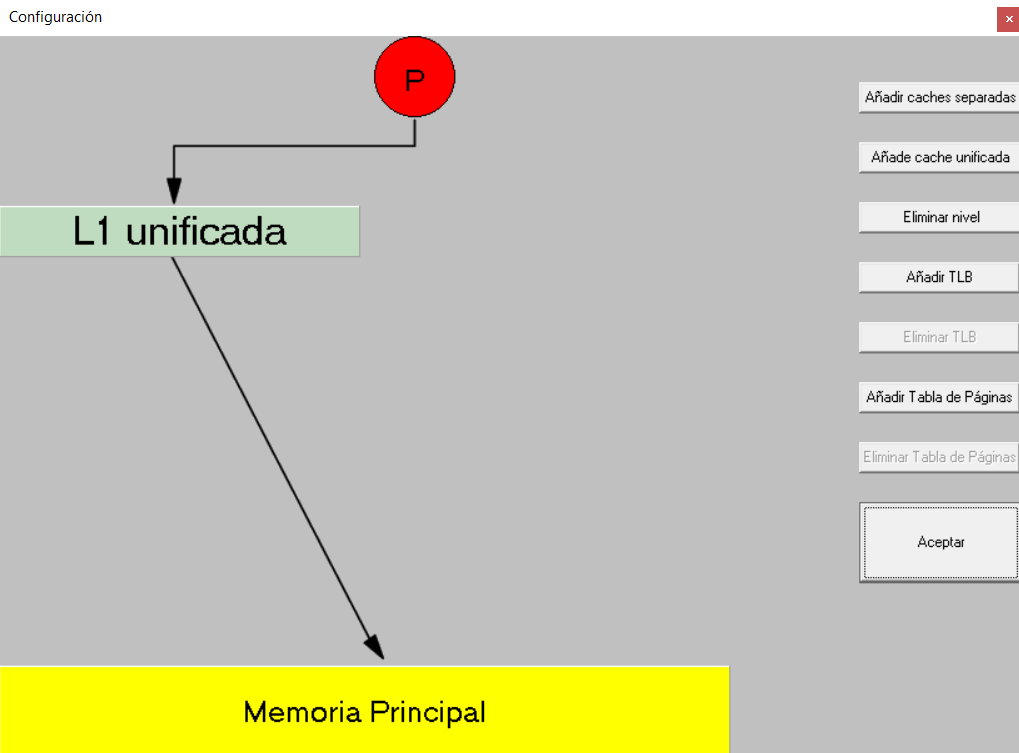
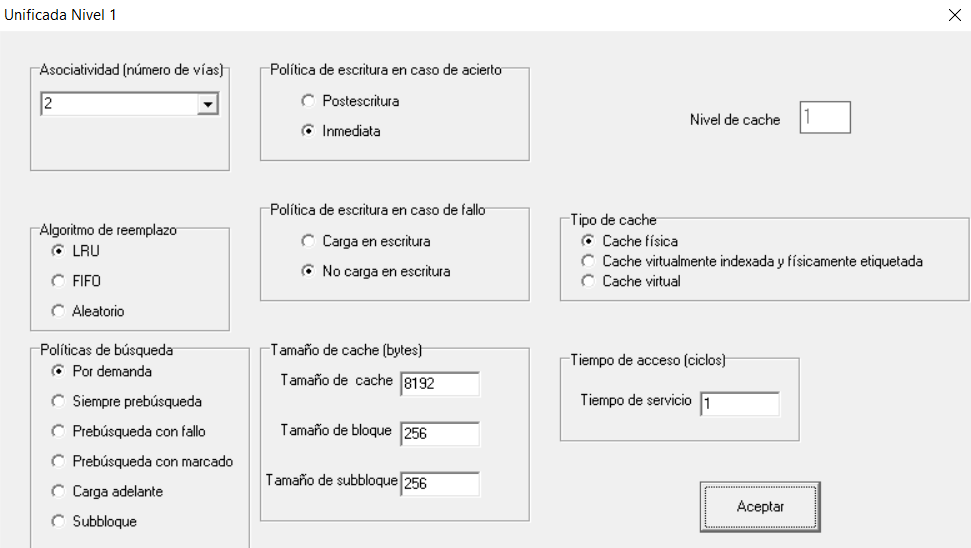
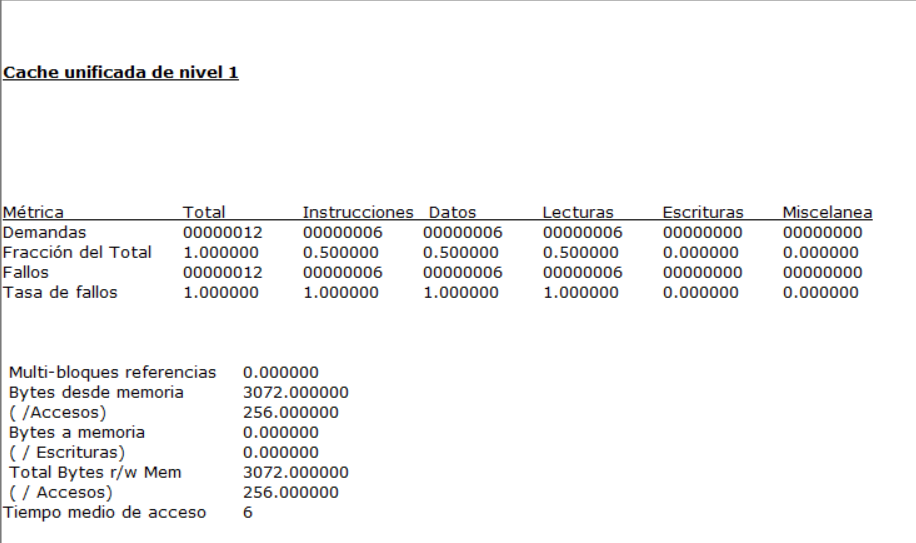
Práctica 5 OC

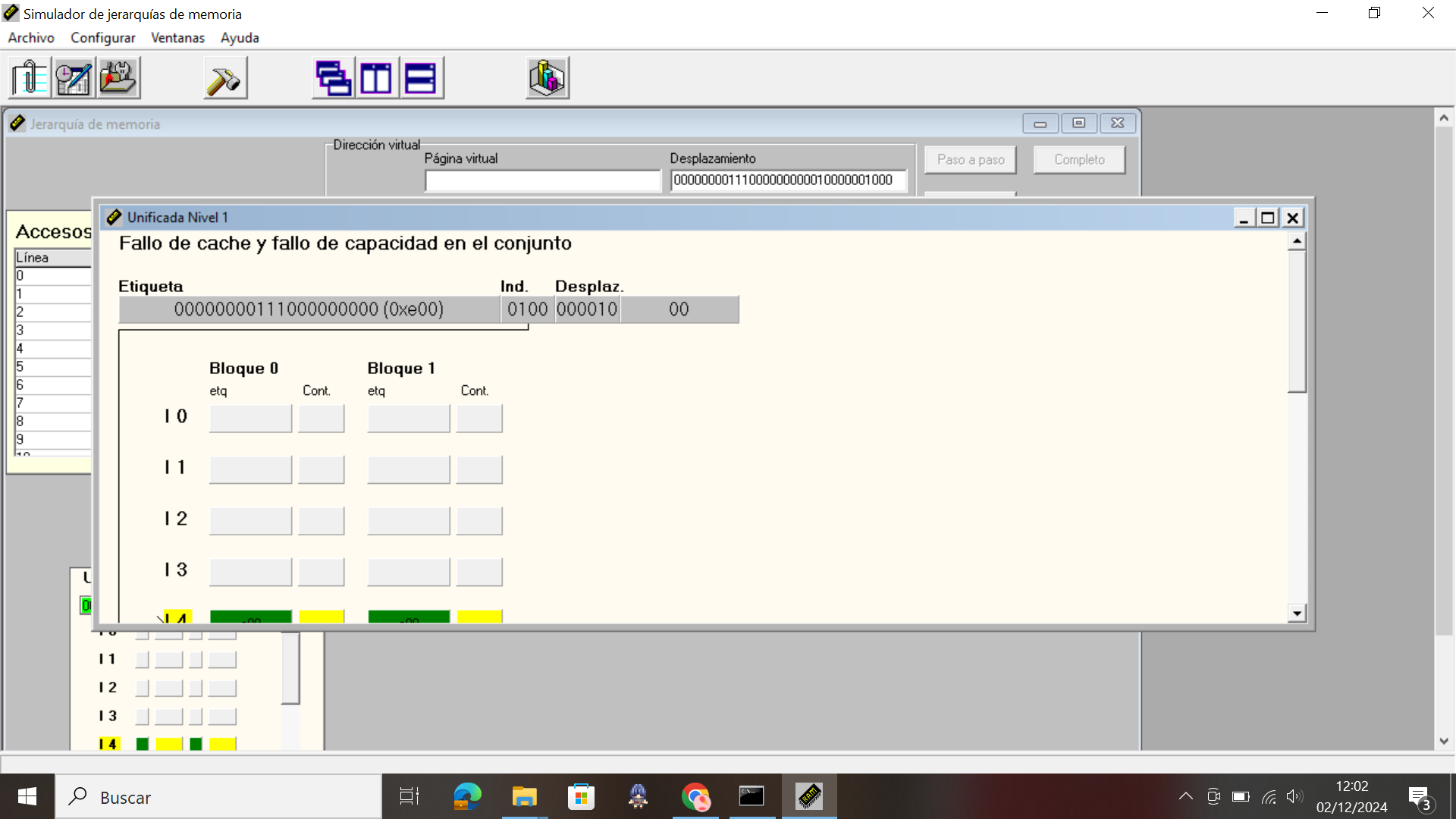
Estudio 1: Caché mononivel vs. Caché multinivel:

**Añadimos L1 y configuramos**

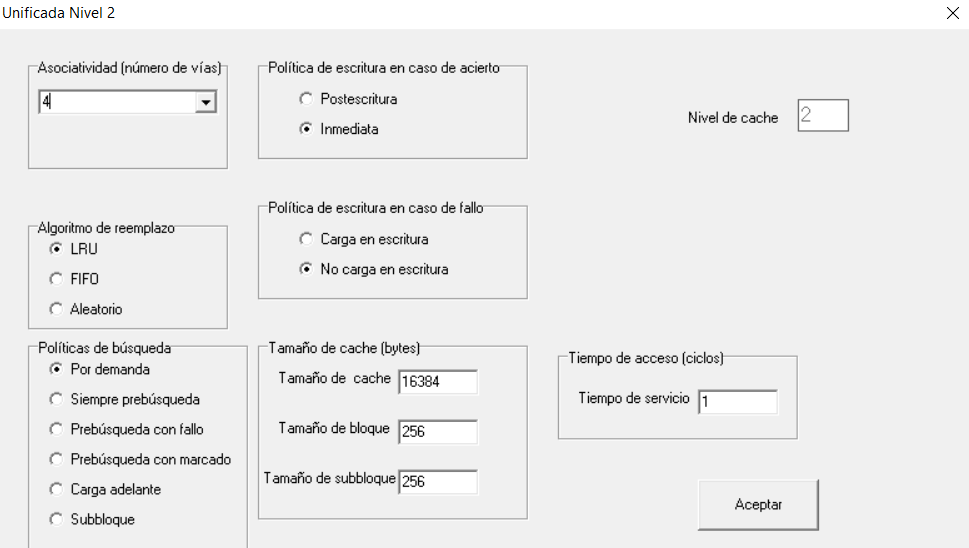
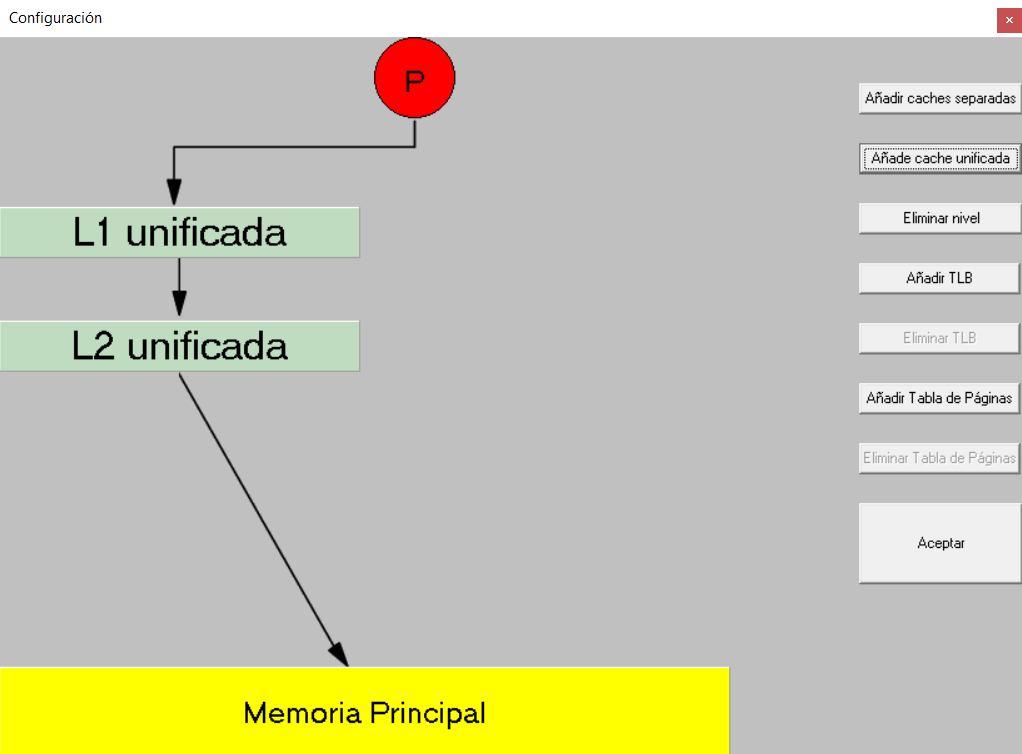




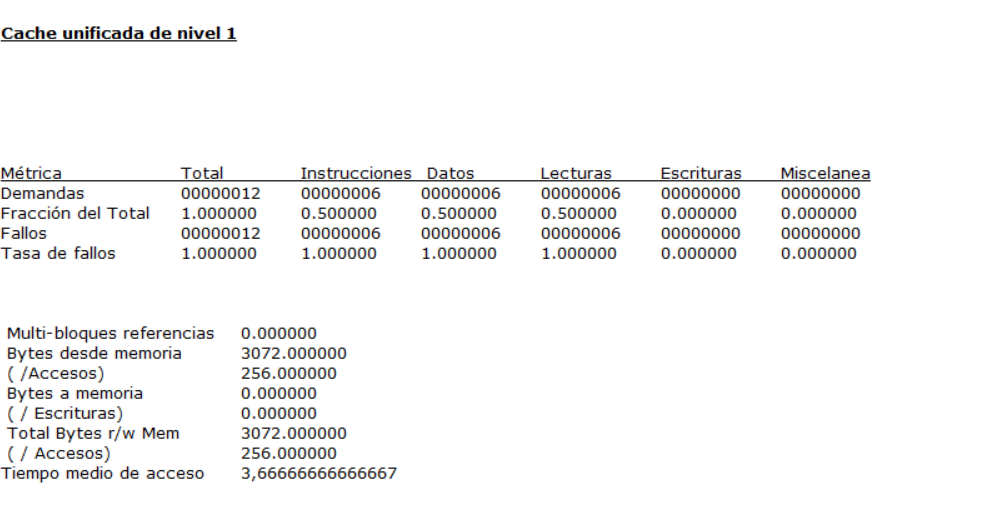




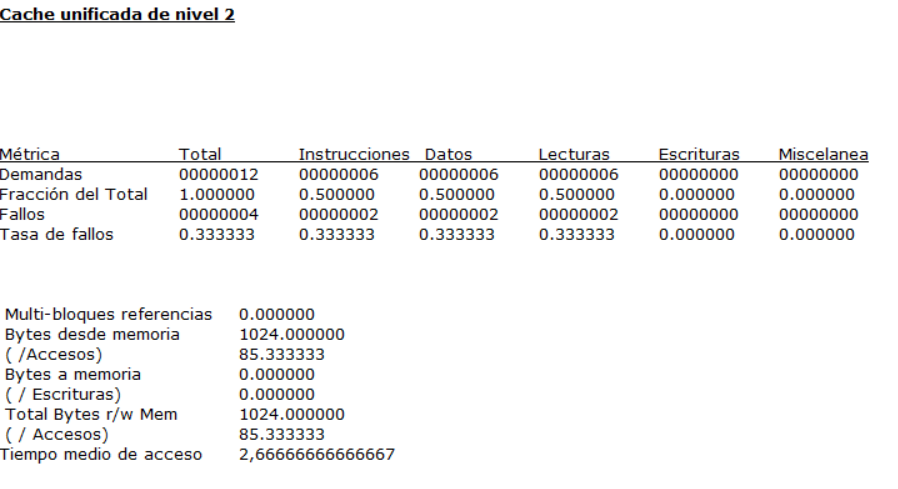
**Añadimos L2 y configuramos L2**



**Datos de L1 en multilevel**



**Datos de L2 en multilevel**

****

|  | **Tasa de fallos** |
| --- | --- |
| **L1 solo** | **1** |
| **L1 multilevel** | **1** |
| **L2 multilevel** | **0.33333** |

**Tiempo medio de acceso total: 5.33333**

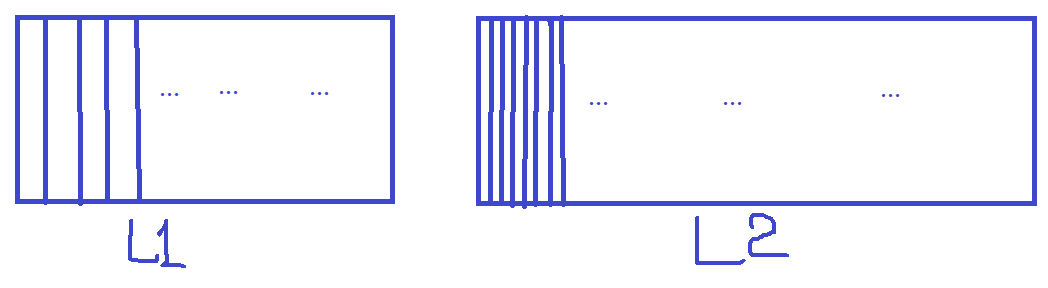
**Tasa de fallos global: Tasa de fallos de L1 x Tasa de fallos de L2 = 1 x 0.33 = 0,33**

**Por qué pasa esto?**

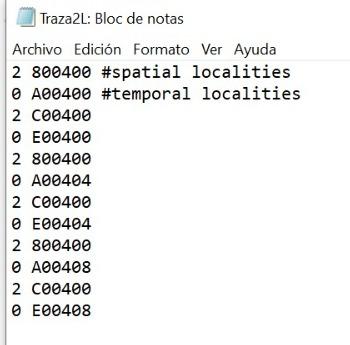
En este caso, la asociatividad de L2 es 4-way, frente a L1 que simplemente es 2-way, por lo que a mayor nivel de asociatividad mayor forma de solucionar conflictos. También influye que el tamaño de caches de L2 es el doble de grande

### **Por qué L2 soluciona los conflict misses de L1:**

1. **Mayor tamaño de la caché L2**:
   * La caché L2 tiene un tamaño mayor (16 KB frente a 8 KB de L1). Esto permite que L2 almacene una mayor cantidad de bloques de datos, reduciendo la probabilidad de reemplazos forzados debidos a conflictos en L1.
2. **Mayor asociatividad en L2**:
   * L2 es **4-way set associative**, mientras que L1 es **2-way set associative**. Esto significa que, en L2, cada conjunto puede almacenar más bloques (4 en L2 frente a 2 en L1). Esto disminuye la probabilidad de conflictos al aumentar la flexibilidad para asignar bloques dentro de cada conjunto.



El dibujo representa que L1 es más pequeño y a su vez tiene menos huecos para ir metiendo las direcciones de memoria, por lo que no puede tomar todos los datos sin que no haya conflictos de espacio, L2 puede hacerse cargo de ello por lo que reduce la tasa de fallos. Podría funcionar igual aunque tuviesen el mismo tamaño L1 y L2, debido a la asociatividad mayor de L2



### **Capacity Miss:**

**Un fallo por capacidad ocurre cuando el conjunto de datos que se necesita acceder no cabe completamente en la caché, independientemente de cómo estén mapeados. Esto se refleja claramente en tu ejemplo:**

1. **Bloque 4 del Set 1:**
   * **Guardas la dirección 800 y luego se reemplaza por c00, pero más adelante necesitas 800 nuevamente.**
   * **Esto indica que 800 no permaneció en la caché porque el espacio en el set (en este caso, con 2 líneas) no era suficiente para mantener tanto 800 como c00 al mismo tiempo.**
2. **Bloque 4 del Set 2:**
   * **Similarmente, la dirección a00 es reemplazada por e00, y luego se reemplaza nuevamente por a00.**
   * **Aquí también se observa que los datos están siendo reemplazados debido a la falta de espacio, lo que es un típico fallo por capacidad.**

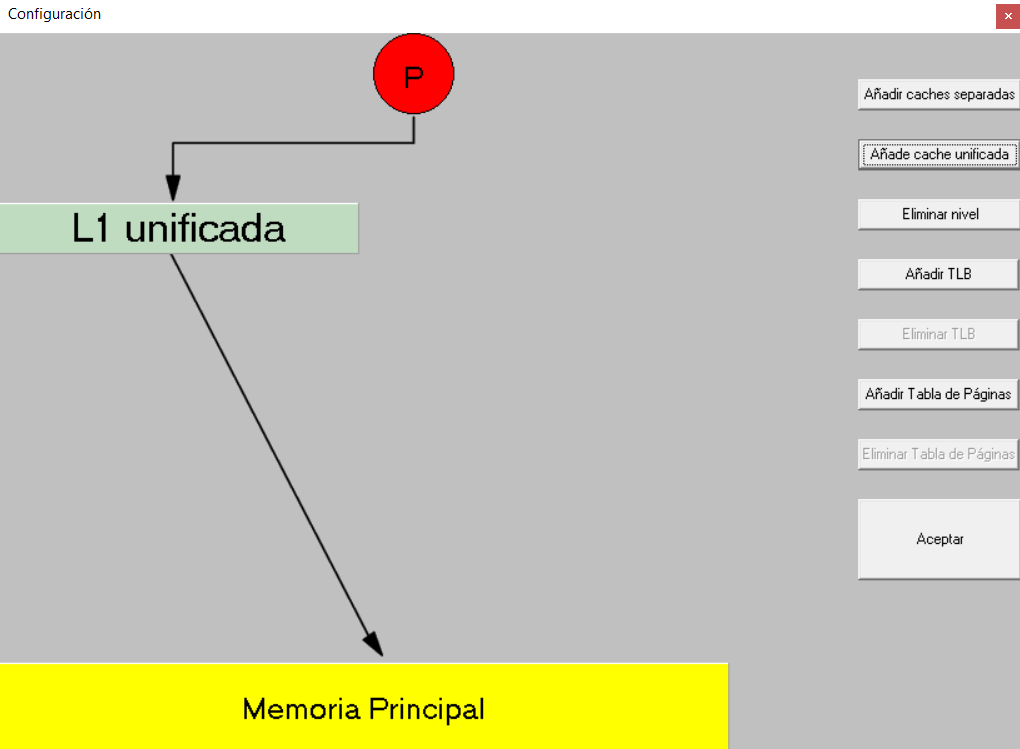
### **Localidad:**

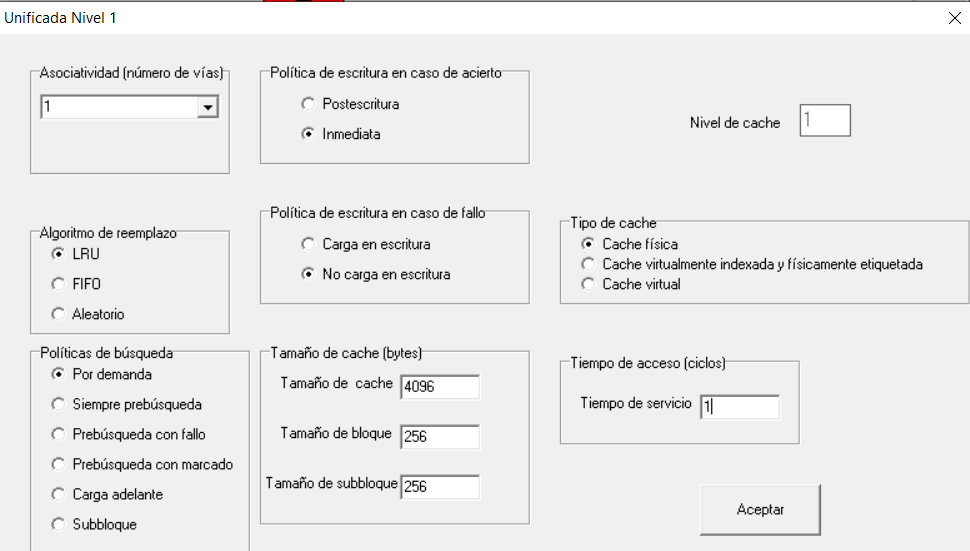
**La deducción de localidad temporal y espacial está bien fundamentada. Aquí un desglose:**

1. **Temporal Locality:**
   * **La localidad temporal se refiere a que una misma dirección se accede repetidamente en un corto periodo de tiempo.**
   * **Ejemplos claros:**
     + **Address 800400 se accede en las líneas 0, 4, y 8.**
     + **Address C00400 se accede en las líneas 2, 6, y 10.**
   * **Esto muestra que ciertos datos son reutilizados repetidamente, pero los fallos por capacidad impiden que se queden en la caché el tiempo suficiente para ser aprovechados.**
2. **Spatial Locality:**
   * **La localidad espacial se refiere a que direcciones cercanas son accedidas dentro de un periodo de tiempo corto.**
   * **Ejemplos claros:**
     + **Addresses A00400, A00404, A00408 se acceden en las líneas 1, 5, y 9.**
     + **Addresses E00400, E00404, E00408 se acceden en las líneas 3, 7, y 11.**
   * **Esto indica que los accesos están aprovechando bloques consecutivos (por ejemplo, los bloques contiguos dentro de una misma página de memoria o dentro de un rango cercano).**

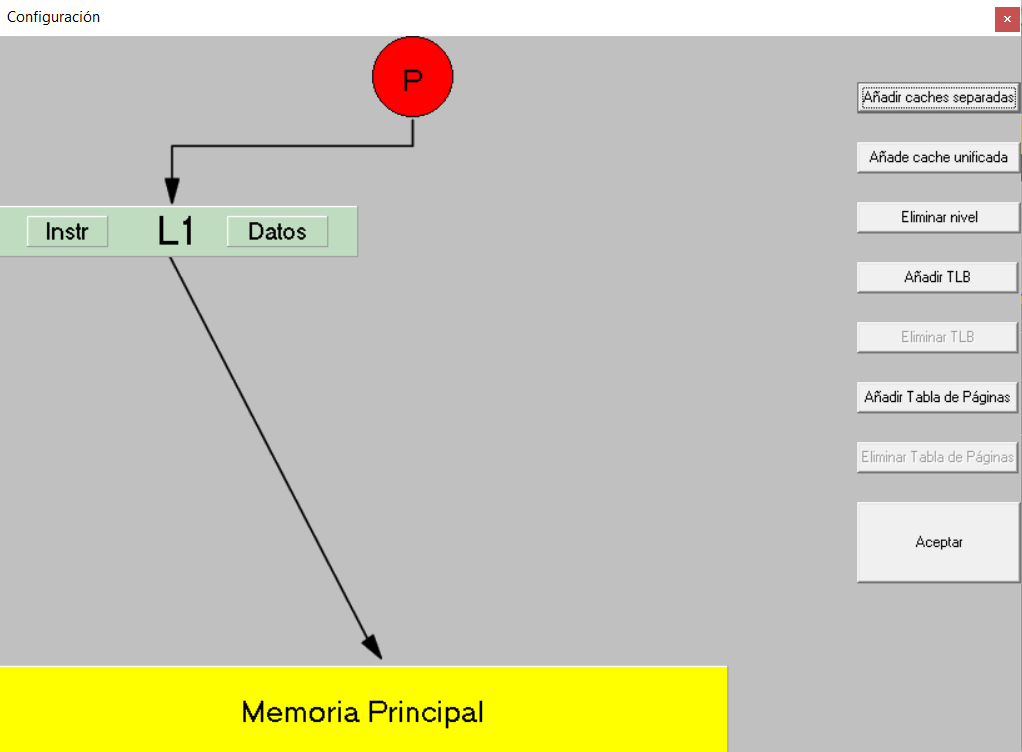
**Estudio 2: Caché unificada vs. Caché separada:**

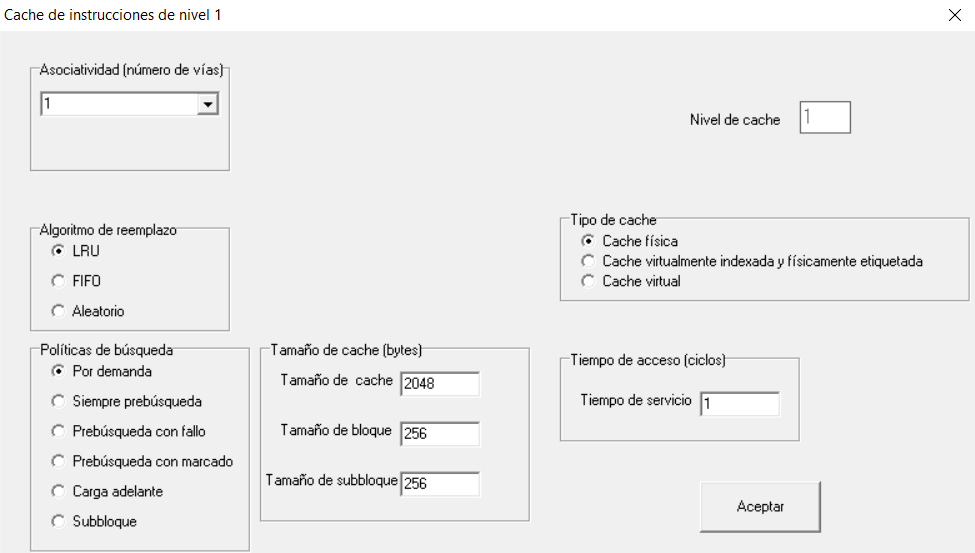
**Caché unificada**

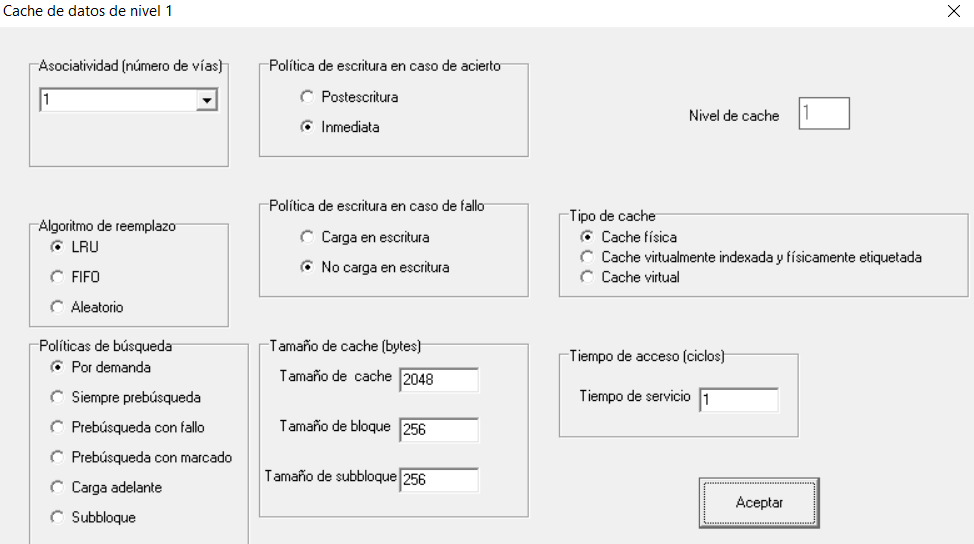
****

****

**Caché separada**

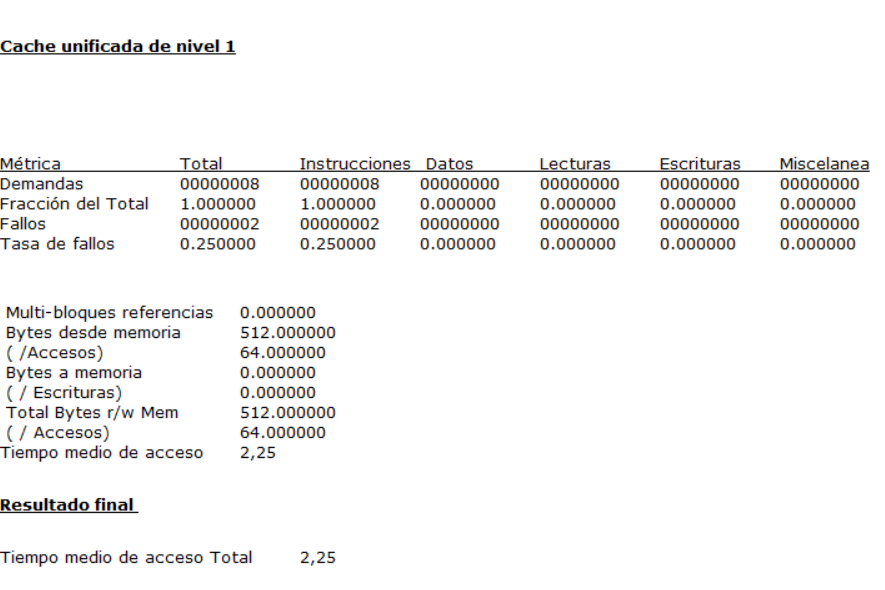
****

****

****

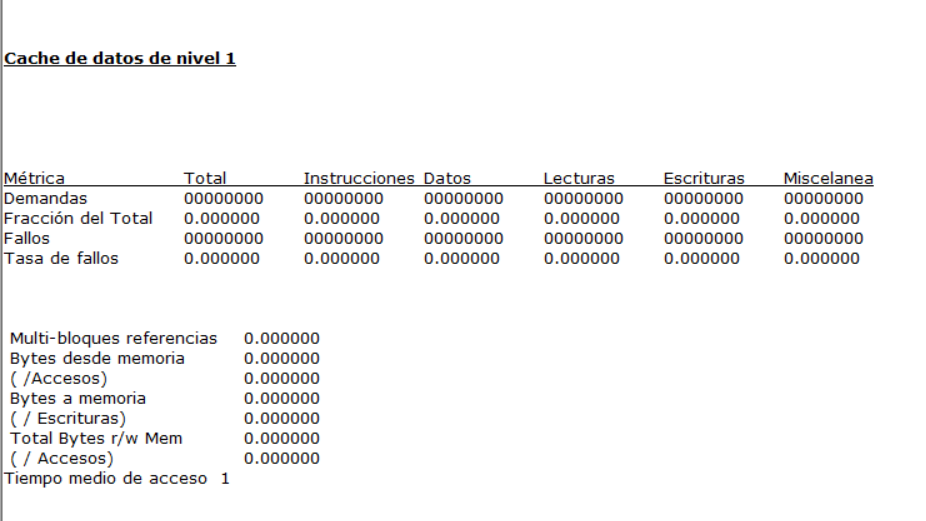
**TrazaUN**

**--->Unificada**

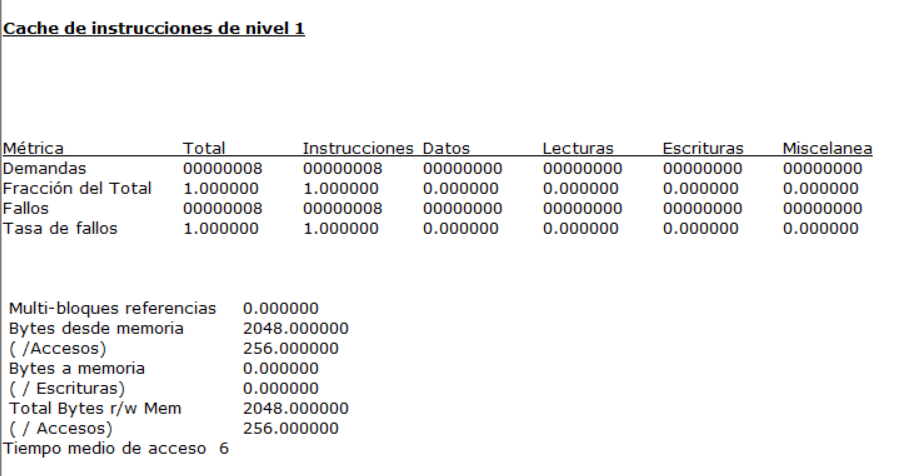
****

**--->Separada**

**--->Datos**

****

**--->Instrucciones**

****

**--->Total**

****

Total misses = miss datos + miss instrucciones

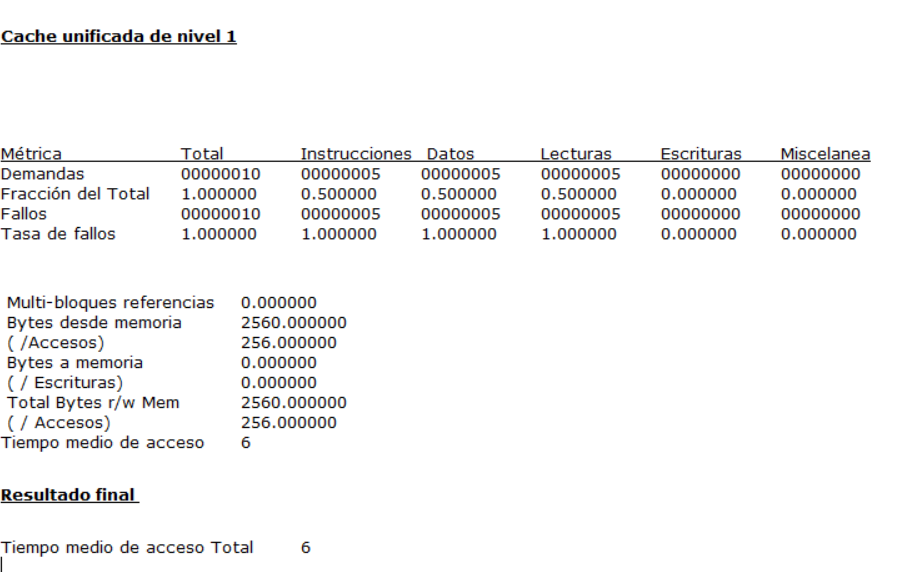
Hay spatial and temporal localities

La traza UN no tiene acceso a datos DIRECTOS

TMAT **mejor en unificiada que en separada**, por qué? Porque la caché unificada consta de más bloques que la caché de instrucciones (separada), por lo que hay conflictos cuando se accede en la de instrucciones, algo que se evita en la unificada

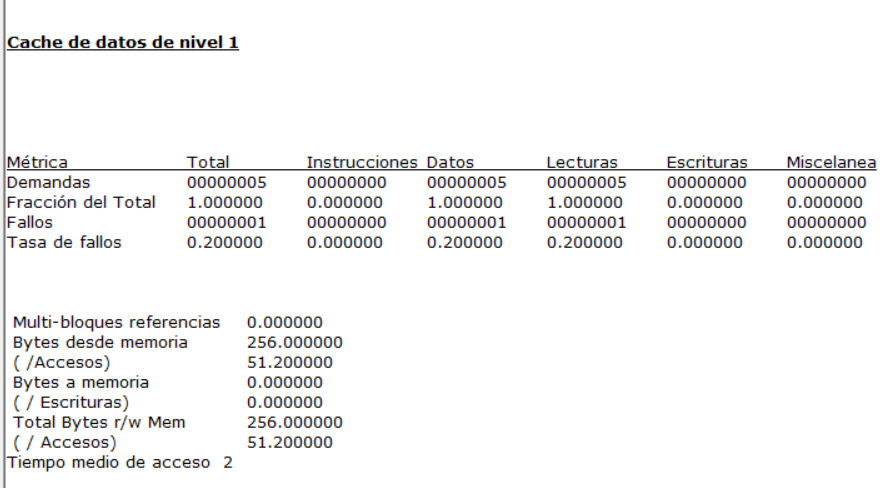
**TrazaSDI**

**--->Unificada**

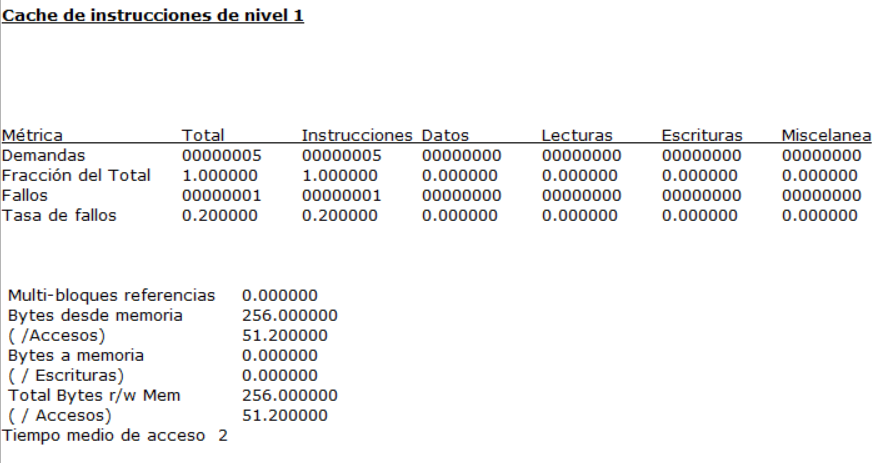
****

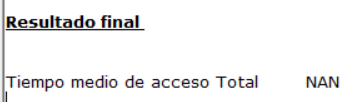
**--->Separada**

**--->Datos**

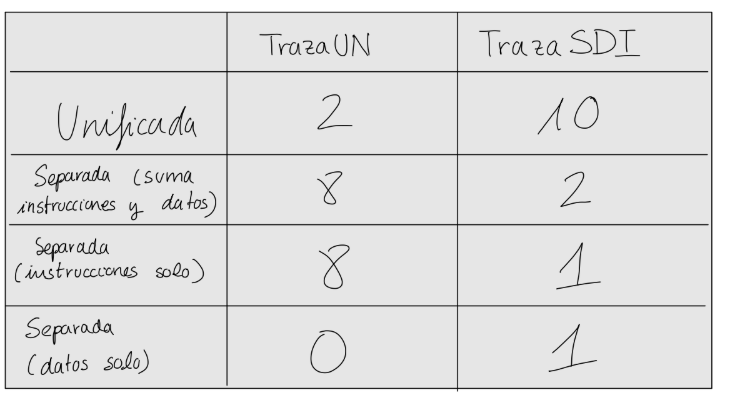
****

**--->Instrucciones**

****

****

¿¿Qué es esto, demasiado rápido??

****

### Interpretación de los resultados:

#### Para TrazaUN:

* Caché unificada: Solo 2 fallos, lo que demuestra que es más eficiente en este caso.
* Caché separada:
  + Total: 8 fallos, todos provenientes de las instrucciones (8 en instrucciones y 0 en datos).
  + Esto indica que las instrucciones en esta traza generan más conflicto en la caché separada, probablemente porque los datos no compiten por espacio con las instrucciones en la caché unificada.

#### Para TrazaSDI:

* Caché unificada: Tiene 10 fallos, mucho más que cualquier configuración de caché separada.
* Caché separada:
  + Total: 2 fallos (1 de instrucciones y 1 de datos).
  + Esto muestra que la separación ayuda a evitar conflictos entre instrucciones y datos en este caso, distribuyendo los accesos entre ambas cachés.

**Además, a la vista de la estructura de las cachés unificada y separada (y de la corres pondencia que establecen), y de las direcciones de los accesos de las trazas empleadas en las simulaciones, analiza cuáles son las razones por las que TrazaUN hace que la caché unificada ofrezca mejores prestaciones que la separada y TrazaSDI hace que la caché separada ofrezca mejores prestaciones que la unificada. De nuevo, este análisis puede realizarse incluso sin el simulador, ya que las trazas son muycortas, y laestructura de las direcciones que resulta de las distintas configuraciones de caché puede deducirse fácilmente. Pero el simulador sin duda aporta una ayuda adicional a dicho análisis**

* TrazaUN: La caché unificada es más eficiente porque no separa datos de instrucciones, lo que permite aprovechar al máximo la capacidad en escenarios con accesos mixtos a direcciones cercanas y repetidas.
* TrazaSDI: La caché separada es más eficiente porque evita conflictos entre datos e instrucciones, lo que es ideal para trazas con patrones claramente diferenciados, como en este caso.

Explicación clara y concisa a la pregunta:

La trazaUN es una mezcla de accesos a datos e instrucciones lo que hace que rente más tomarlo todo de golpe, sin separación

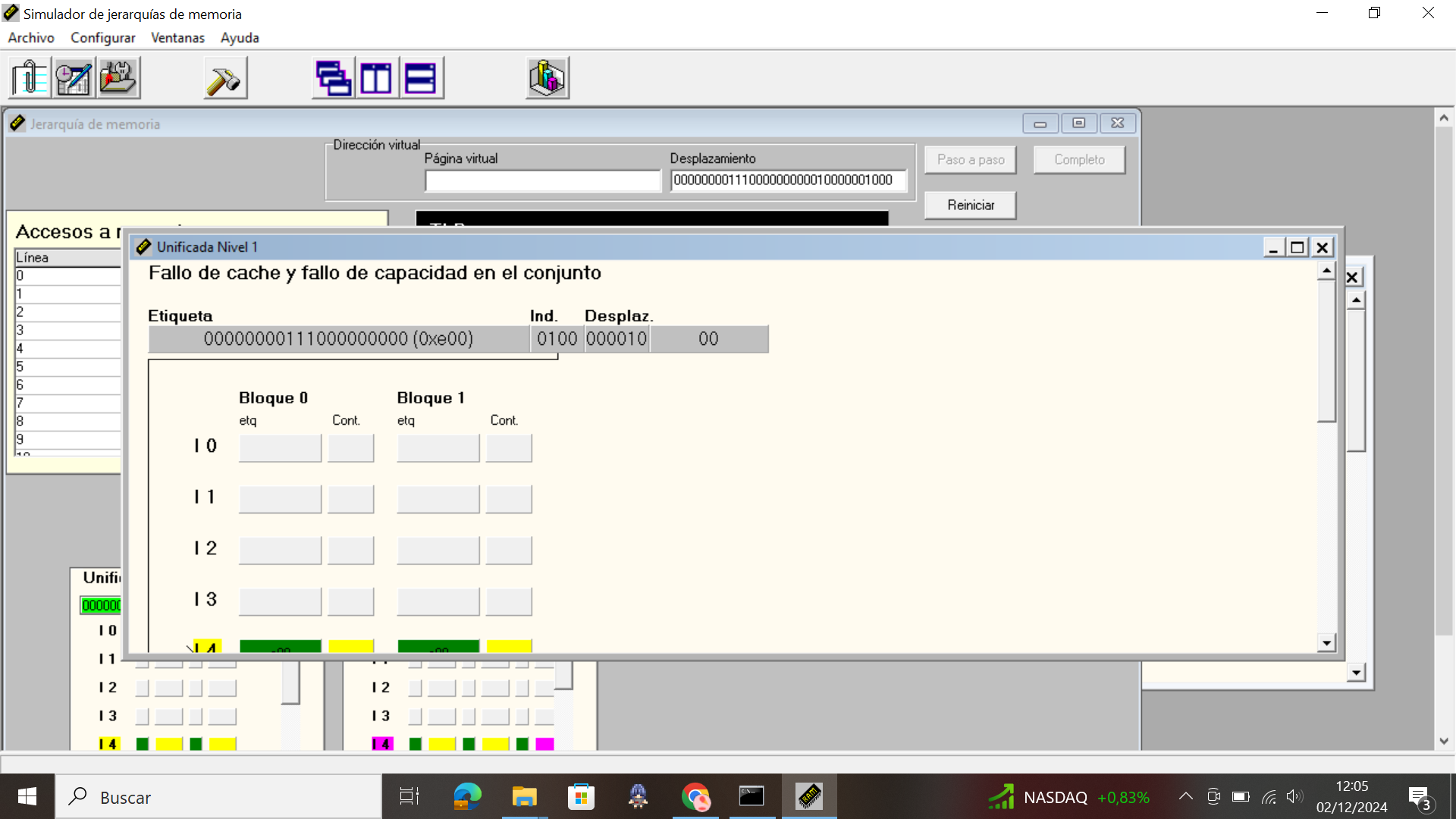
La trazaSDI tiene muy diferenciados los accesos a datos y a instrucciones, por lo que no se entremezclan y renta más primero ir a por una parte(los datos) y después la otra (instrucciones)

Otra explicación más seria:

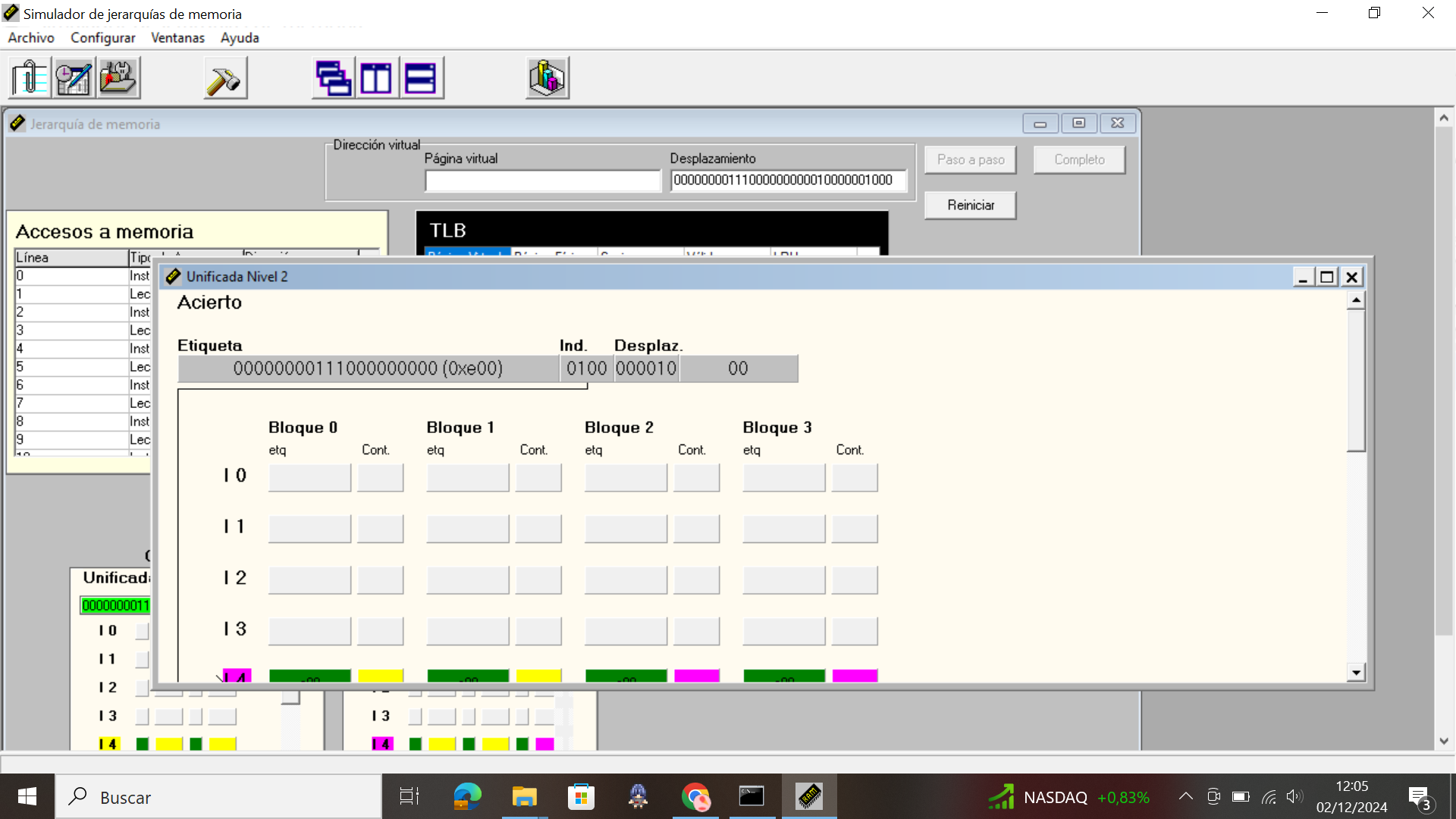
UNIFICADA : The TrazaUN has better "prestaciones" than TrazaSDI for the unified cache, ya que the traze just has access to instructions and none to data. The TrazaSDl, alternates instructions and data, so the cache fails.

SEPARADA: The TrazaUN has worse prestaciones than TrazaSDI for the separated cache, ya que it just uses the instruction cache (no lectures of data) and as the sizo of cache is of 2k, all the instructions are going to substitute the previous one because they access the same cache block.

L1



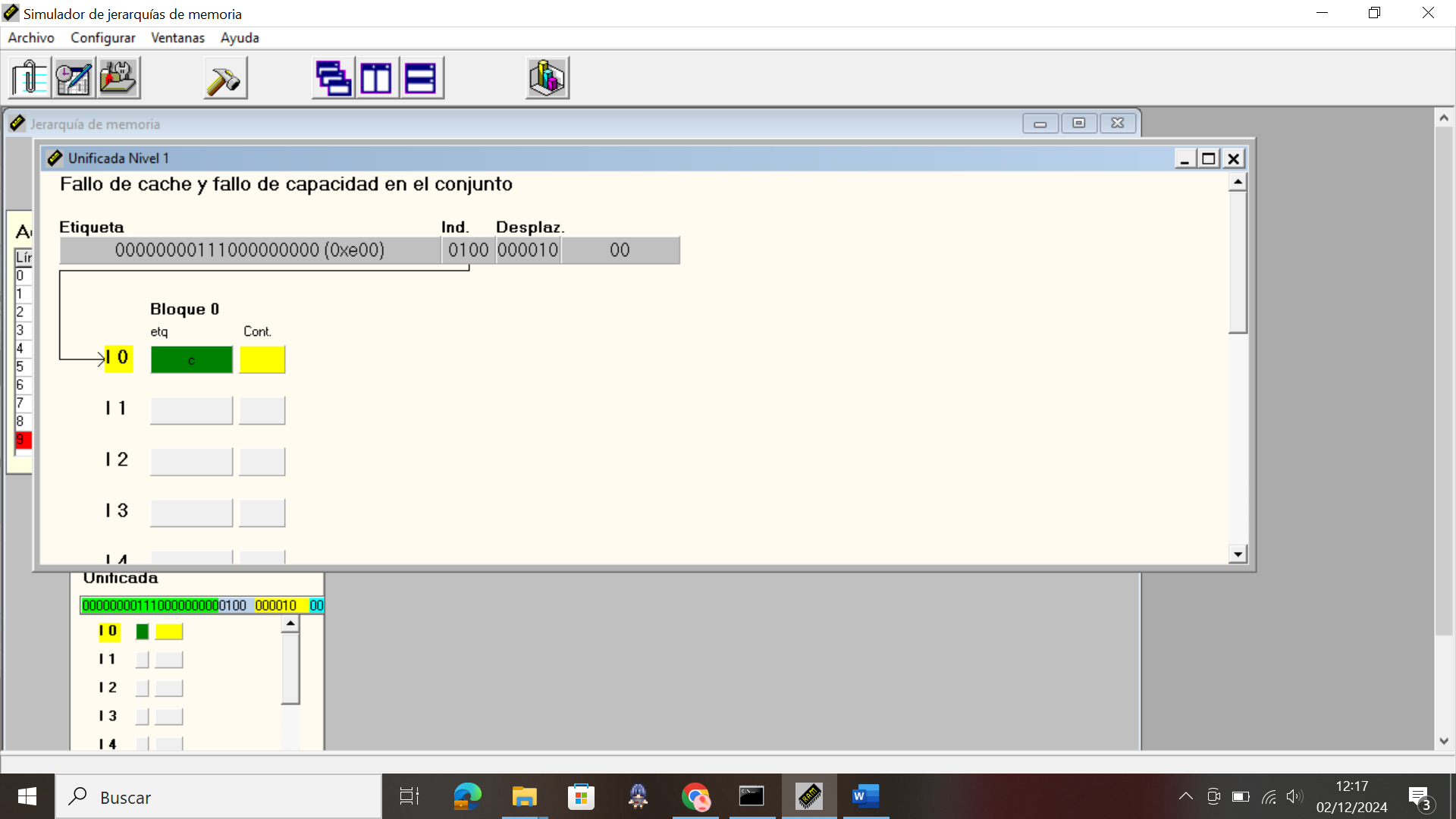
L2



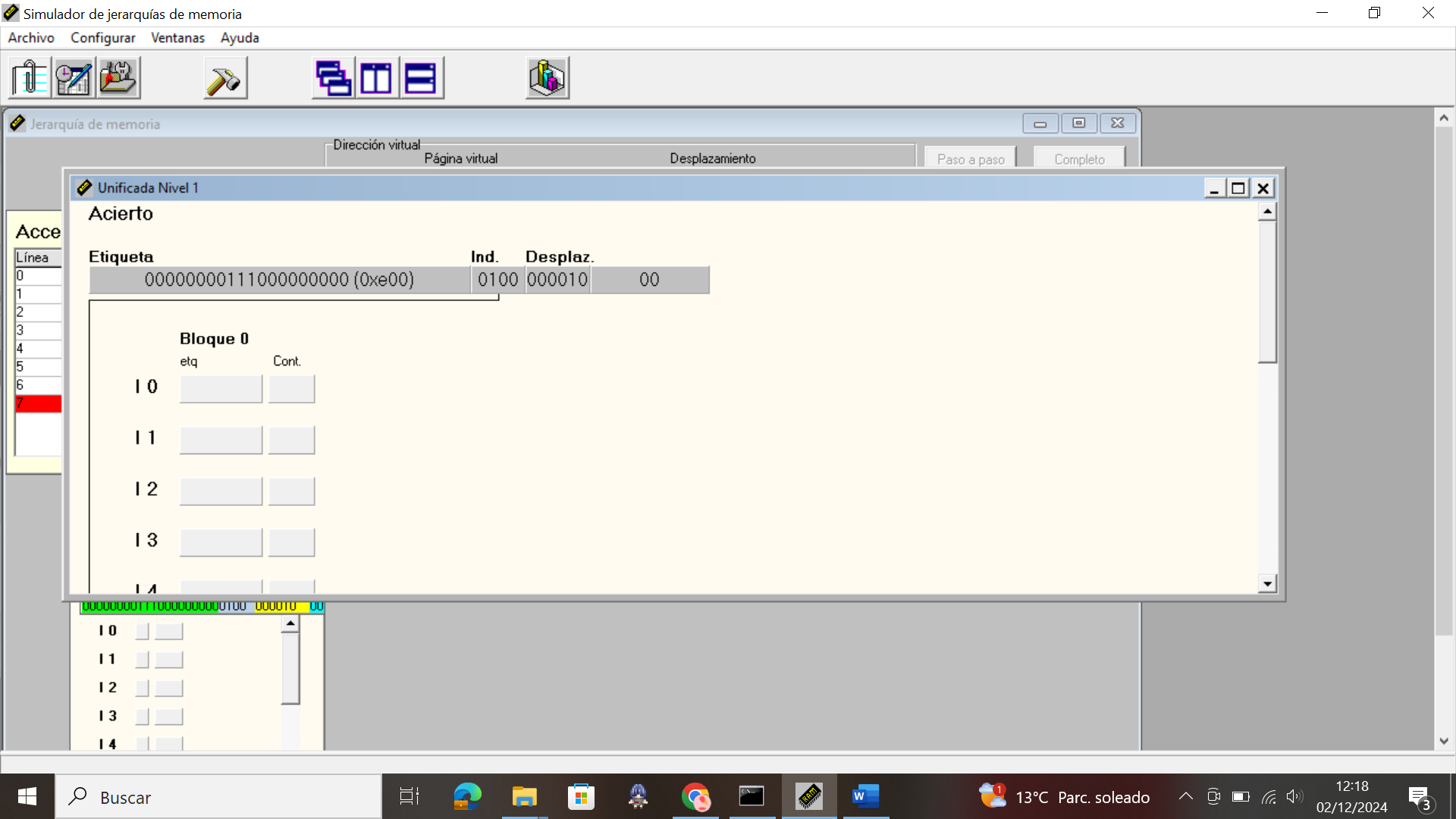
Study 2

COMBINED

SDI

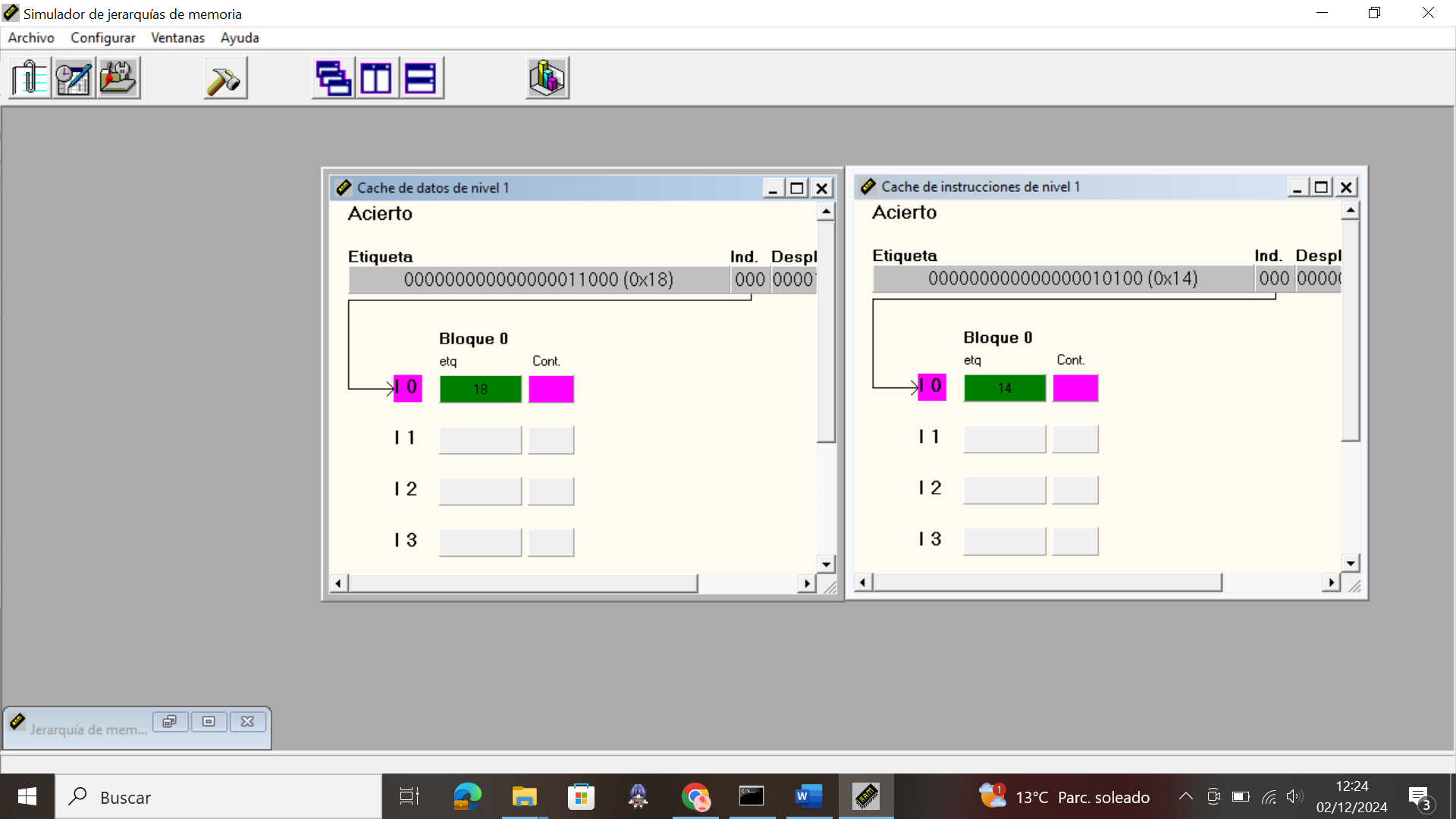


UN



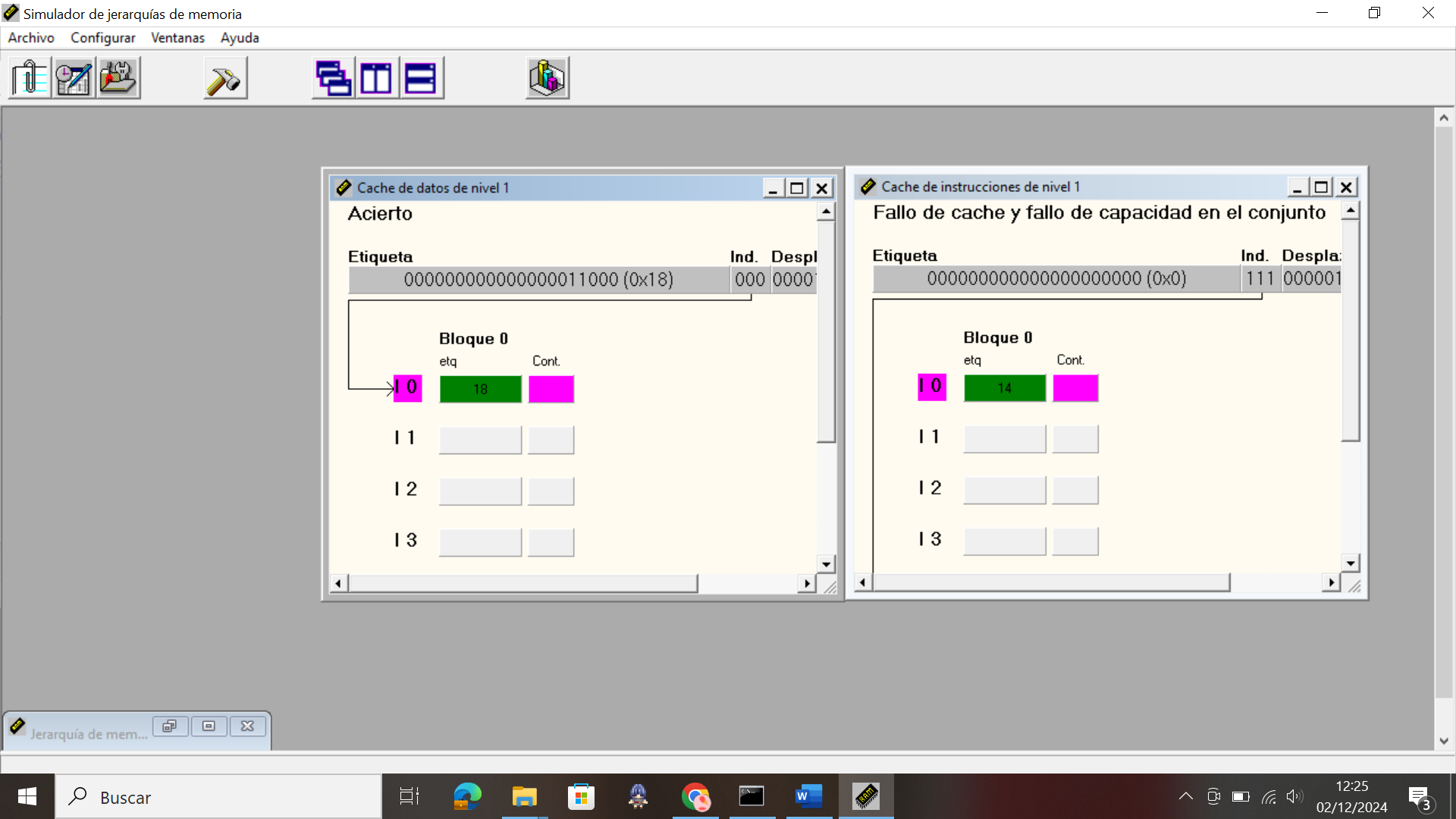
SDI

DATOS e INSTRUCCIONES



UN

DATOS e INSTRUCCIONES



**Extra para calcular los fields en RISC-V. ¡¡¡¡ESTUDIAR POR TU CUENTA!!!!**

**Desglose de direcciones en la caché**

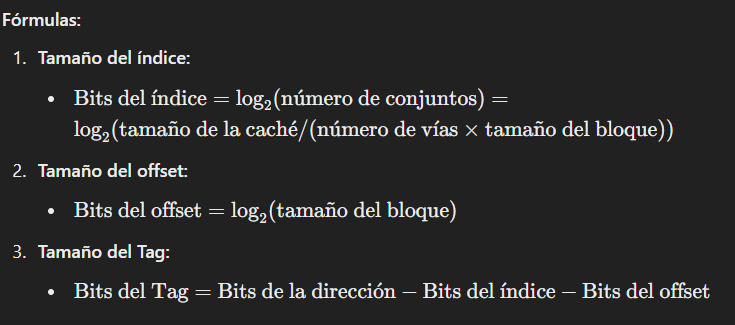
En sistemas de memoria caché, una dirección de memoria se divide en tres partes principales:

1. **Tag Field (Etiqueta):**
   * Se utiliza para identificar el bloque específico de memoria en un conjunto o línea de la caché.
   * Ayuda a resolver colisiones cuando múltiples bloques de memoria pueden mapearse a la misma línea.
2. **Index Field (Índice):**
   * Determina a qué conjunto o línea de la caché pertenece la dirección.
   * El tamaño del índice depende de la cantidad de conjuntos en la caché.
3. **Block Offset (Desplazamiento):**
   * Identifica la posición específica dentro de un bloque de datos (por ejemplo, una palabra o byte).
   * El tamaño del desplazamiento depende del tamaño del bloque de caché.

### **Cómo se calcula el campo Tag en RISC-V**

#### **Suposiciones típicas:**

* **Tamaño de caché = 2^N bytes**
* **Tamaño de bloque = 2^B bytes**
* **Número de líneas de caché = 2^L líneas**

****