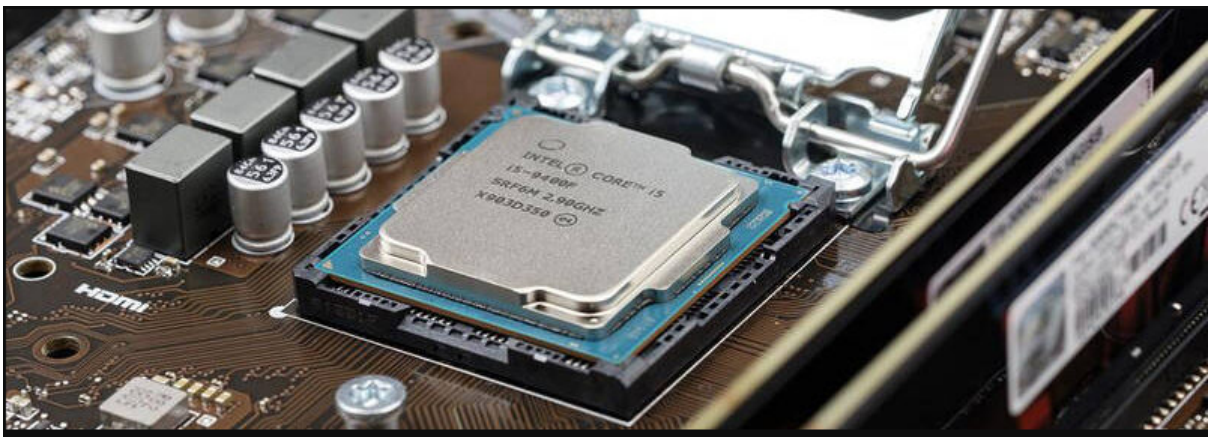
Informe de Proyecto Final de Arquitectura de Computadoras
Simulador del Protocolo MESI



CÁTEDRA: Arquitectura de Computadoras

DOCENTE: Cristian PACHECO

ALUMNO: Ignacio GRILLI.

2do año- Licenciatura Informática.

Índice

1- Resumen del informe

Este informe tiene como objetivo explicar el funcionamiento de un simulador de coherencia de caches que utiliza el protocolo MESI. En él se presentará el programa desarrollado en el lenguaje de programación Java y se proporcionarán instrucciones sobre cómo modificarlo para obtener resultados diferentes.

El programa es un simulador educativo para profundizar en el protocolo MESI.
MESI (Modified Exclusive Shared Invalid) es un protocolo de coherencia de caché utilizado en sistemas de computación multiprocesador con memoria compartida.

El informe fue elaborado por Ignacio David Grilli, alumno de la Universidad Nacional de la Patagonia San Juan Bosco, como trabajo final para la materia Arquitectura de Computadoras del segundo año de la carrera Licenciatura en Informática. El profesor a cargo de la cátedra es Cristian Pacheco.

2-Introducción:

En el contexto de la materia y el trabajo final, se desarrolló un simulador del protocolo MESI en el lenguaje de programación Java, utilizando la metodología orientada a objetos. El simulador permite generar diferentes memorias caché y realizar lecturas y escrituras a través de un lenguaje pseudo-assembler con un número limitado de instrucciones, aunque es posible incluir más para una futura decodificación y mejora.

El programa también incluye una funcionalidad para rastrear el estado de los registros de memoria que se ingresan como parámetros para su configuración, y que se almacenan en un archivo de configuración de texto. En resumen, el programa consta de dos archivos: uno que contiene el programa en lenguaje pseudo-assembler y otro (txt de configuración) que indica la cantidad de memorias caché y procesadores que se desean utilizar para garantizar la consistencia y coherencia de la memoria compartida.

¿Qué es el protocolo MESI?

El protocolo MESI (Modified Exclusive Shared Invalid) es un protocolo de coherencia de caché utilizado en sistemas de computación multiprocesador con memoria compartida. Su objetivo es asegurar la consistencia de la memoria compartida entre los múltiples núcleos de procesamiento que comparten acceso a una misma memoria física.

El protocolo MESI utiliza cuatro estados para cada línea de caché: Modificado (M), Exclusivo (E), Compartido (S), e Inválido (I). Cada uno de estos estados representa una condición diferente de la línea de caché y determina las operaciones que pueden realizarse con ella.

Cuando un procesador necesita acceder a una línea de caché que ya está siendo utilizada por otro procesador, el protocolo MESI permite que los procesadores coordinen sus operaciones para asegurar que la información almacenada en la caché sea coherente y consistente en todos los procesadores.

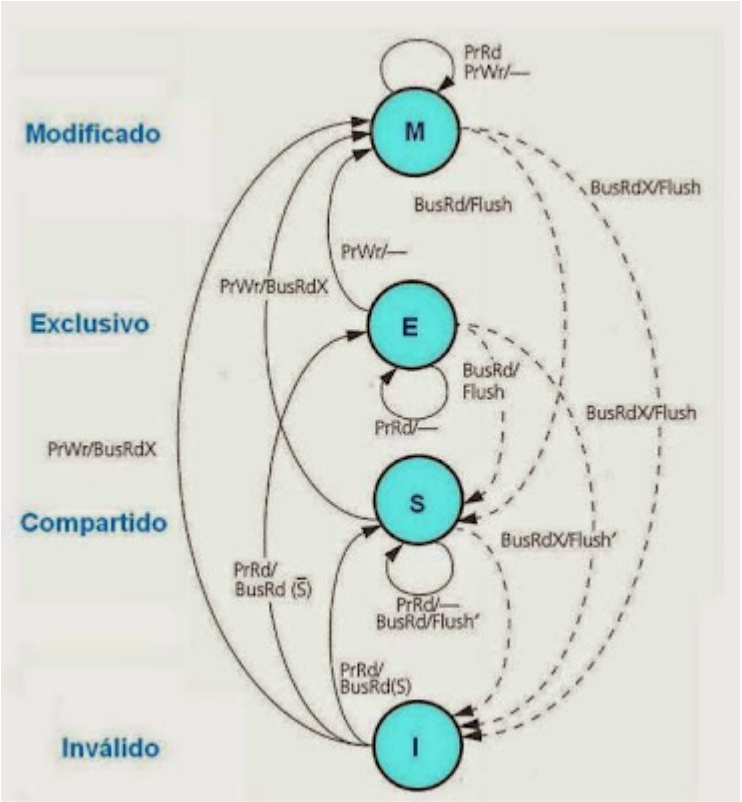
En resumen, el protocolo MESI es un mecanismo de coordinación utilizado en sistemas multiprocesador con memoria compartida para asegurar la coherencia de la memoria caché entre los diferentes procesadores y garantizar que los datos sean consistentes y precisos en todo momento.

A continuación se presentan las transiciones entre los estados de las cachés utilizando el protocolo MESI:

1. Estado Modificado (Modified):
    - Este estado indica que la línea de caché es válida y ha sido modificada localmente por el procesador actual.
    - Transiciones:
        - Lectura (Read): Sí otro procesador solicita leer los datos, el procesador actual debe escribir los cambios a la memoria principal y cambiar al estado Compartido.
        - Escritura (Write): Si el procesador actual modifica los datos almacenados en la línea de caché, no es necesario realizar ninguna transición de estado.
2. Estado Exclusivo (Exclusive):
    - Este estado indica que la línea de caché es válida y sólo está presente en la caché de un procesador.
    - Transiciones:
        - Lectura (Read): Sí otro procesador solicita leer los datos, el procesador actual responde con los datos y permanece en el estado Exclusivo.
        - Escritura (Write): Si el procesador actual modifica los datos almacenados en la línea de caché, cambia al estado Modificado.
3. Estado Compartido (Shared):
    - Este estado indica que la línea de caché es válida y está presente en la caché de múltiples procesadores.
    - Transiciones:
        - Lectura (Read): Sí otro procesador solicita leer los datos, el procesador actual responde con los datos y permanece en el estado Compartido.

- Escritura (Write): Si el procesador actual modifica los datos almacenados en la línea de caché, debe invalidar todas las copias de la línea en otras cachés y cambiar al estado Modificado.

4. Estado Inválido (Invalid):
    - Este estado indica que la línea de caché no es válida o no contiene datos actualizados.
    - Transiciones:
        - Lectura (Read): Sí otro procesador solicita leer los datos, el procesador actual responde con una respuesta de "falla en caché" y permanece en el estado Inválido.
        - Escritura (Write): Si el procesador actual modifica los datos almacenados en la línea de caché, cambia al estado Modificado.



3-Metodología y Resultados

En la programación del simulador se destacan dos métodos que le dan vida al programa y permiten la lógica de la coherencia, detectan el estado de un registro de una caché y actúan de distintas maneras en caso de escritura o lectura de los registros.
Para la decodificación el seudo-Assembler, utiliza estas dos metodos (lecturaDato y escrituraDato en la clase Main.java).
En la parte de decodificación se detectan las instrucciones del seudo-Assembler y estas hacen uso de los métodos anteriormente mencionados y realizan operaciones extras para tomar otras decisiones de escritura o lectura según corresponda.

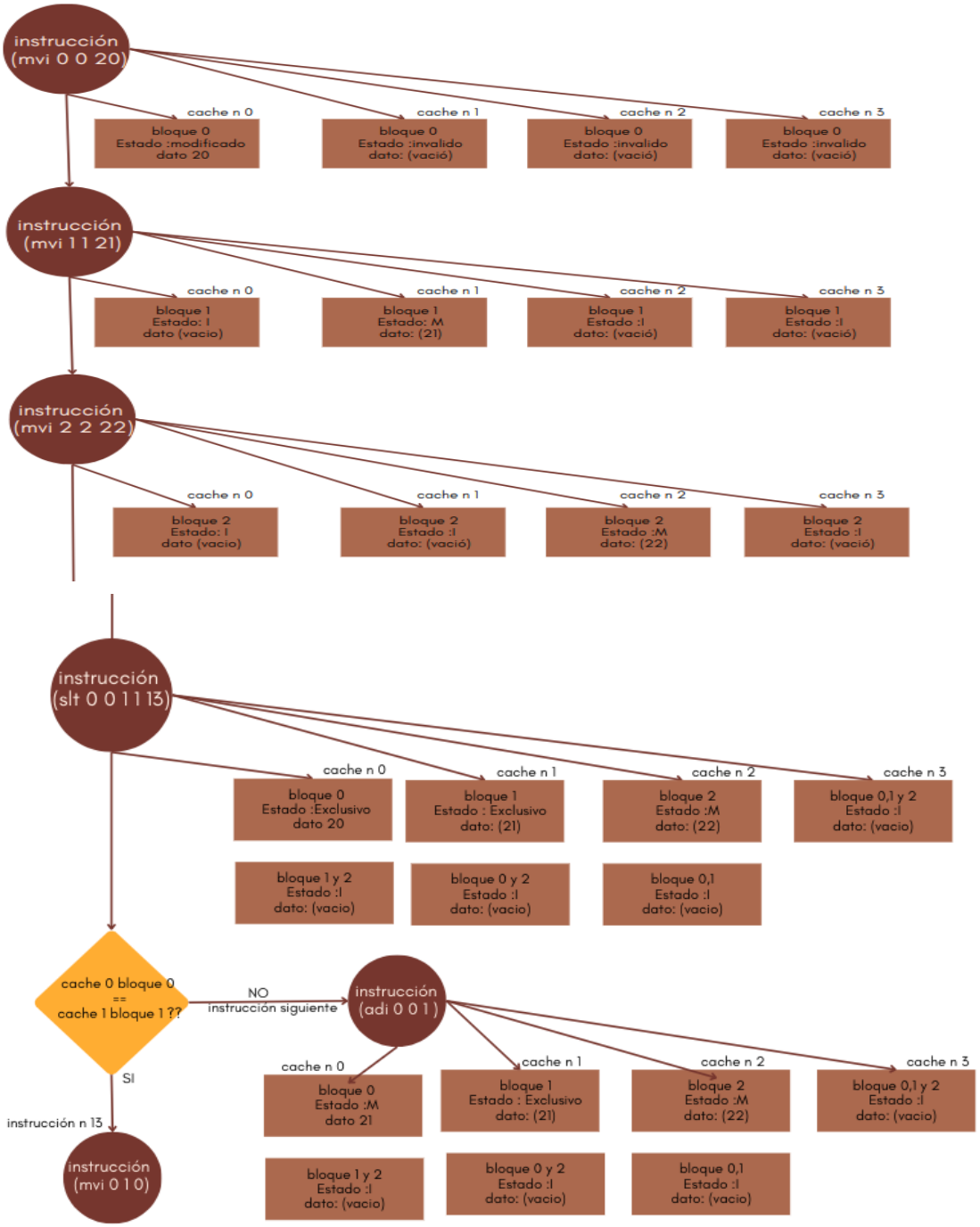
- *Observación en la programación:*

Como el protocolo MESI se puede representar por medio de un autómata por lo que en un principio consideré usar una matriz de estados para resolver el problema, sin embargo, noté

que los estados solo cambian cuando se leen los bloques. Por esta razón, decidí utilizar una estructura de control basada en cuatro condicionales if en lugar de la matriz de estados. Esta solución resultó más sencilla ya que cuando se escribe en un bloque, todos los demás bloques se invalidan, lo que significa que solo se necesita cambiar el estado del bloque en el que se está escribiendo (que pasa a un estado modificado). Como resultado, la cantidad de estados posibles se reduce a la mitad, y la solución utilizando if resultó ser más simple y eficiente en este caso.

3.1 - **Diagrama de flujo**

A modo de ejemplo a continuación agrego el diagrama de flujo de algunas instrucciones del programa:



3.2-Ejemplo de prueba

Las siguientes imágenes son un caso de ejemplo.
Las tres primeras líneas de código del programa realizan tres escrituras en los bloques de memoria de las caches 0 , 1 y 2.

```
1    mvi 0 0 20
2    mvi 1 1 21
3    mvi 2 2 22
4    slt 0 0 1 1 13
5    adi 0 0 1
6    adi 2 2 1
7    div 3 3 0 0 1 1
8    mov 1 1 0 1
9    jmp 11
10   mov 2 2 0 2
11   mvi 3 2 24
12   mvi 0 0 0
13   mvi 0 1 0
14   mvi 0 2 0
15   mov 0 1 2 2
```

El estado de las caches debería cambiar a modo M(modificado) y contener los datos 20, 21 y 22.El resultado esperado en la terminal de comandos del programa debería ser el siguiente:

```
mvi 0 0 20 i:1
{ num='0', estado='M', dato='20', direc='000'}
{ num='1', estado='I', dato='000', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='000', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='000', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='000', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}

mvi 1 1 21 i:2
{ num='0', estado='M', dato='20', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='M', dato='21', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='000', direc='010'}
```

```
mvi 2 2 22 i:3
{ num='0', estado='M', dato='20', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='M', dato='21', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='M', dato='22', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}
```

Además de lecturas y escrituras sobre las memorias se pueden realizar saltos condicionales (en caso de igualdad de dos datos) e incondicionales a una línea o instrucción específica del programa.
Como vemos en el ejemplo anterior cada instrucción va acompañada de las siglas i:n, donde el número referenciado es el línea por donde se va ejecutando el pseudocódigo, esto se implementó para un mejor seguimiento de la traza del programa.
A modo de ejemplo de salto condicional se adjunta la siguiente imagen:

```
slt 0 0 1 1 13 i:4
lectura Valida Modificado reciente
lectura Valida Modificado reciente
4
{ num='0', estado='E', dato='20', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='E', dato='21', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='M', dato='22', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}
```

Esta instrucción recibe 5 parámetros (cache, bloque de las dos registros que se desean comparar), el último parámetro, en este ejemplo el número(13), actúa como etiqueta del programa. En caso de que la comparación sea verdadera, el programa salta al número de línea del programa que indica la etiqueta. En caso contrario, en que la comparación es falsa el programa sigue con la siguiente línea del código en este caso la línea 5.
Vemos también en el ejemplo que el estado de los registros de las memorias 0 y 1, pasa a un estado E(exclusivo), esto significa que se realizó una sola lectura de cada registro, lo cual es correcto porque son los datos que se están comparando con esta instrucción.

Como en este caso la comparación de los datos es falsa, el curso normal del programa debería continuar en la línea 5.

```
adi 0 0 1 i:5
lectura valida Exclusivo valor de lectura 20
{ num='0', estado='M', dato='21', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='E', dato='21', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='M', dato='22', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}
```

En la línea 5 vemos una instrucción sencilla la cual suma un valor en este caso 1 al registro que indican sus otros dos parámetros (cache 0, bloque 0). Vemos en la traza como el estado cambia nuevamente al valor M y se suma correctamente uno al dato.

Como último ejemplo vemos un salto incondicional

```
jmp 11 i:9
intruccion de salto, linea: 11
mvi 3 2 24 i:11
{ num='0', estado='M', dato='21', direc='000'}
{ num='1', estado='M', dato='21', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='I', dato='', direc='010'}

{ num='0', estado='I', dato='', direc='000'}
{ num='1', estado='I', dato='', direc='001'}
{ num='2', estado='M', dato='24', direc='010'}
```

Esta instrucción no realiza ninguna lectura ni escritura, salta a la posición indicada en este caso la línea 1.
La línea 11 realiza un escritura inmediata a la caché número 3 en el bloque dos. Apreciamos el nuevo dato y el cambio de estado, también se observa la invalidación del dato de la cache 2 porque el dato ya no es coherente con la nueva modificación.

4-Conclusiones y recomendaciones:

Luego de llevar a cabo el proceso de desarrollo de la aplicación educativa, pude concluir que he logrado crear una herramienta útil y efectiva para facilitar el aprendizaje y afianzar el concepto de coherencia de memoria y protocolo MESI como estudiante universitario en la materia Arquitectura de computadora.

La aplicación simula situaciones reales de fácil visualización y modificación, permitiendo aplicar y poner en práctica mis conocimientos de manera segura y controlada, lo que me proporciona una experiencia de aprendizaje más enriquecedora, afianzando mis conocimientos en la materia de una forma efectiva.

Además, durante el desarrollo de la aplicación se han identificado oportunidades para su mejora y expansión, como la incorporación de nuevas funcionalidades y la creación de nuevos escenarios de aprendizaje.Una de estas mejoras consiste en la incorporación de nuevas instrucciones assembler, lo que permitiría a los usuarios realizar programas más complejos y aprovechar al máximo el simulador. Asimismo, se podría considerar la simulación de una memoria RAM, lo que proporciona una fuente de datos realista para las operaciones de lectura y escritura, así también como la incorporación de un bus. Aprovechando más la programación orientada a objetos que ofrece java la clase bus podría tener métodos para recibir peticiones de datos y recibir datos de una caché o de memoria RAM, mientras que a la clase cache se le agregaría métodos que realizan peticiones y enviar datos al bus.

Otra mejora técnica importante podría ser proporcionar al usuario una salida más interactiva y visual en lugar de simplemente mostrar los resultados en la terminal. Esto podría lograrse mediante la implementación de interfaces gráficas o visualizaciones más intuitivas que permitan al usuario comprender mejor el estado de las cachés.

Además, se podría considerar hacer parametrizable la cantidad de bloques que se simulan en la caché. Esto permitiría a los usuarios ajustar la configuración de la caché de acuerdo a sus necesidades y evaluar el impacto de diferentes tamaños de caché en el rendimiento del sistema.

En conclusión, entiendo que la aplicación educativa ha cumplido con los objetivos planteados en el proyecto y que será una herramienta valiosa para complementar el proceso de enseñanza en nuestra área de especialización en la universidad. Es importante destacar que se puede seguir trabajando en mejoras y actualizaciones continuas, con el objetivo de mantenerla actualizada y ofrecer la mejor experiencia posible para otros posibles usuarios.

5-Bibliografía

https://es.wikipedia.org/wiki/Protocolo_MESI
https://www.youtube.com/watch?v=ndyKrPMUqwE
https://prezi.com/njgjfmb-wo6u/coherencia-cache-y-protocolo-mesi/