Arquitectura de Computadores II

CE-4302

Grupo 01

Proyecto 1

Modelo de protocolo para coherencia de caché en sistemas multiprocesador

Alumno:

Fabián Ramírez Arrieta - 2018099536

I Semestre 2023

Índice

Índice.....1
Listado de requerimientos del sistema.....2
Opciones de solución del problema.....3
Valoración de opciones de solución.....5
Selección de la propuesta final.....8
Diseño de la alternativa seleccionada.....9
Validación del diseño.....20
Bibliografía.....23

Listado de requerimientos del sistema

La creciente demanda de sistemas multiprocesador eficientes y de alto rendimiento ha llevado al desarrollo de protocolos de coherencia de caché, como MOESI, que facilitan la comunicación y sincronización entre procesadores. Para garantizar que el diseño e implementación de una aplicación que simula el protocolo MOESI en un sistema multiprocesador se adapte a las necesidades y expectativas de los usuarios así también como a la sociedad en general, es fundamental abordar diversos requerimientos y consideraciones en áreas como la salud, seguridad pública, costos, medio ambiente entre otros factores relevantes.

Esta sección presenta un listado de requerimientos del sistema que abarcan aspectos clave, incluyendo eficiencia energética, accesibilidad, seguridad, compatibilidad, escalabilidad, sostenibilidad, responsabilidad social y cultural, e impacto ambiental. Al atender estos requerimientos, el proyecto no solo proporcionará una solución tecnológica efectiva, sino que también contribuirá al bienestar y desarrollo sostenible de la sociedad y el medio ambiente. Estos requerimientos ayudarán a guiar el diseño, implementación y distribución de la aplicación, asegurando que se adapte de manera adecuada a un problema complejo de ingeniería, en línea con las expectativas y demandas actuales. A continuación se puntualizan cada una de las áreas importantes para diseñar la solución del proyecto:

- **Eficiencia energética:** La solución debe ser diseñada teniendo en cuenta el consumo energético y recursos del sistema. Esto implica optimizar el código y las operaciones para reducir el consumo de energía y recursos computacionales.
- **Accesibilidad:** La solución debe ser fácil de usar y accesible para usuarios con diferentes habilidades y conocimientos técnicos. Esto incluye el uso de una interfaz gráfica de usuario (GUI) simple e intuitiva.
- **Compatibilidad:** La aplicación debe ser compatible con diferentes sistemas operativos y arquitecturas de hardware. Esto facilitará su uso en diversos entornos y dispositivos, aumentando su alcance y versatilidad.

Opciones de solución del problema

Para definir las opciones de solución del problema como un todo es de suma importancia dividir la solución en distintas áreas que poseen distintas opciones compatibles entre sí para determinar qué herramientas son las más adecuadas para formar parte de la solución final. A continuación se enumeran las opciones que se contemplaron a la hora de implementar la solución:

- **Lenguaje de programación:** Sin lugar a dudas esta es una de las decisiones de diseño más importantes a tomar. Se tenía claro que el programa a implementar tenía dos grandes áreas. La parte lógica y la parte gráfica. A pesar de que se pueden crear sistemas multilenguaje, en este caso se quería un lenguaje que permitiera crear ambas áreas de manera eficiente. Se barajaron las siguientes opciones:
    - **Python:** Uno de los lenguajes más famosos de la actualidad. En cuanto a lo gráfico posee la librería Tkinter, utilizada para crear aplicaciones gráficas de escritorio.
    - **Java:** También es un lenguaje con mucha popularidad. Es un lenguaje orientado a objetos. En cuanto a la creación de la aplicación gráfica, posee una librería muy potente, como lo es JavaFX.
- **Función de probabilidad formal:** Un requisito fundamental de diseño para este proyecto era la elección de una función de probabilidad formal para la generación aleatoria de instrucciones. En este caso se evaluaron las siguientes soluciones:
    - **Distribución Uniforme:** Es una distribución en la que todos los resultados tienen la misma probabilidad de ocurrir. Es fácil de implementar y es adecuada cuando no hay preferencias o conocimientos previos sobre las probabilidades de los eventos.
    - **Distribución de Poisson**: Modela el número de eventos que ocurren en un intervalo fijo de tiempo o espacio, donde los eventos son raros y aleatorios. Es apropiada para situaciones en las que se desea contar eventos independientes que ocurren a una tasa promedio constante.
    - **Distribución Binomial:** Describe la probabilidad de un número fijo de éxitos en una serie de ensayos independientes de Bernoulli, cada uno con una probabilidad constante de éxito. Es útil para modelar situaciones donde se tienen eventos dicotómicos con una probabilidad de éxito conocida.

- **Distribución Hipergeométrica:** Modela la cantidad de éxitos en una muestra tomada de una población finita sin reemplazo. Es adecuada para situaciones en las que se desea contar el número de éxitos en una muestra extraída de una población con una cantidad fija de éxitos y fracasos.

Valoración de opciones de solución

En la siguiente sección, se analizarán los dos aspectos clave anteriormente mencionados para la implementación exitosa del proyecto: la selección del lenguaje de programación y la función de probabilidad formal. La elección del lenguaje de programación influirá en la facilidad de desarrollo, accesibilidad, eficiencia energética y la compatibilidad, mientras que la función de probabilidad formal influirá en el comportamiento del sistema implementado, lo que afectará cómo se comportará el protocolo MOESI simulado.

Lenguaje de programación:

- **Java (JavaFX):**
  - Pros:
    - **Rendimiento:** Java generalmente ofrece un mejor rendimiento que Python debido a la compilación Just-In-Time y optimizaciones adicionales.
    - **Portabilidad:** JavaFX permite crear aplicaciones multiplataforma de forma nativa, sin depender de librerías externas.
    - **Estilo y apariencia:** JavaFX ofrece una mayor variedad de controles y opciones de estilo para personalizar la apariencia de la aplicación.
    - **Comunidad y soporte:** Java tiene una gran comunidad de desarrolladores y amplio soporte en línea, lo que facilita encontrar soluciones a problemas específicos.
  - Contras:
    - **Curva de aprendizaje:** Java puede ser más difícil de aprender para principiantes debido a su sintaxis más rígida y estructuras más complejas.
    - **Verbosidad:** El código en Java tiende a ser más extenso, lo que puede dificultar su mantenimiento y comprensión.
    - **Tiempo de desarrollo:** El desarrollo en Java suele ser más lento que en Python debido a su complejidad y verbosidad.

- **Python:**
    - Pros:
        - **Facilidad de uso:** Python es conocido por su sintaxis simple y legible, lo que facilita el aprendizaje y la escritura de código.
        - **Tiempo de desarrollo:** Python permite un desarrollo más rápido debido a su simplicidad y facilidad de uso.
        - **Amplio soporte para bibliotecas científicas:** Python cuenta con un ecosistema de bibliotecas científicas y de análisis de datos muy sólido, lo que lo hace adecuado para proyectos que requieran cálculos matemáticos y estadísticos.
        - **Comunidad y soporte:** Python tiene una comunidad activa y creciente, con numerosos recursos y tutoriales en línea disponibles.
    - Contras:
        - **Rendimiento:** Python suele tener un rendimiento más bajo que Java debido a su naturaleza interpretada.
        - **Estilo y apariencia:** Tkinter ofrece menos opciones de personalización y un aspecto menos moderno en comparación con JavaFX.

Por otro lado, con respecto a la función de probabilidad formal, tenemos los siguientes aspectos a considerar:

- **Distribución Uniforme:**
    - Pros:
        - Simple de entender e implementar.
        - Adecuada para situaciones en las que todos los resultados son igualmente probables.
    - Contras:
        - No es apropiada para modelar eventos con diferentes probabilidades de ocurrencia.
        - Puede no ser representativa de situaciones reales donde ciertos eventos son más probables que otros.

- **Distribución de Poisson:**
    - Pros:
        - Adecuada para modelar eventos raros y aleatorios en un intervalo fijo de tiempo o espacio.
        - Útil para contar eventos independientes que ocurren a una tasa promedio constante.
    - Contras:
        - No es aplicable a eventos con dependencia temporal o espacial.
        - Requiere conocer la tasa promedio de eventos para ser precisa.
- **Distribución Binomial:**
    - Pros:
        - Útil para modelar eventos dicotómicos con una probabilidad de éxito conocida.
        - Adecuada para situaciones donde se tienen ensayos independientes de Bernoulli.
    - Contras:
        - No es aplicable a eventos con más de dos resultados posibles.
        - Requiere conocer la probabilidad de éxito y el número de ensayos para ser precisa.
- **Distribución Hipergeométrica:**
    - Pros:
        - Adecuada para contar éxitos en una muestra extraída de una población finita sin reemplazo.
        - Permite modelar situaciones donde se desea analizar la relación entre subconjuntos de una población.
    - Contras:
        - No es aplicable a muestreos con reemplazo.
        - Requiere conocer la cantidad de éxitos y fracasos en la población y el tamaño de la muestra para ser precisa.

Selección de la propuesta final

La elección del lenguaje de programación adecuado para este proyecto es crucial, ya que puede influir en factores como la facilidad de desarrollo, el rendimiento, y la compatibilidad. En el caso de este proyecto, que implica la simulación del protocolo MOESI en un sistema multiprocesador mediante una aplicación con interfaz gráfica, es fundamental seleccionar un lenguaje y librería que permitan un desarrollo ágil y eficiente, al tiempo que cumpla con los requerimientos del sistema previamente establecidos. Después de analizar las ventajas y desventajas de Java (JavaFX) y Python (Tkinter), se ha decidido optar por Python como el lenguaje de programación para este proyecto.

Existen varias razones para seleccionar Python en este caso. En primer lugar, Python es conocido por su simplicidad y facilidad de uso, lo que permite un desarrollo más rápido y eficiente. Dado que el objetivo del proyecto no se centra en la apariencia estilizada de la aplicación, sino en la funcionalidad y precisión del protocolo MOESI simulado, la limitada personalización de Tkinter no debería ser un inconveniente importante. Por otro lado, como se mencionó anteriormente, el desarrollador posee mayor experiencia en el uso de Python y Tkinter, lo que permitirá una implementación más eficiente y efectiva de la solución, aprovechando el conocimiento y habilidades previas en estas tecnologías. En conjunto, estos factores hacen que Python y Tkinter sean una opción adecuada para el desarrollo de este proyecto, cumpliendo con los requerimientos del sistema y facilitando la entrega de una solución eficaz y fácil de usar.

Por otro lado, la selección adecuada de una distribución estadística es de suma importancia en la resolución del problema planteado ya que permite modelar y predecir eventos de manera precisa y confiable. Una distribución que se ajuste bien al contexto del problema permitirá obtener un comportamiento adecuado en la generación de instrucciones.

La Distribución Uniforme es adecuada para la generación de los tres tipos de instrucciones (CALC, READ y WRITE) en este caso particular, ya que se asume que cada instrucción es igualmente probable y no hay preferencia o conocimiento previo sobre las probabilidades de cada evento. Al utilizar la Distribución Uniforme, se garantiza que todas las instrucciones tienen la misma probabilidad de ser generadas, lo que proporciona una representación justa y equilibrada de las instrucciones en el conjunto de datos generado. Además, esta distribución es simple de entender e implementar, lo que facilita su uso en el proyecto y permite un análisis directo de los resultados.

Diseño de la alternativa seleccionada

En esta sección se iniciará por la distribución estadística utilizada. Como se definió anteriormente, la distribución utilizada fue la Distribución Uniforme, ya que esta permite darle la misma probabilidad a los tres tipos de instrucciones de ser generados. Dando como resultado la siguiente función:

```python
def newInstruction(self):
    def binary(num, length=4):
        return format(num, f"0{length}b")

    def hexa(num, length=4):
        return format(num, f"0{length}X")

    # Distribución de probabilidad
    p_calc = 1 / 3
    p_write = 1 / 3

    r = random.random()

    if r < p_calc:
        return f"P{self.id}: CALC"
    elif r < p_calc + p_write:
        addr = binary(random.randint(0, 7), 3)
        data = hexa(random.randint(0, 65535))
        return f"P{self.id}: WRITE {addr} {data}"
    else:
        addr = binary(random.randint(0, 7), 3)
        return f"P{self.id}: READ {addr}"
```

Figura 1. Código generador de instrucciones.

En la figura anterior se puede observar como hay una función que recibe un número binario y le da un formato específico según la longitud y lo devuelve en formato binario. Este valor corresponde al valor de memoria principal al cual se le hará la escritura o lectura según sea el caso. Luego bajo esta misma línea, está la función hexa que devuelve un número hexadecimal. Este valor se utiliza en la función Write, debido que es la única instrucción que necesita de este valor extra.

Luego, se definen las probabilidades p_calc y p_write para las instrucciones CALC y WRITE, respectivamente, asignándoles 1/3 cada una. Esto deja implícita una probabilidad de 1/3 para la instrucción READ. Se genera un número aleatorio r entre 0 y 1 utilizando random.random().

Se evalúa el valor de r para determinar qué instrucción generar:

- Si r es menor que p_calc, se genera una instrucción CALC, la cual es una instrucción de procesamiento. En términos prácticos, se ha definido que esta instrucción tomará un ciclo de reloj del programa para ser ejecutada.
- Si r es mayor o igual a p_calc pero menor que p_calc + p_write, se genera una instrucción WRITE. Primero, se crea una dirección de memoria aleatoria binaria de 3 bits (addr) y un dato hexadecimal de 4 dígitos (data). Luego, se devuelve la instrucción con el formato "P{id}: WRITE {addr} {data}".
- Si no cumple ninguna de las condiciones anteriores, se genera una instrucción READ. Se crea una dirección de memoria aleatoria binaria de 3 bits (addr) y se devuelve la instrucción con el formato "P{id}: READ {addr}".

Para mayor claridad, el id es un identificador para cada procesador, de esta manera se mantiene la trazabilidad sobre qué procesador generó esta instrucción. Es utilizado mayormente en cuestiones de debug.

Ahora bien, por el lado la interfaz gráfica el diseño de la misma se basó completamente en la estructura que se puede ver en la figura 2. Este es una estructura que se utiliza didácticamente para enseñar cómo funciona un sistema multiprocesador y la coherencia de caché.

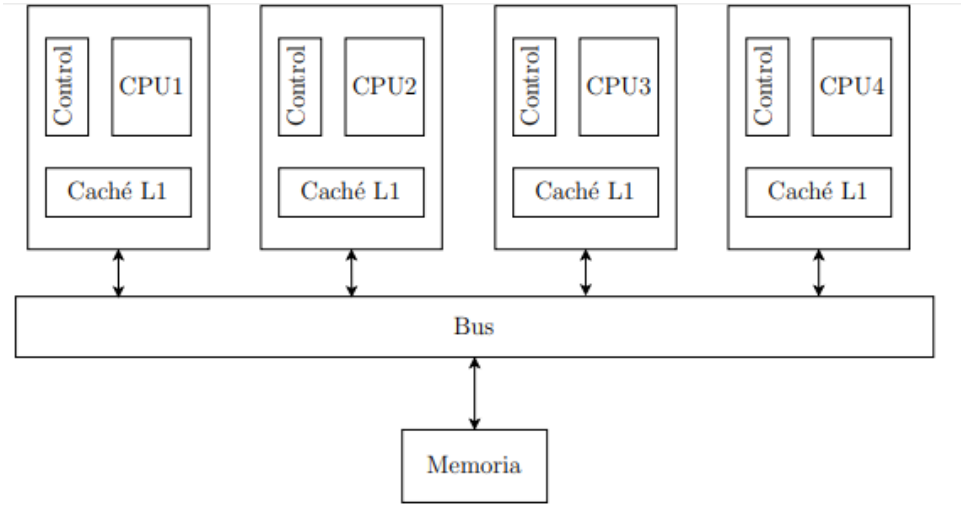

Figura 2. Arquitectura general del sistema.

A partir de esta estructura, se comenzó por crear una estructura a modo de boceto en la página web Figma para un mayor orden de cómo se crearán los frames de cada uno de los componentes en la pantalla principal. Este boceto se puede observar en la figura 3.

Figura 3. Boceto hecho en Figma

Este boceto ayudó a definir las dimensiones de la pantalla, ya que por facilidad en el desarrollo se definió que las mismas iban a ser de dimensiones fijas. En este caso, la pantalla completa es de 900 x 500 píxeles. Luego, los rectángulos superiores serán los encargados de albergar la información de cada uno de los CPUs. El recuadro celeste se pensó para albergar la información del bus, mientras que el rectángulo inferior será el encargado de albergar la información de la memoria del sistema.

Utilizar este sistema es de gran utilidad ya que se acopla de gran forma en como Tkinter ordena los componentes. El primer paso en la creación de la interfaz fue crear los frames que albergan cada uno de los componentes. Al hacerlo de esta forma permite que colocar componentes en común pueda hacerse de manera recursiva ya que cada Frame posee su propio eje de coordenadas.

El resultado de la interfaz ya implementada se puede observar en la figura 4. Acá se puede observar que hay un cambio importante con respecto al boceto de la figura 3, el cual es que no existe el bus como tal a nivel gráfico. Esto se decidió de esta manera debido a que abarcaba mucho espacio gráfico y realmente no desplegaba información relevante para el usuario durante su uso.

Además, de esta interfaz principal, también se programó una ventana emergente para generar instrucciones de manera manual. Esta ventana permite seleccionar a qué procesador se le asignará la nueva instrucción, y hará las verificaciones necesarias según sea la instrucción seleccionada. Esta ventana se puede observar en la figura 5.

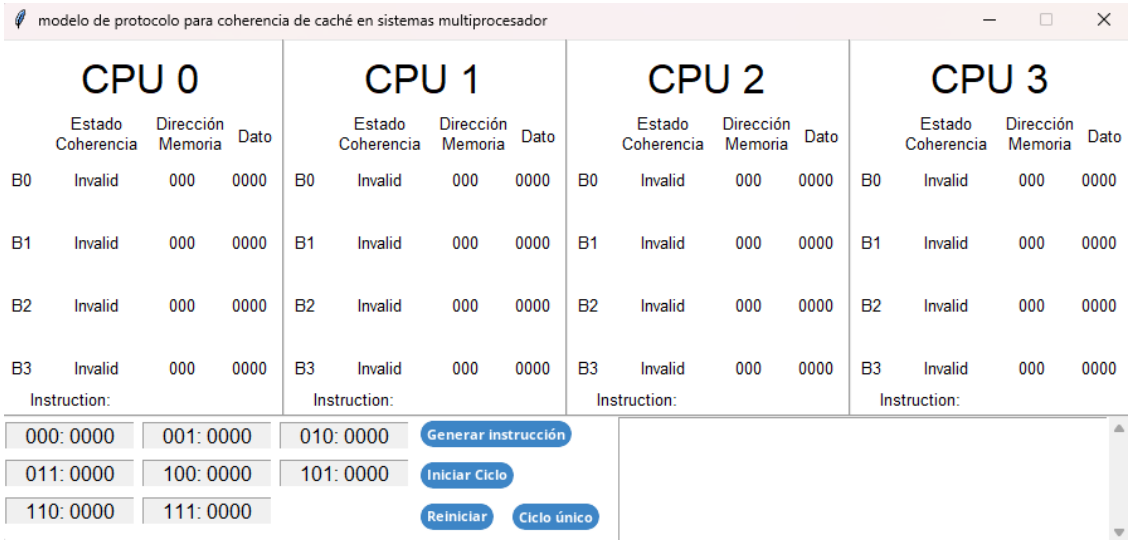

Figura 4. Interfaz gráfica implementada en Tkinter

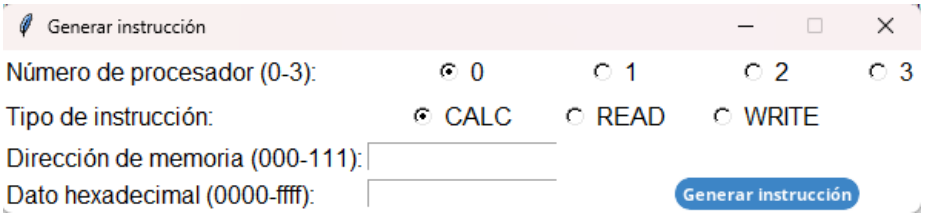

Figura 5. Ventana emergente para generar una instrucción de manera manual

En la zona inferior de la interfaz existen tres diferentes elementos. Al lado izquierdo se puede observar el estado de la memoria principal, el cual se va actualizando en tiempo real, de acuerdo a las operaciones que van realizando los procesadores.

En cuanto a los botones, existen cuatro diferentes botones, los cuales realizan las siguiente funciones:

- **Generar Instrucción:** Despliega la ventana emergente de la figura 5 y permite al usuario generar una instrucción de manera manual, seleccionando todos los parámetros necesarios.
- **Iniciar/Detener Ciclo Ciclo:** Es un botón con una imagen dinámica que permite iniciar la ejecución en bucle. Es decir, cada cierto tiempo, actualmente definido en 2 segundos, se ejecutará un ciclo de reloj de manera indefinida hasta que se decida detener el programa.
- **Reiniciar:** Hace que vuelva el programa a su estado original. Reiniciando las cachés de cada procesador, el estado de la memoria principal y el estado de la consola, el cual se explicará más adelante.
- **Ciclo único:** Este botón permite la ejecución paso a paso. Cada vez que se presiona, se ejecutará un único ciclo de reloj para cada uno de los CPUs. Es importante recalcar que no permite ser utilizada si la ejecución en bucle está activada.

Con respecto al componente de la sección inferior derecha, se trata de una consola que imprime en tiempo real los logs que están teniendo los procesadores. También se le brindó la probabilidad al usuario de exportar estos logs a un archivo de texto para una mayor facilidad de análisis. Esto se puede lograr con el comando ctrl + S.

Por otro lado, con respecto al algoritmo implementado está basado completamente en el protocolo MOESI. Este protocolo se puede modelar como una máquina de estados, tal como se observa en la figura 6. Es importante recalcar que este algoritmo es multiprocesador, por lo que es necesario realizar las validaciones necesarias entre los distintos procesadores en caso de que la misma dirección de memoria esté siendo utilizada en varios procesadores.

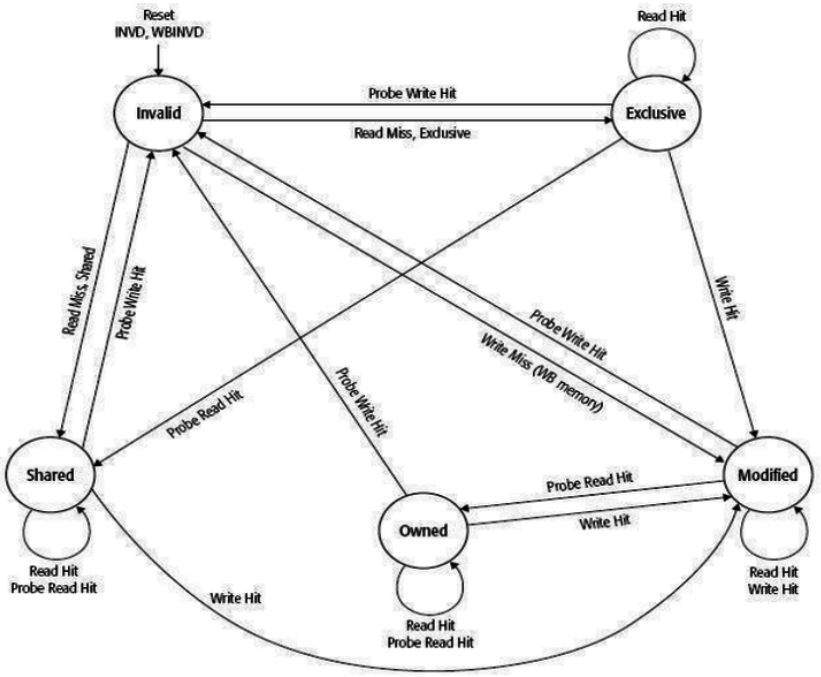

Figura 6. Diagrama de protocolo MOESI.

Una de las ventajas principales de Python como lenguaje de programación es que es un lenguaje multiparadigma, esto quiere decir que permite programar tanto en el paradigma orientado a objetos como en el paradigma funcional. En la mayoría de este proyecto se utilizó el paradigma funcional, sin embargo, es importante recalcar que se crearon dos clases: La clase CPU y la clase Bloque de caché.

La clase CPU posee toda la lógica para ejecutar instrucciones, cada CPU tendrá la capacidad para generar y ejecutar instrucciones de manera independiente. El grueso de su procesamiento está en las instrucciones de READ y WRITE. Además tendrá cuatro bloques de caché, pertenecientes a la clase de bloque de caché, los cuales tendrán la información básica necesaria, como por ejemplo, el estado, la dirección de memoria y el dato a escribir.

A continuación se mostrarán las principales estructuras de datos que han sido utilizadas para el correcto funcionamiento del programa, en su mayoría diccionarios. Además se detalla su uso y el cómo se inicializa cada una:

Memoria principal: Esta se ha modelado como un diccionario, donde el valor de la dirección actúa como llame y el valor de el dato se encuentra en hexadecimal. Inicialmente se setea en cero como una variable global. Se seleccionó este tipo de estructura debido a su fácil acceso a cada valor por medio del valor de su llave, y de esta manera se evitan iteraciones innecesarias para modificar valores. En la figura 7 se observa como se ha definido a nivel de código:

```
# Memoria

memory_dict = {
    "000": 0x0,
    "001": 0x0,
    "010": 0x0,
    "011": 0x0,
    "100": 0x0,
    "101": 0x0,
    "110": 0x0,
    "111": 0x0,
}
```

Figura 7. Inicialización de la memoria

Bus de datos: El bus de datos, es una estructura muy similar a la anterior, tal como se puede observar en la figura 8. Con respecto a las llaves, son las mismas, solamente que el bus como valores posee una lista. Esto se realizó de esta forma a modo de registro, donde el bus conoce qué procesadores tienen en cargado el dato de memoria en sus respectivas caché. En esta lista se almacenarán objetos de tipo Bloque de caché.

Bus de procesadores: Otro diccionario que se utiliza es uno que contiene los cuatro objetos de tipo CPU que se utilizarán durante la ejecución del programa. Este diccionario es importante para poder renderizar la información en pantalla de manera iterativa con la menor cantidad de código posible. Este bus se puede observar en la figura 9. Este diccionario se mantiene constante durante todo el ciclo de vida del programa.

```
# Bus

bus_dict = {
    "000": [],
    "001": [],
    "010": [],
    "011": [],
    "100": [],
    "101": [],
    "110": [],
    "111": [],
}
```

Figura 8. Bus de datos

```
# CPUS
cpu0 = CPU(0)
cpu1 = CPU(1)
cpu2 = CPU(2)
cpu3 = CPU(3)

cpus_dict = {
    "cpu0": cpu0,
    "cpu1": cpu1,
    "cpu2": cpu2,
    "cpu3": cpu3,
}
```

Figura 9. Diccionario de CPUs.

Ahora, profundizando un poco más en el desarrollo del algoritmo, sin lugar a dudas que uno de las mayores retos del proyecto fue la implementación de la ejecución del algoritmo a modo de paso a paso, ya que se identificó que era necesario dividir el proceso de completitud de cada instrucción por partes y que cada una de las partes sea ejecutada en un solo ciclo de reloj. De esta manera se asemeja al delay existente en los procesadores reales cuando un dato cuando se identifican misses en la lectura. Para lograr la implementación del algoritmo se hicieron los siguientes diagramas de flujo:

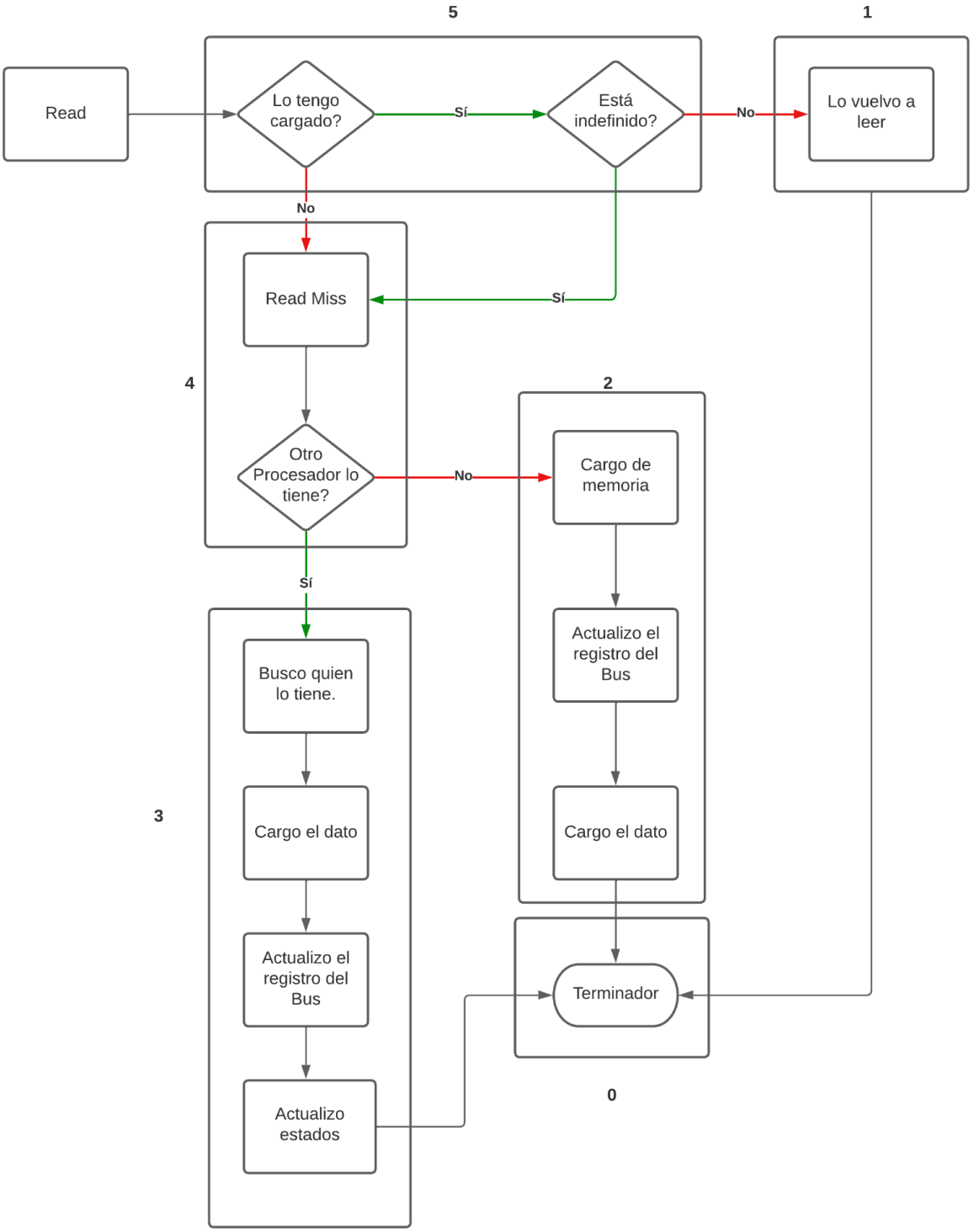

Figura 10. Diagrama de Flujo. Instrucción READ

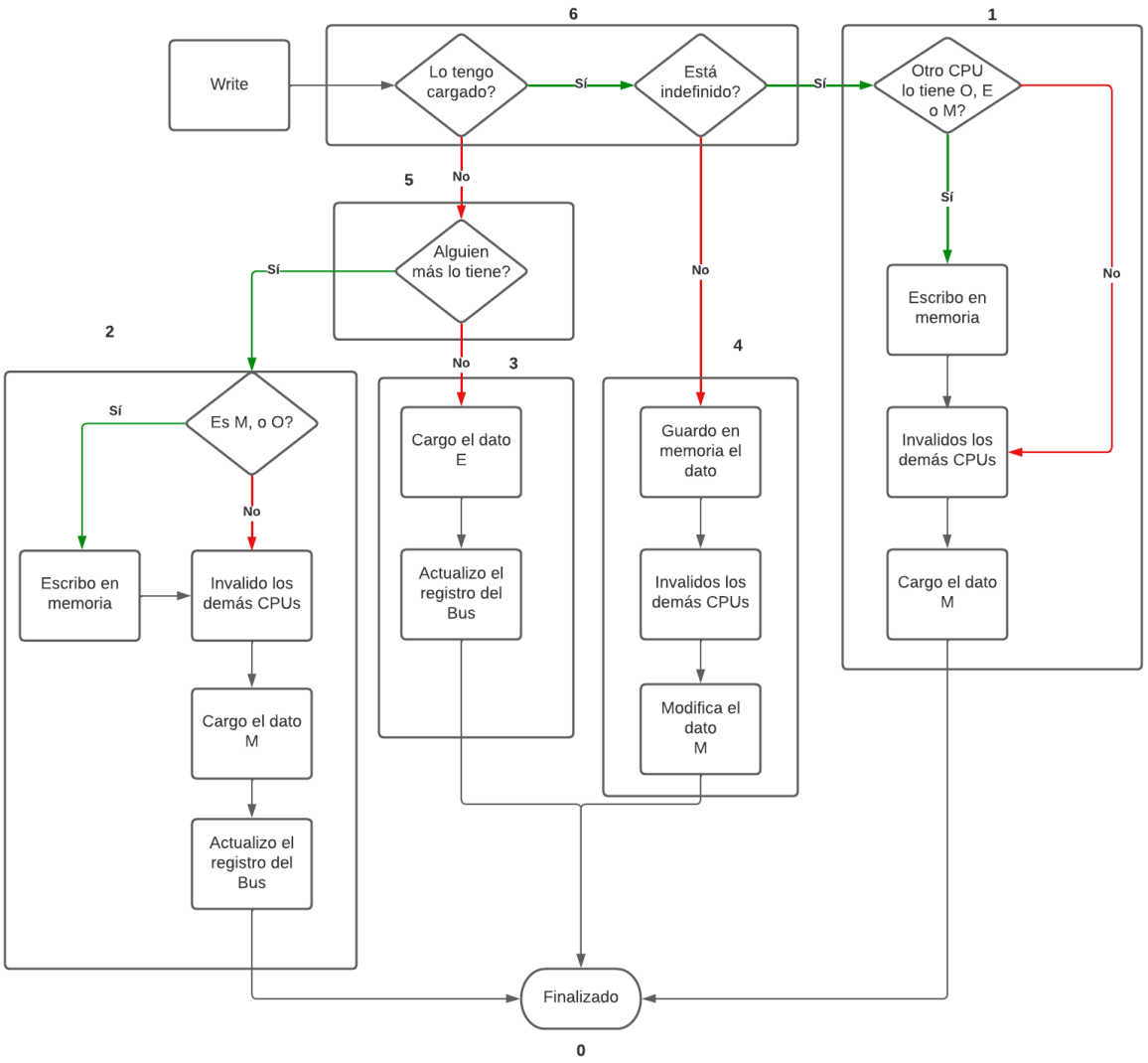

Figura 11. Diagrama de flujo Instrucción WRITE.

Estos diagramas muestran cómo funcionará la ejecución por ciclo de reloj. Cada recuadro funciona como un paso necesario para completar cada una de las instrucciones. Cada objeto de tipo CPU posee dos atributos necesarios para esta lógica, los cuales son el *instruction*, que es un string que almacena la instrucción en ejecución. También posee *instruction_Step*. Este es un integer que almacena el paso por el que se encuentra en ejecución. De este modo sabe por donde se encuentra, independientemente del ciclo de reloj.

Se puede explicar que se hace por cada ciclo de reloj por cada procesador con la explicación de un par de funciones. Inicialmente cada ciclo de reloj se llama a la siguiente función mostrada en la figura 12.

```python
def executeCycle(self):
    if self.instruction == "":
        self.generateNewInstruction()
    else:
        self.executeInstruction()
```

Figura 12. Método para ejecutar instrucción.

Esta es la función ejecutada en cada ciclo de reloj por cada procesador. En caso de que la instrucción sea “” significa que ya no existe instrucción en ejecución en el procesador, por lo que procede a generar una nueva instrucción por medio de la distribución uniforme explicada anteriormente. En caso de que haya una instrucción en ejecución, se pasa a la función de ejecutar instrucción, la cual se puede observar en la figura 13.

```python
def executeInstruction(self):
    # 1 = calc
    # 2 = read
    # 3 = write
    inst_type = len(self.instructionList)

    # Calc
    if inst_type == 1:
        printToConsole(f"CPU{self.id}: Calculando.")
        self.instruction_Step = 0
    if self.instruction_Step == 0:
        self.instruction = ""
    # Read
    elif inst_type == 2:
        mem_dir = self.instructionList[1]
        self.executeRead(mem_dir)
    # Write
    else:
        mem_dir = self.instructionList[1]
        data = self.instructionList[2]
        data = int(data, 16)
        self.executeWrite(mem_dir, data)
```

Figura 13. Función para ejecutar una instrucción.

En la figura anterior se observa como la tarea de esta función es identificar el tipo de instrucción que se encuentra en ejecución. En caso de una instrucción de tipo CALC solamente utiliza un ciclo de reloj para “calcular”, y luego define en el *instruction_step* en 0 para dar por finalizada la instrucción y de esta forma eliminar la instrucción en ejecución a “” para generar una nueva instrucción en el siguiente ciclo de reloj.

En caso de que se dé una instrucción de tipo WRITE o READ, toma los datos necesarios para la ejecución y llama a una función principal de cada tipo de instrucción. En este caso profundizaremos con myor detalle en el caso del WRITE, sin embargo, ambas funcionan igual, lo que cambia es que cada una sigue el orden de su respectivo diagrama. La función *executeWrite* se puede observar en la figura 14.

```python
def executeWrite(self, mem_dir, data):
    # Checkea si es un writeMiss o si tengo el dato
    if self.instruction_Step == 6:
        printToConsole(
            f"CPU{self.id}: Ejecutando instruccion: {self.instruction[4:]}"
        )
        self.write_CheckMiss(mem_dir)
        return
    # No tengo el dato, checkeo si alguien más lo tiene
    elif self.instruction_Step == 5:
        printToConsole(f"CPU{self.id}: Write miss")
        self.write_missedCheckOthers(mem_dir)
        return

# Tengo el dato pero no está invalidado. Si lo tengo pero es S, E, M o O, puedo modificarlo de nuevo, solam
ente invalidando el resto de ser necesario
    elif self.instruction_Step == 4:
        self.write_notMissedNotInvalid(mem_dir, data)
        return
    # Nadie tiene el dato, lo cargo de memoria
    elif self.instruction_Step == 3:
        self.write_NobodyHasIt_LoadFromMemory(mem_dir, data)
        return
    # Alguien tiene el dato, lo cargo de otro CPU
    elif self.instruction_Step == 2:
        self.write_SomebodyHasIt_LoadFromThere(mem_dir, data)
        return
    # Tengo el dato pero está invalidado. Doy por hecho que alguien más lo tiene porque me invalidó mi dato
    elif self.instruction_Step == 1:
        self.write_notMissedButInvalid(mem_dir, data)
        return
    # Doy por finalizada la ejecución
    elif self.instruction_Step == 0:
        self.instruction == ""
        return
```

Figura 14. Función ExecuteWrite.

La figura anterior puede parecer un poco confusa y compleja a primera vista, sin embargo, si se mira en compañía del diagrama de la figura 11 se puede observar como cada comparativa donde busca el valor del atributo *instruction_step*, coincide con los pasos expuesto en el diagrama. Por lo tanto, las funciones auxiliares a las que se llaman dentro de esta función principal solamente realizan los pasos explicados uno a uno en este diagrama. De esta forma se consigue la ejecución paso a paso. A nivel interno hay varias validaciones que se deben hacer para su correcto funcionamiento.

Validación del diseño

Finalmente, se puede comprobar la solución implementada ejecutando una serie de casos claves y luego verificar si el estado de los bloques de caché es el esperado según el protocolo MOESI.

Primeramente se verificará el funcionamiento de la función de probabilidad. Se harán tres pruebas para generar una instrucción en cada uno de los procesadores y de esta manera observar si la posibilidad de que se presente cada una de las instrucciones es realmente igual.

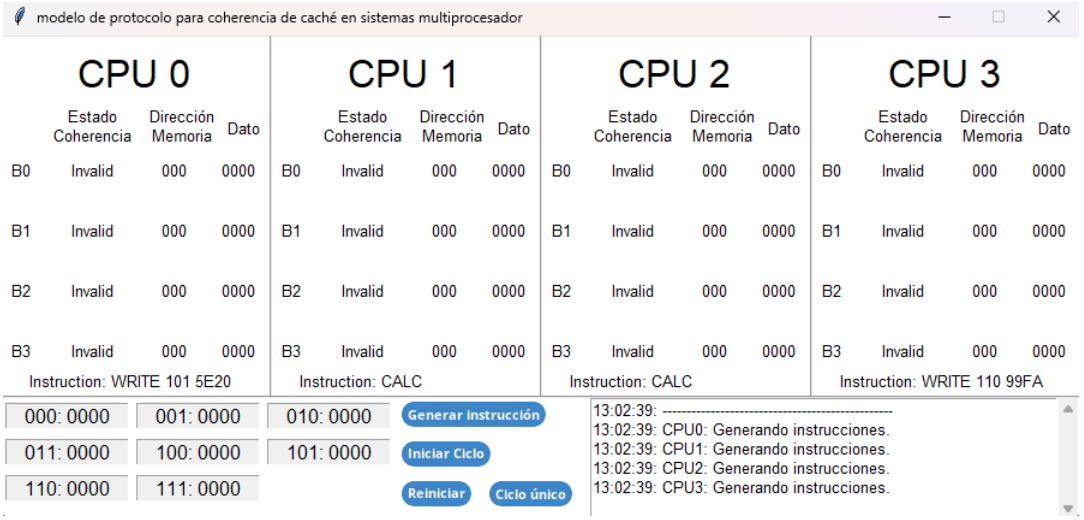

Figura 15. Primera prueba de generación de instrucciones.

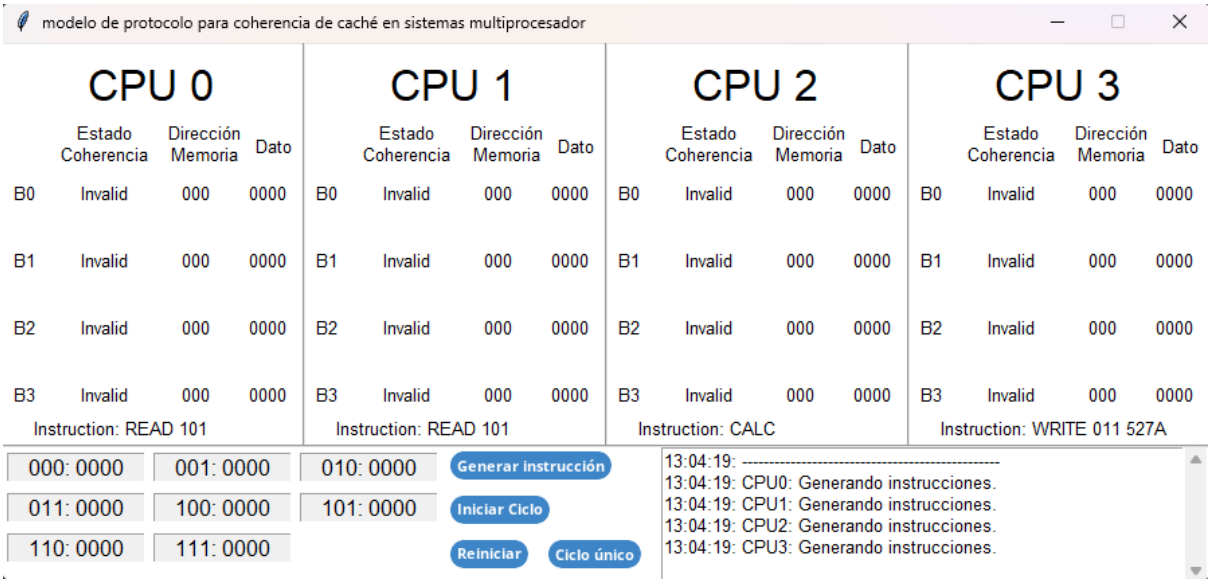

Figura 16. Segunda prueba de generación de instrucciones.

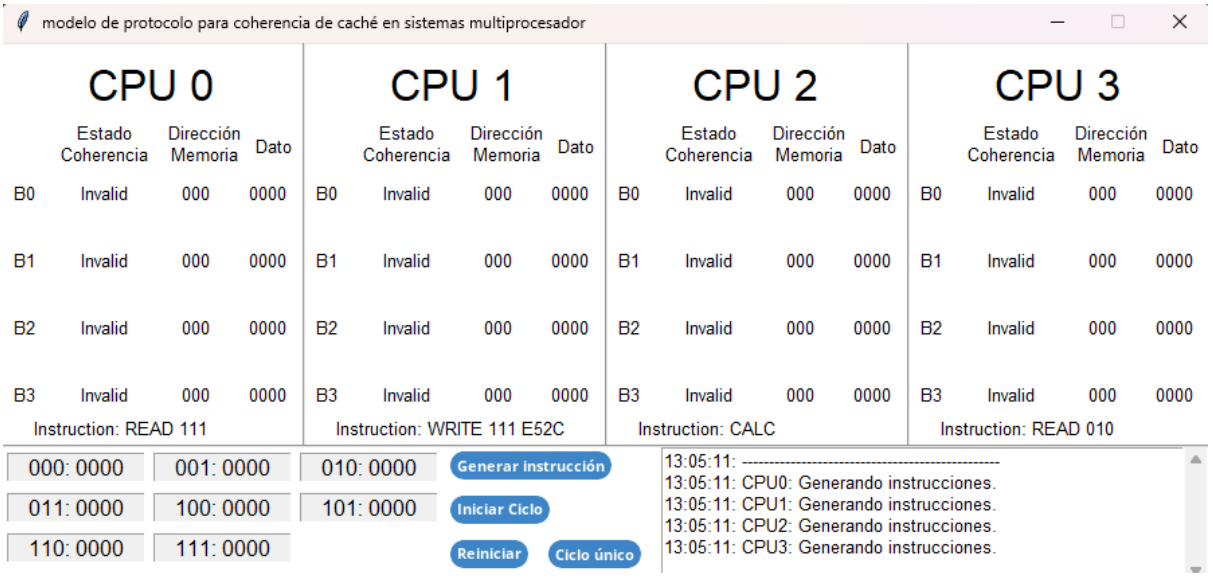

Figura 17. Tercera prueba de generación de instrucciones.

Como se puede observar, en las pruebas de las figuras 15, 16 y 17, de un total de 12 instrucciones generadas se generaron cuatro instrucciones de tipo CALC, cuatro instrucciones de tipo READ y cuatro instrucciones de tipo WRITE. En este experimento se mantiene la proporción de manera esperada, sin embargo, puede ocurrir que por pequeñas variaciones no ocurra, siempre dentro de un margen de error aceptable.

A continuación se mantendrá la ejecución en ciclo durante 2 minutos para comprobar si los estados de caché resultantes son coherentes con el protocolo MOESI.

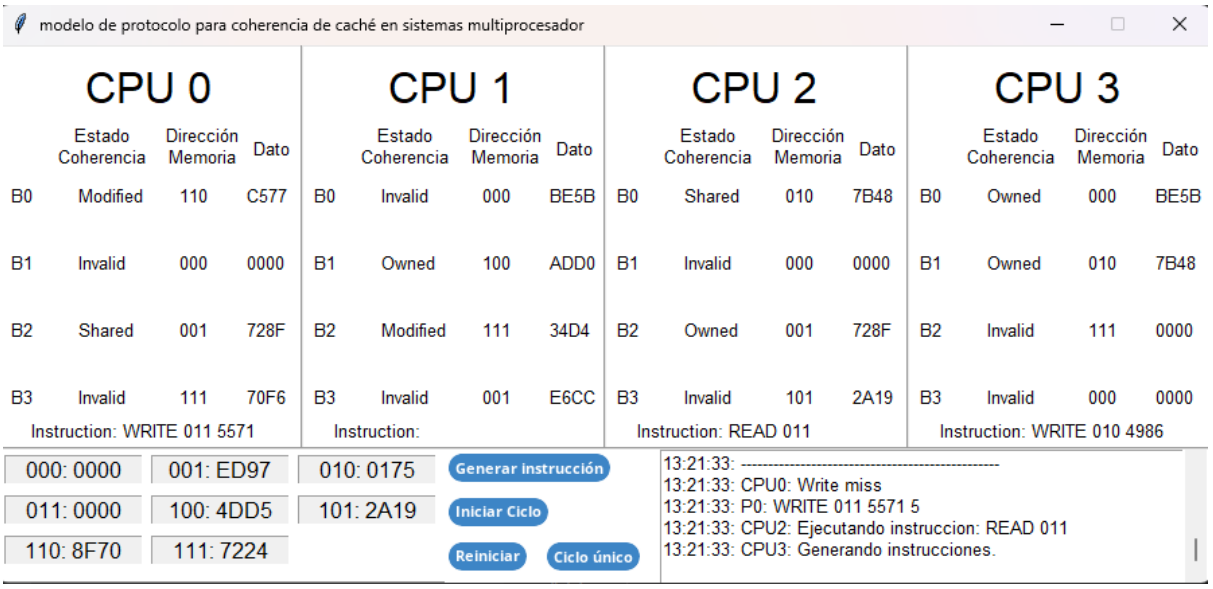

Figura 18. Resultado luego de dos minutos de ciclo.

En la figura 18 se logra observar como luego de dos minutos de ejecución, el protocolo MOESI ha sido aplicado de manera satisfactoria. Esto se puede observar al analizar

varios elementos presentes en la anterior figura. Por ejemplo, el primer detalle a recalcar es el estado de la memoria principal y hacer notar cómo las distintas instrucciones ejecutadas han modificado varias direcciones de la memoria principal.

Por otro lado, otro ejemplo de su funcionamiento es el caso de la memoria 001. El CPU2 posee esta instrucción en el estado Owned, con un estado de 0x728F. En el CPU0 se hizo un READ de este valor, obteniendo el valor que el CPU0 modificó y además el CPU1 posee un valor inválido de esta dirección de memoria, el cual es 0xE6CC, por lo que en caso tener que leerlo de nuevo, deberá actualizar su valor con el dato que el CPU2 estableció.

En conclusión, una implementación adecuada del protocolo MOESI en este proyecto permite cumplir con los requerimientos del sistema previamente establecidos en la sección *Listado de requerimientos del sistema*, como la eficiencia energética, accesibilidad y la compatibilidad. Al garantizar la coherencia de caché en un sistema multiprocesador simulado, se logra un uso eficiente de los recursos computacionales, lo que, a su vez, contribuye a la reducción del consumo de energía y el impacto ambiental. Además, al desarrollar una aplicación con una interfaz gráfica de usuario intuitiva y accesible, se promueve la inclusión y equidad en el acceso a la tecnología, respetando la diversidad cultural y las necesidades de los usuarios.

Es fundamental tomar en cuenta estos requerimientos a la hora de diseñar la solución, ya que no solo garantizan que el proyecto cumpla con las expectativas técnicas, sino que también aseguran que la aplicación contribuya al bienestar y desarrollo sostenible de la sociedad y el medio ambiente. Al abordar estos aspectos en el diseño e implementación del proyecto, se crea una solución integral que va más allá de la simple funcionalidad técnica y se alinea con las demandas y responsabilidades actuales en el ámbito de la ingeniería y la tecnología. En última instancia, al considerar estos requerimientos, se logra entregar una solución que satisfaga las necesidades de los usuarios y promueva un impacto positivo en el mundo en general.

Bibliografía

1. OpenStax. Distribución de Poisson. (2022). Recuperado de Distribución de Poisson.

2. OpenStax. Distribución Uniforme. (2022). Recuperado de Distribución Uniforme.

3. OpenStax. Distribución Binomial. (2022). Recuperado de Distribución binomial.

4. OpenStax. Distribución Hipergeométrica. (2022). Recuperado de Distribución hipergeométrica.

5. IBM. Distribución estadística. (2023). Recuperado de Distribución estadística.

6. Python. tkinter. Python interface to Tcl/Tk (2023). Recuperado de Tkinter.

7. HandWiki. MOESI Protocol. (2021). Recuperado de MOESI.

8. Bukunmi, O. Comparing the top Python GUI frameworks. (2021). Recuperado de Top Python GUI.