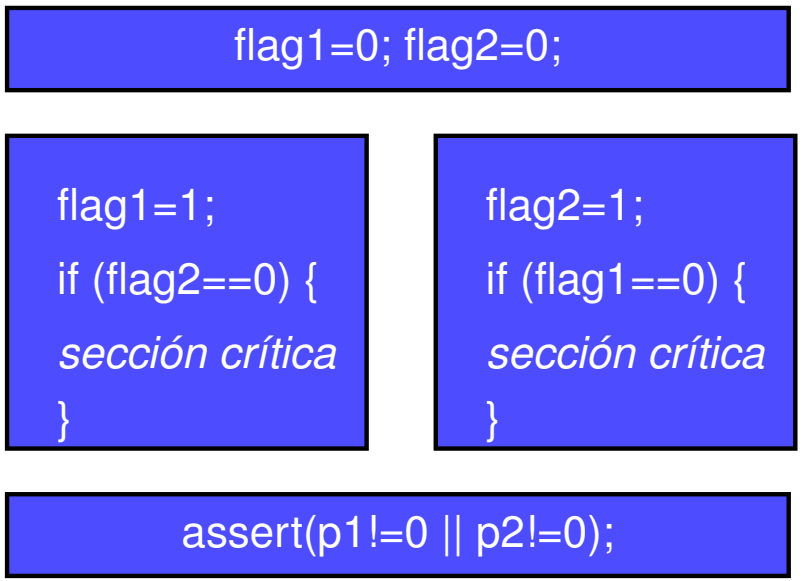
Atomicidad



- **Escrituras no atómicas**:
  - La escritura en **b** podría adelantar al bucle **while** y la lectura de **a** adelantaría la escritura.
    - **X=0**.
- **Escrituras atómicas**:
  - Se preserva la **consistencia secuencial**.

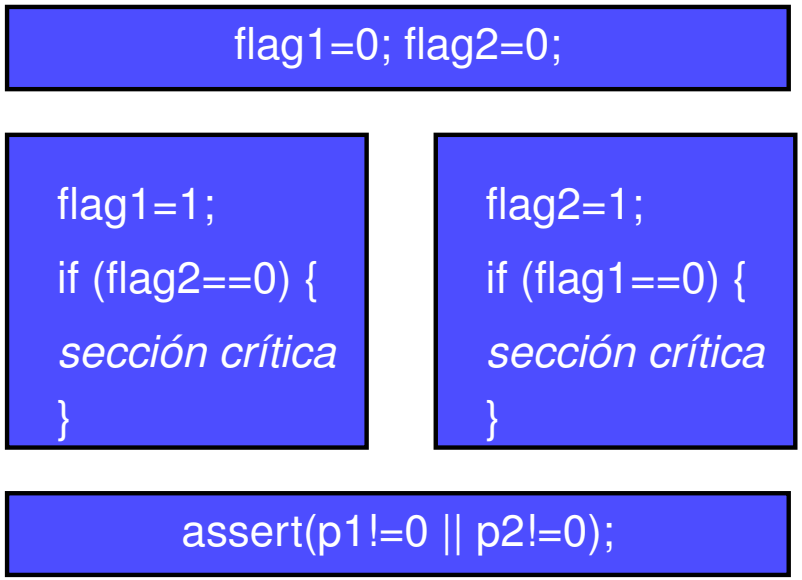
- La **consistencia secuencial** **restringe** todas las operaciones de memoria:
  - Write $\rightarrow$ Read.
  - Write $\rightarrow$ Write.
  - Read $\rightarrow$ Read, Read $\rightarrow$ Write.
- **Modelo simple** para razonar sobre programas paralelos.
- Pero, reordenaciones simples para monoprocesador pueden **violar** el modelo de **consistencia secuencial**:
  - **Reordenación de hardware** para mejora del rendimiento.
    - Write buffers, escrituras solapadas, ...
  - **Optimizaciones de compilador** aplican transformaciones que reordenan operaciones de memoria.
    - Remplazo de escalares, asignación de registros, planificación de instrucciones, ...
  - **Transformaciones** por programadores o **herramientas de refactoring** también modifican la semántica del programa.

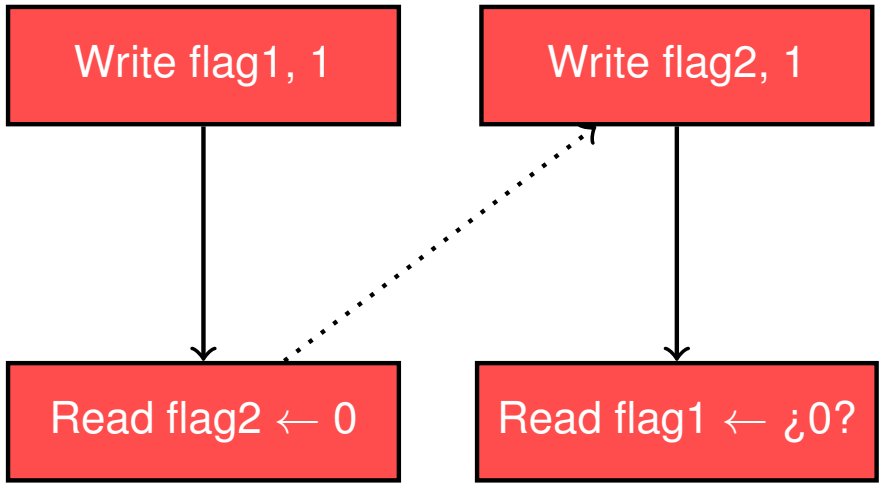
Violación de consistencia secuencial



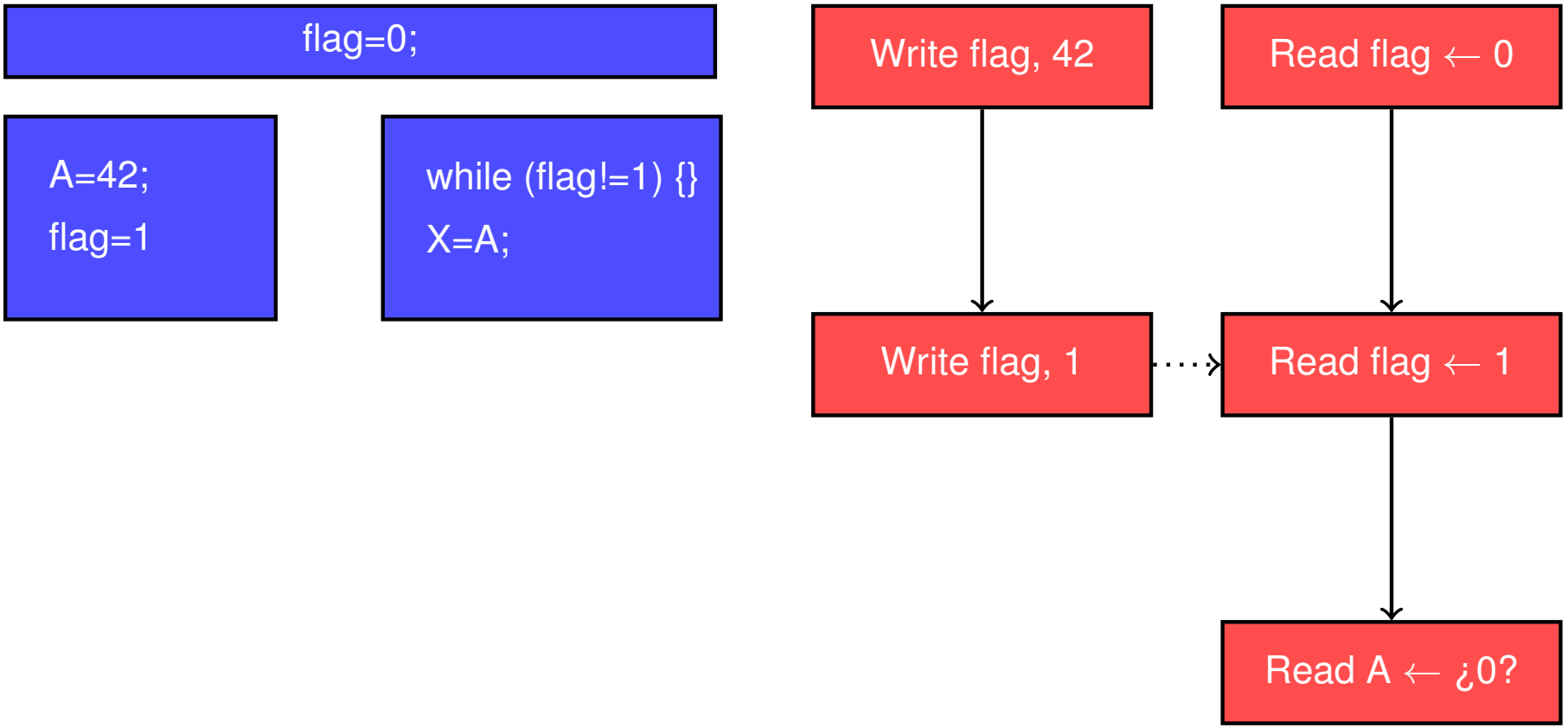
- Si las cachés usan búfer de escritura:
  - Escrituras se **retrasan** en búfer.
  - Lecturas **obtienen el valor antiguo**.
  - Se **invalida** el **algoritmo de Dekker**.
    - El **algoritmo de Dekker** es la primera solución conocida al problema de la exclusión mutua.

Orden de programa





Orden de programa



Condiciones para la consistencia secuencial

- **Condiciones suficientes**:
  - Cada proceso **emite las operaciones** de memoria en orden de programa.
  - Después de la **emisión de una escritura**, el proceso de emisión **espera a que se complete** la escritura **antes de emitir** otra operación.
  - Después de **emitir una lectura**, el proceso que la emitió **espera a que se complete** la lectura y a que la escritura del valor que se está leyendo se **complete**.
    - Esperar la propagación de escrituras a todos los procesos.

- **Son condiciones muy exigentes.**
  - Puede haber condiciones necesarias menos exigentes.

Optimizaciones

- Modelos que relajan el orden de ejecución de programa.
  - W $\rightarrow$ R.
  - W $\rightarrow$ W.
  - R $\rightarrow$ W, W $\rightarrow$ W.

Reordenaciones

Procesador	R → R	R → W	W → R	W → W
Alpha	✓	✓	✓	✓
PA-RISC	✓	✓	✓	✓
POWER	✓	✓	✓	✓
SPARC				✓
x86				✓
AMD64				✓
IA64	✓	✓	✓	✓
zSeries				✓

Lecturas adelantas a escrituras (W→R)

- Una **lectura puede ejecutarse antes** que una **escritura** anterior.

- Típico en sistemas con **búfer de escritura**.
  - Comprobación de consistencia con búfer.
  - Permiten lectura de búfer.

Otros modelos

- $R \rightarrow W$, $W \rightarrow R$.
  - Permiten que las **escrituras** **puedan llegar a memoria** fuera de orden de programa.

- $R \rightarrow W$, $W \rightarrow R$, $R \rightarrow R$, $W \rightarrow W$.
  - Solamente se evitan dependencias de datos y control dentro del procesador.
  - **Alternativas**:
    - Consistencia débil.
    - Consistencia de liberación.

Ordenamiento débil

- Divide las operaciones a memoria en **operaciones de datos** y **operaciones de sincronización**.
- Las **operaciones de sincronización** actúan como una **barrera**.
  1. Todas las operaciones de datos previas en orden de programa a una sincronización deben completarse antes de ejecutar la sincronización.
  2. Todas las operaciones de datos posteriores en orden de programa a una sincronización deben esperar a que se complete la sincronización.
  3. Las sincronizaciones se realizan en orden de programa.
- Implementación hardware de la **barrera**.
  - **Procesador mantiene un contador**:
    - **Emisión** de operación de datos $\Rightarrow$ **incremento**.
    - Operación de datos **completada** $\Rightarrow$ **decremento**.

Consistencia de adquisición/liberación

- **Más relajada** que la consistencia débil.
- **Accesos de sincronización** divididos en:
  - **Acquire** → Adquisición.
  - **Relase** → Liberación.

- **Semántica**:
  - **Acquire**
    - Debe completarse antes que todos los accesos a memoria subsiguientes.
  - **Release**
    - Deben completarse todos los accesos a memoria previos.
    - Accesos memoria posteriores **SI** pueden iniciarse.
    - Operaciones que siguen a **release** y que deben esperar se deben proteger con un **acquire**.

4 Caso de uso: Intel

- Modelo de consistencia
- Ejemplos
- Efectos del modelo

Consistencia de memoria en Intel

- Hasta el año 2005 Intel no había clarificado completamente su **modelo de consistencia de memoria**.
  - Complejidad para formalización del modelo.
  - Problemas para implementaciones de lenguajes (Java, C++, . . .).
- Actualmente el modelo está completamente clarificado y es público.

Modelo inicial de Intel

- **i486** y **Pentium**:
  - Operaciones en orden de programa.
    - **Excepción**: Fallos de lectura adelantan escrituras en *write buffer* solamente si todas las escrituras son aciertos de caché.
    - Es imposible que el fallo de lectura coincida con una escritura.

Operaciones atómicas

- Desde **i486**:
  - Leer o escribir 1 byte.
  - Leer o escribir una palabra alineada a 16 bits.
  - Leer o escribir una doble palabra alineada a 32 bits.
- Desde **Pentium**:
  - Leer o escribir quadword alineado a 64 bits.
  - Acceso a memoria no cacheada que cabe en bus de datos de 32 bits.
- Desde **P6**:
  - Acceso no alineado a datos de 16, 32 o 64 bits que caben en una línea de caché.

Bloqueo del bus (I)

- Un procesador puede emitir una **señal de bloqueo** del bus.
  - Otros elementos **no pueden acceder** al bus.

- **Bloqueo automático del bus**:
  - Instrucción **XCHG**.
  - Actualización de **descriptores de segmento**, **directorio de páginas** y **tabla de páginas**.
  - Aceptación de interrupciones.

Bloqueo del bus (II)

- **Bloqueo software del bus**:
  - Uso del prefijo **LOCK** en:
  - Instrucciones de comprobación y modificación de bit (**BTS**, **BTR**, **BTC**).
  - Instrucciones de intercambio (**XADD**, **CMPXCHG**, **CMPXCHG8B**).
  - Instrucciones aritméticas de 1 operando (**INC**, **DEC**, **NOT**, **NEG**).
  - Instrucciones aritmético-lógicas de 2 operandos (**ADD**, **ADC**, **SUB**, **SBB**, **AND**, **OR**, **XOR**).

Instrucciones de barrera

- **LFENCE**:
  - Barrera para **operaciones de load**.
  - Cada **load previo** a **LFENCE** se hace **globalmente visible** antes que cualquier **load posterior**.
- **SFENCE**:
  - Barrera para **operaciones de store**.
  - Cada **store previo** a **SFENCE** se hace **globalmente visible** antes que cualquier **store posterior**.
- **MFENCE**:
  - Barrera para **operaciones de load/store**.
  - Todos los **load y store previos** a **MFENCE** son **globalmente visibles** antes que cualquier **load o store posterior**.

Modelo de memoria actual dentro del procesador (I)

- Lecturas no adelantan lecturas (R $\rightarrow$ R).
- Escrituras no adelantan lecturas (R $\rightarrow$ W).
- Escrituras no adelantan escrituras (W $\rightarrow$ W).
- Hay excepciones para strings y movimientos no temporales.
- Lecturas si adelantan escrituras anteriores (W $\rightarrow$ R) a direcciones diferentes.
- Lecturas/escrituras no adelantan a operaciones de (E/S), instrucciones con cerrojo o instrucciones de serialización.

Modelo de memoria actual dentro del procesador (II)

- Lecturas no pueden sobrepasar **LFENCE** o **MFENCE** anteriores.
- Escrituras no pueden sobrepasar **LFENCE**, **SFENCE** o **MFENCE** anteriores.
- **LFENCE** no puede sobrepasar lectura anterior.
- **SFENCE** no puede sobrepasar escritura anterior.
- **MFENCE** no puede sobrepasar lectura o escritura anterior.

Modelo de memoria multiprocesador

- Cada procesador cumple con reglas anteriores individualmente.
- Las escrituras de un procesador se observan en el mismo orden por todos los demás.
- Las escrituras de un procesador NO se ordenan con respecto a las escrituras de otros procesadores.
- La ordenación de memoria es transitiva.
- Dos escrituras son vistas en un orden consistente por cualquier procesador distinto de esos dos.
- Las instrucciones de cerrojo tienen un orden total.

4 Caso de uso: Intel

- Modelo de consistencia
- Ejemplos
- Efectos del modelo

Ejemplo: Orden de escrituras

Procesador A	Procesador B	Procesador C
write A.1	write B.1	write C.1
write A.2	write B.2	write C.2
write A.3	write B.3	write C.3

- Las escrituras de cada procesador mantienen el orden.

Posible Orden (I)	Posible Orden (II)
Write A.1	...
Write B.1	Write B.3
Write B.2	Write A.3
Write C.1	Write C.2
Write A.2	Write C.3

- Se mantiene el orden de cada proceso.
- No se garantiza ningún orden entre procesos.

No reordenación R→R,W→W

Estado inicial

X=0, Y=0

Procesador 1

MOV [_x], 1
MOV [_y], 1

Procesador 2

MOV r1, [_y]
MOV r2, [_x]

Estado **NO permitido**

r1=1 y r2=0

No reordenación R→W

Estado inicial

X=0, Y=0

Procesador 1

```
MOV r1, [_x]
MOV [_y], 1
```

Procesador 2

```
MOV r2, [_x]
MOV [_x], 1
```

Estado NO permitido

r1=1 y r2=1

Reordenación W(a)→R(b)

Estado inicial

X=0, Y=0

Procesador 1

```
MOV [_x], 1
MOV r1, [_y]
```

Procesador 2

```
MOV [_y], 1
MOV r2, [_x]
```

Estado **permitido**

r1=0 y r2=0

No reordenación W→R

Estado inicial

X=0

Procesador 1

MOV [_x], 1
MOV r1, [_x]

Estado **NO permitido**

r1=0

Visibilidad de escrituras por otro procesador

Estado inicial

X=0, Y=0

Procesador 1

```
MOV [_x], 1
MOV r1, [_x]
MOV r2, [_y]
```

Procesador 2

```
MOV [_y], 1
MOV r3, [_y]
MOV r4, [_x]
```

Estado **permitido**

r2=0 y r4=0

Las escrituras pueden percibirse en distinto orden por cada procesador.

Visibilidad transitiva de escrituras

Estado inicial

X=0, Y=0

Procesador 1

```
MOV [_x], 1
```

Procesador 2

```
MOV r1, [_x]
MOV [_y], 1
```

Procesador 3

```
MOV r2, [_y]
MOV r3, [_x]
```

Estado NO permitido

r1=1 y r2=1 y r3=0

Orden consistente de escritura para otros procesadores

Estado inicial

X=0, Y=0

Procesador 1

MOV [_x], 1

Procesador 2

MOV [_y], 1

Procesador 3

MOV r1, [_x]
MOV r2, [_y]

Procesador 4

MOV r3, [_y]
MOV r4, [_x]

Estado **NO permitido**

r1=1 y r2=0 y r3=1 y r4=0

Instrucciones con cerrojo definen un orden total

Estado inicial

r1=1, r2=1, X=0, Y=0

Procesador 1	Procesador 2	Procesador 3	Procesador 4
XCHG [_X], r1	**XCHG** [_y], r2	**MOV** r3, [_x] **MOV** r4, [_y]	**MOV** r5, [_y] **MOV** r6, [_x]

Estado **NO permitido**

r1=1 y r2=0 y r3=1 y r4=0

Lecturas no se reordenan con cerrojos

Estado inicial

X=0, Y=0, r1=1, r3=1

Procesador 1

XCHG [_x], r1
MOV r2, [_y]

Procesador 2

XCHG [_y], r3
MOV r4, [_x]

Estado **no permitido**

r2=0 y r4=0

Escrituras no se reordenan con cerrojos

Estado inicial

X=0, Y=0, r1=1

Procesador 1

XCHG [_x], r1
MOV [_y], r1

Procesador 2

MOV r2, [_y]
MOV r3, [_x]

Estado **no permitido**

r2=1 y r3=0

4 Caso de uso: Intel

- Modelo de consistencia
- Ejemplos
- Efectos del modelo

Modelos de consistencia en Intel

- **Consistencia secuencial**
  - **Load**: **mov reg**, **[mem]**
  - **Store**: **xchg [mem]**, **reg**
- **Consistencia relajada**
  - **Load**: **mov reg**, **[mem]**
  - **Store**: **mov [mem]**, **reg**
- **Consistencia de liberación adquisición**
  - **Load**: **mov reg**, **[mem]**
  - **Store**: **mov [mem]**, **reg**

Resumen

- Modelo de consistencia de memoria determina qué optimizaciones son válidas.
- La **consistencia secuencial** establece como restricciones la **atomicidad** y el **orden de programa**.
- Se pueden usar modelos más relajados que la consistencia secuencial.
  - **Consistencia débil**.
  - **Consistencia adquisición liberación**
- El modelo de memoria de Intel ha ido evolucionando en la última década.
  - Formalizado y públicamente disponible.
  - Establece qué operaciones son atómicas, cuándo se bloquea el bus y cómo se definen barreras.
  - Define el modelo de memoria dentro del procesador y entre distintos procesadores.

Referencias

- **Computer Architecture. A Quantitative Approach.**
  5th Ed.
  Hennessy and Patterson.
  Secciones: 5.6

- **Shared memory consistency models: A tutorial.**
  Adve, S. V., and Gharachorloo, K.
  IEEE Computer 29, 12 (December 1996), 66-76.

- **Intel 64 and IA-32 Architectures Software Developer Manuals.**
  Volume 3: Systems Programming Guide.
  8.2: Memory Ordering

Modelos de consistencia de memoria

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid