Área Académica de Ingeniería en Computadores

Curso

Arquitectura de Computadores 2

Proyecto 1 Propuesta 2

Profesor: Ronald García Fernández

Elaborado por:

Leonardo Guillen Fernandez

Josue Cubero Montero

Juan Peña Rostrán

Brandon Gomez Gomez

II Semestre 2023, Cartago, Costa Rica

Plan de pruebas y cobertura de funcionalidades

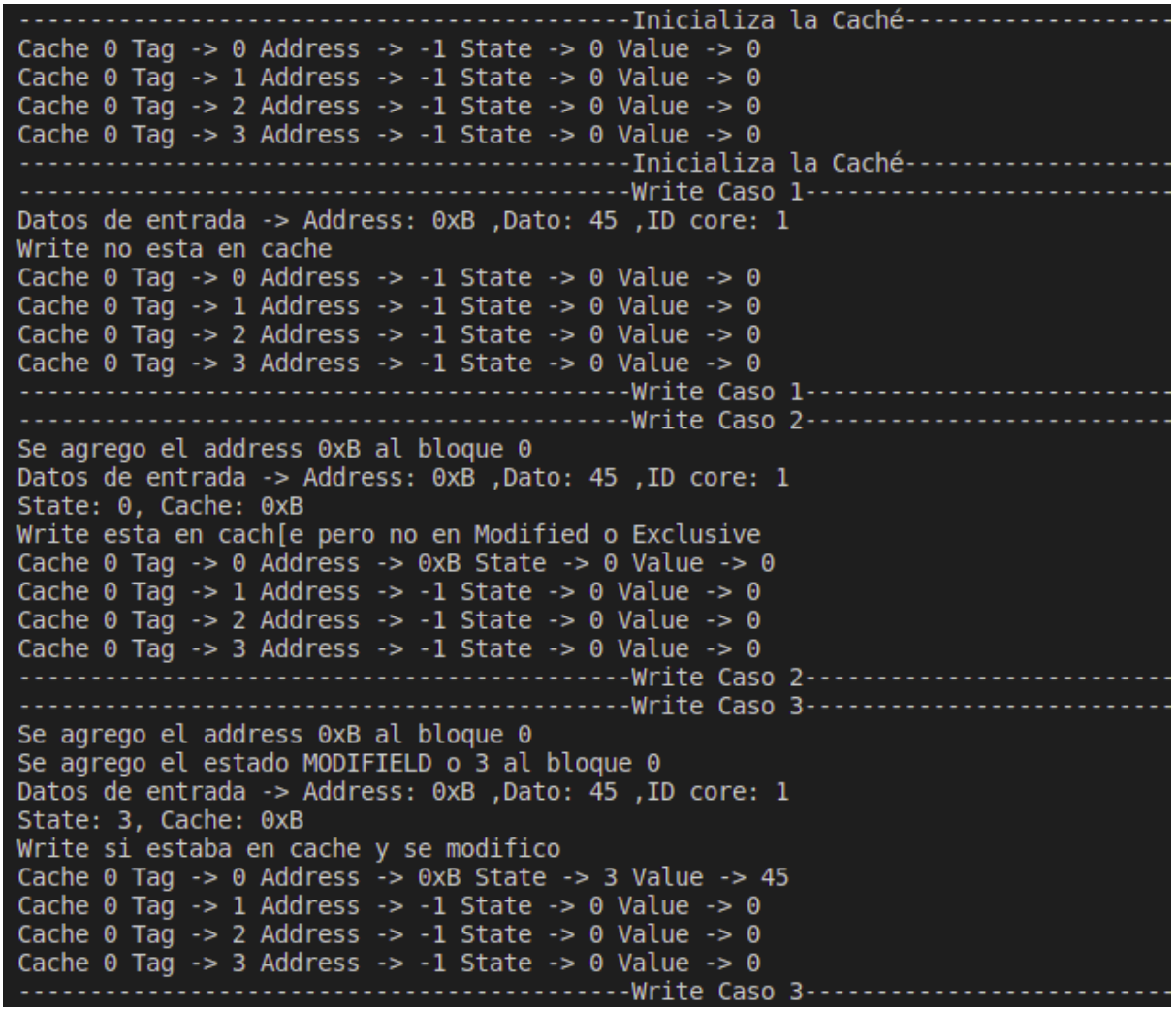
Objetivo General:

Verificar que el sistema multiprocesador funcione correctamente, mediante los protocolos MOESI y MESI en coordinación de los CPUs con las caches privadas y el bus de datos relacionado con la memoria principal.

Pruebas de Unidad (Unit Testing):

Pruebas de Caché:

Prueba de Write:



En el inicio del código se muestra la inicialización de la cache, con tag relacionado con la línea de cache, el address inicializado con -1 , los estados en 0 significativo al estado INVALID.
En el caso 1, se ingresan los datos mostrados en la imagen y como la caché no tiene esa dirección en sus bloques el retorna un mensaje de que esa dirección no está en cache, en este punto el debe enviar un mensaje al bus write miss con un tag para colocar el dato, este dato por la función getBlockIdWithWriteBackPolicy(), probada en esta sección.

En el caso 2, se ingresan los datos mostrados en la imagen, pero esta vez se agrega en el tag 0 de la caché el address “0xB”, entonces el sistema debe reconocer que esa dirección ingresada en los datos ,si está en caché , sin embargo está en estado INVALID o 0, por lo que procede a enviar un mensaje al bus haciendo un writemiss con el tag en el que se encuentra el address en caché.

En el caso 3, se ingresan los datos mostrados en la imagen, por lo que ahora deberá reconocer que si esta en memoria y al estar en estado MODIFIELD o EXCLUSIVE, lo modifica directamente en caché, tal y como se muestra en el tag 0 , agrega el address, modifica su estado y agrega el valor de la instrucción ingresada.

Prueba del Read:

```
-----------------------------------------Inicializa la Caché-------------------
Cache 0 Tag -> 0 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
-----------------------------------------Inicializa la Caché-------------------
-----------------------------------------Read Caso 1---------------------------
Datos de entrada -> Address: 0xB ,Dato: 45 ,ID core: 1
Read no esta en cache write back
Cache 0 Tag -> 0 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
-----------------------------------------Read Caso 1---------------------------
-----------------------------------------Read Caso 2---------------------------
Se agrego el address 0xB al bloque 0, con dato = 9
Datos de entrada -> Address: 0xB ,Dato: 45 ,ID core: 1
Read esta en cache pero en Invalid
Cache 0 Tag -> 0 Address -> 0xB State -> 0 Value -> 9
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
-----------------------------------------Read Caso 2---------------------------
-----------------------------------------Read Caso 3---------------------------
Se agrego el address 0xB al bloque 0, con dato = 9
Se agrego el estado MODIFIELD o 3 al bloque 0
Datos de entrada -> Address: 0xB ,Dato: 45 ,ID core: 1
Read esta en cache
Cache 0 Tag -> 0 Address -> 0xB State -> 3 Value -> 9
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
-----------------------------------------Read Caso 3---------------------------
```

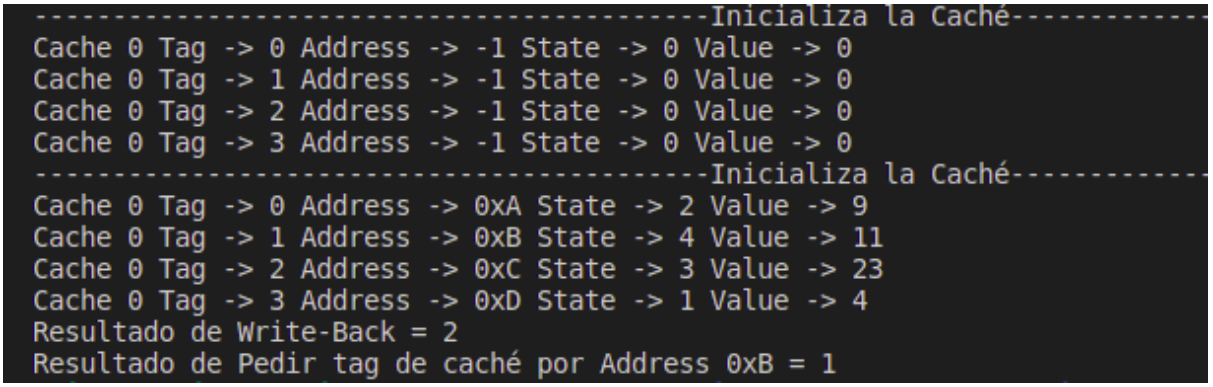
En este bloque de igual manera se inicializa la memoria caché con los mismos valores anteriores.

En el caso 1: se realiza un read con los datos ingresado como se visualiza en la imagen y como no encuentra el address en la caché, envía un mensaje al bus con la política de write back que indica en donde debe escribir el dato en caché que se desea leer de memoria principal.

En el caso 2: se realiza un read con los datos ingresado como se visualiza en la imagen, pero en este caso se agrega a la caché los datos mencionados con la misma address de la instrucción, por lo que ahora se encuentra en caché sin embargo está en estado INVALID por lo que debe enviar un mensaje al bus, solicitando el dato y guardarlo en la tag donde estaba la address consultada.

En el caso 3: se tiene el mismo ejemplo anterior pero el estado del bloque 0 es MODIFIED o 3 por lo que simplemente se procede a leer el bloque desde la caché.

Demás funciones:

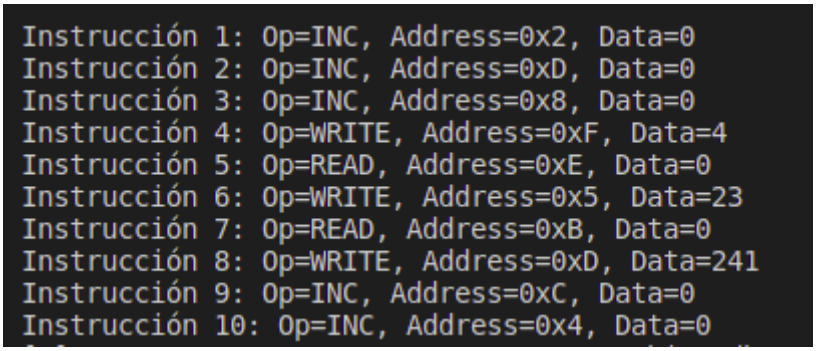


Se inicializa la memoria caché, y luego se le dan valores aleatorios para probar las siguientes funciones:

getBlockIdWithWriteBackPolicy :Esta función aplica la política de write-back, lo que indica que busca una posición disponible para poder guardar un dato en caché, y funciona primero buscando INVALID y como no hay busca MODIFIED o 3 en este caso y como si lo encuentra devuelve el tag encontrado para poder identificarlo y así guardar el dato en esa posición.

getCacheBlock_by_address: Esta función busca un bloque de cache por medio de un address tal y como se muestra, se busca “0xB” y se encuentra en el tag 1.

Pruebas de Instrucciones:



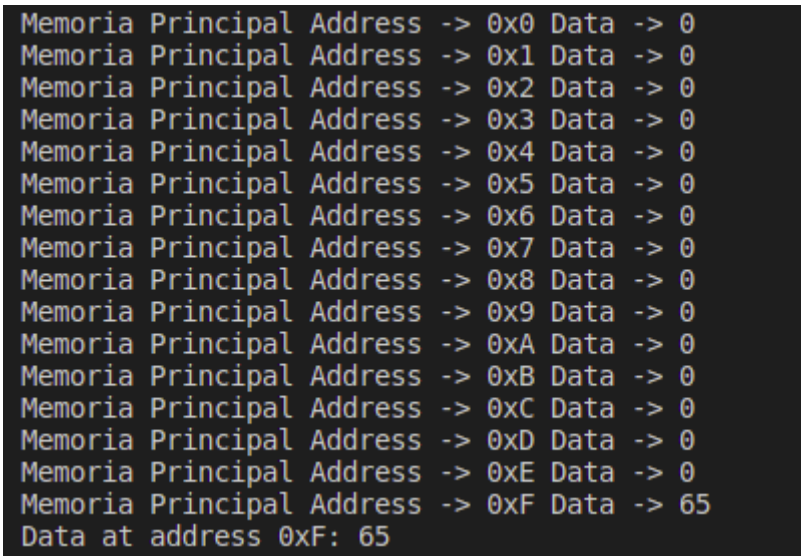
Para esta prueba, se ejecutará el generador de funciones aleatorias, la cual nos da las 3 operaciones esenciales como lo son Read, Write e Incr, de manera aleatoria con valores de address aleatorios en base a las 16 direcciones de memoria principal disponibles, en el caso de Write tiene valores diferentes de 0 porque se debe escribir algún dato, pero en el Read e Incr solo es necesaria la address y el valor se pone en 0 pues no será necesario.

Pruebas de CPU:

```
Instruccion generada Op= READ , Address: 0xC ,Valor= 0
CPU 1 - Ejecutando instrucción:
  Operación: READ
  Dirección: 0xC
  Datos: 0
Instruccion Recibida = Read
--------------------------------------------------------------------------
Instruccion generada Op= WRITE , Address: 0xD ,Valor= 251
CPU 1 - Ejecutando instrucción:
  Operación: WRITE
  Dirección: 0xD
  Datos: 251
Instruccion Recibida = Write
--------------------------------------------------------------------------
Instruccion generada Op= READ , Address: 0x1 ,Valor= 0
CPU 1 - Ejecutando instrucción:
  Operación: READ
  Dirección: 0x1
  Datos: 0
Instruccion Recibida = Read
--------------------------------------------------------------------------
Instruccion generada Op= INCR , Address: 0x9 ,Valor= 0
CPU 1 - Ejecutando instrucción:
  Operación: INCR
  Dirección: 0x9
  Datos: 0
Instruccion Recibida = Incremento
--------------------------------------------------------------------------
Instruccion generada Op= WRITE , Address: 0x3 ,Valor= 204
CPU 1 - Ejecutando instrucción:
  Operación: WRITE
  Dirección: 0x3
  Datos: 204
Instruccion Recibida = Write
```

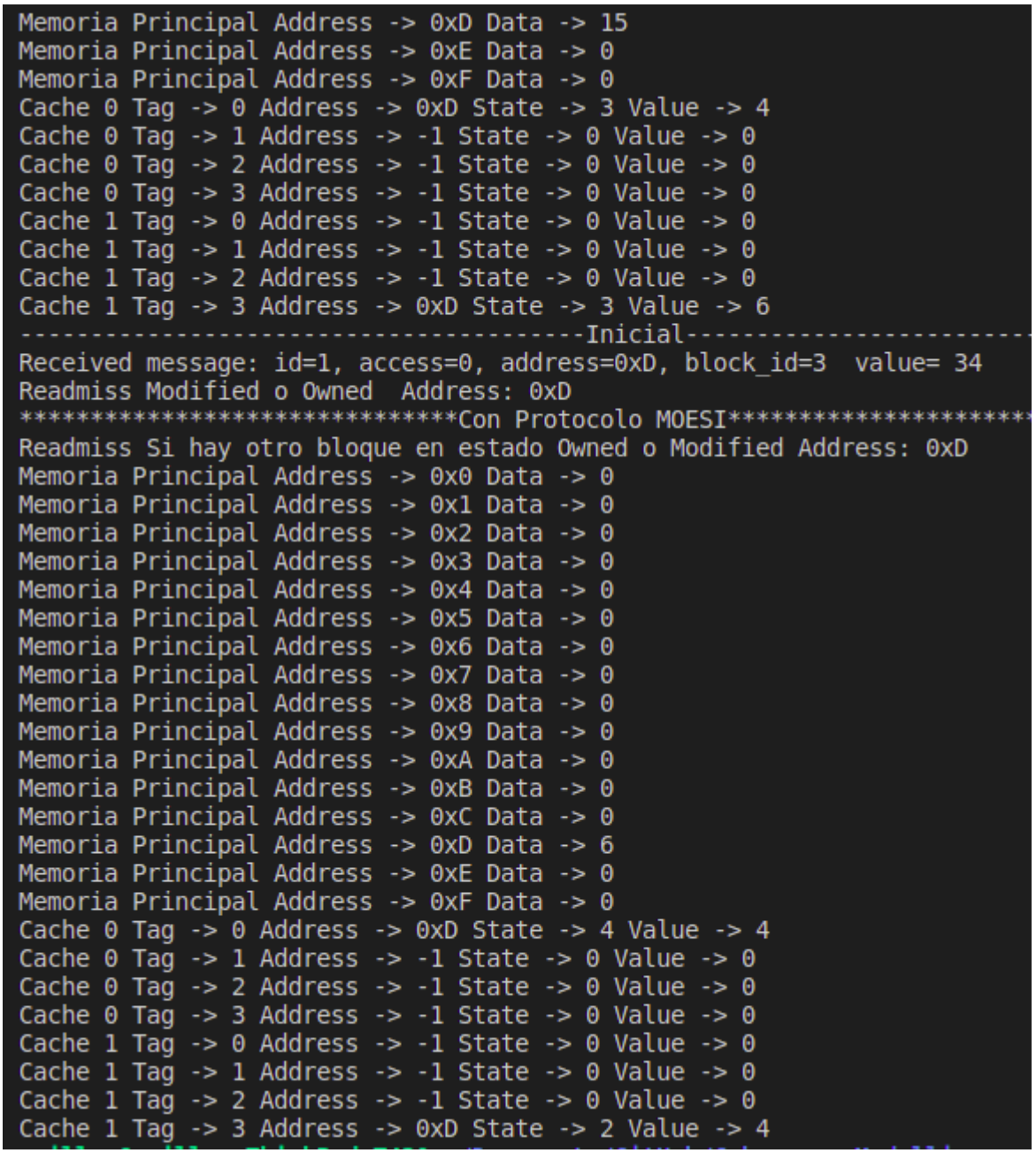
En esa prueba podemos evidenciar que el CPU recibe una instrucción y este tiene la capacidad de recibirla y poder mandar a ejecutar dicha instrucción, con los datos correspondientes.

Pruebas de Bus:



En esta imagen se prueba la función llamada get_data_from_memory() que se encarga de traer el dato de una dirección de memoria que se le ingrese, en este caso se ingresa la dirección 0xF y se aprecia en los prints anteriores todo el contenido de la memoria principal, y vemos que el resultado de la función testeada, calza con el dato guardado en dicha dirección de memoria.

Caso 1 de Readmiss con MOESI:

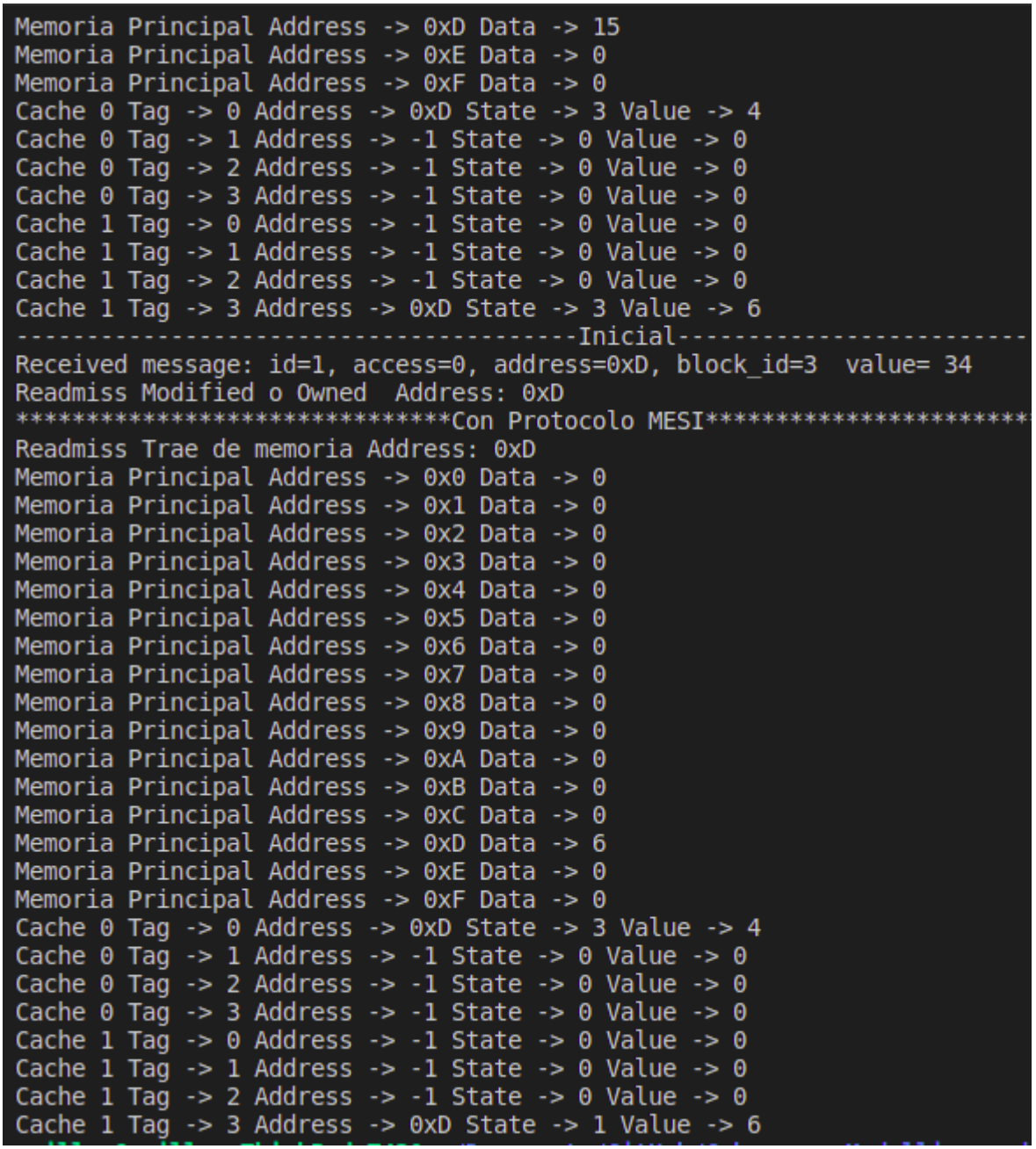


La primera parte de la imagen, es una impresión de la memoria principal y en este caso de prueba, con 2 CPU por lo tanto serían las 2 caches con ciertos datos necesarios para este caso.

El bus es el medio receptor de mensajes o peticiones realizadas por los CPUs , entonces en print de Received message, indica todos los parámetros que un CPU envío, en este caso, lo envió el CPU con id=0 , con acceso de 0 o Readmiss , una address 0xD, en la línea de cache 3 y el valor 34, sin embargo para el read no se necesita un valor.

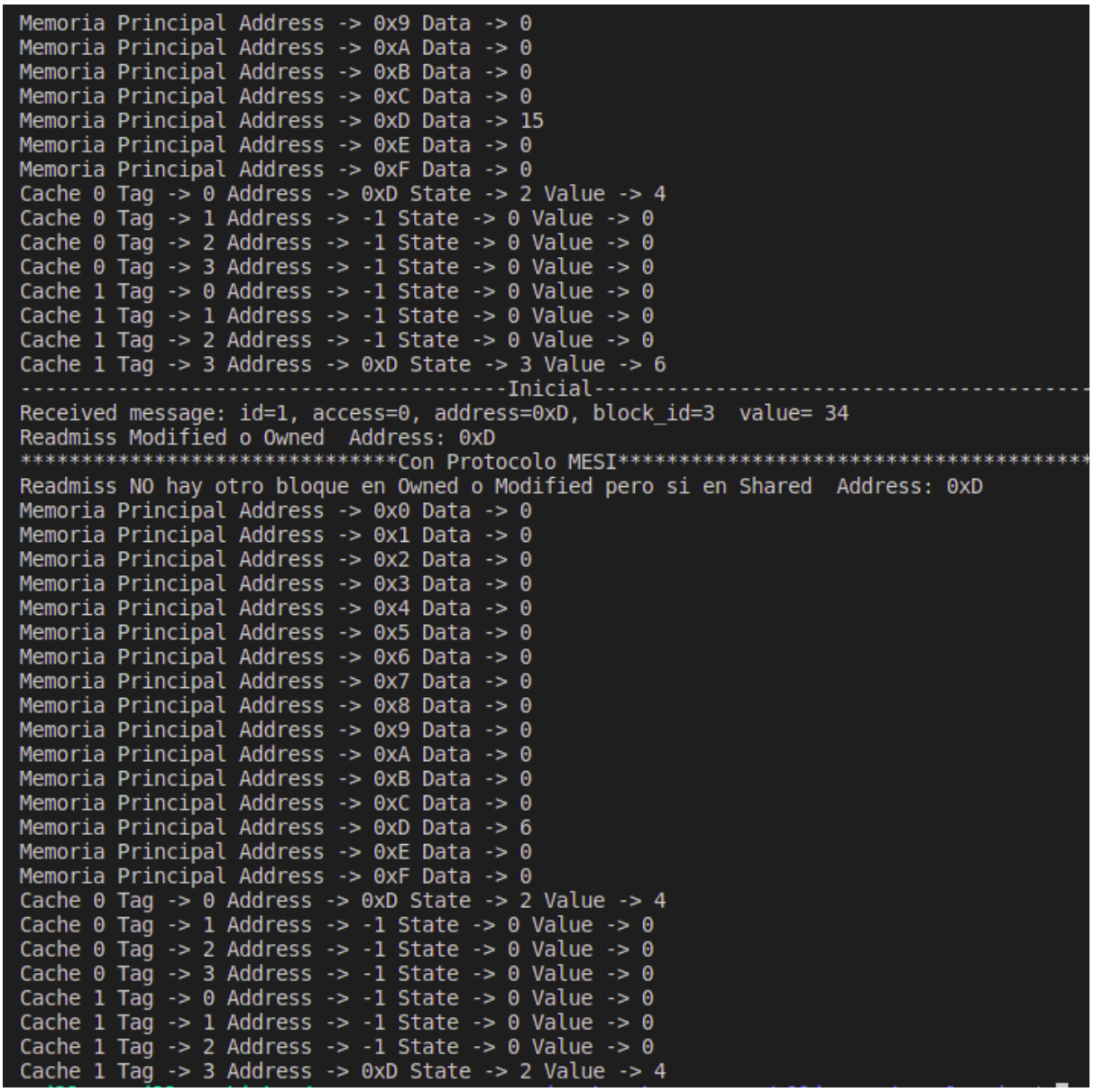
Luego menciona que se está ejecutando con el Protocolo MOESI, también identifica que el estado de la línea de caché actual está en MODIFIED o OWNED y se aprecia que las dos caches tienen la misma address en el mismo estado MODIFIED o 3, por lo que procede a cambiar las otras cachés con la misma address y cambiarles el estado a OWNED , como se refleja en el print de la memoria principal y las caches.

Caso 1 Readmiss con MESI:



Observamos el mismo caso anterior pero ahora con el protocolo MESI, hace exactamente lo mismo, pero comuna leve diferencia, no tiene el estado OWNED por lo que al proceder al mismo caso de que los dos estados estaban en MODIFIED o 3, procede a buscar en las otras cachés pero que se encuentren en estados SHARED 2 o EXCLUSIVE 1, y no cumple esta condición entonces va a traer el dato a memoria como se muestra en la imagen, trajo el valor 6 en la cache 1 en su linea 3.

Caso 2 Readmiss:

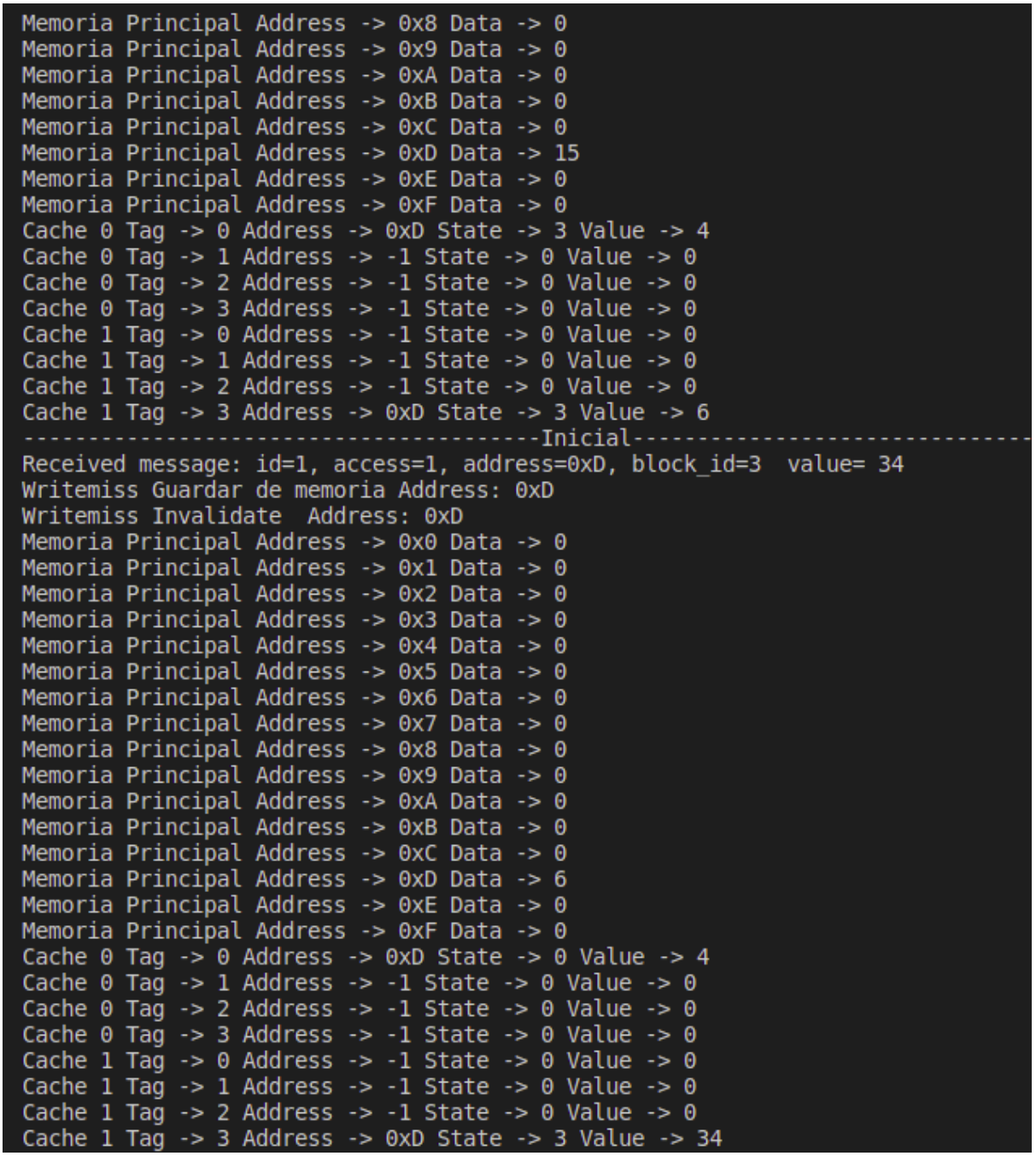


F12

En este caso al igual que los casos anteriores se ingresan datos en memoria principal y en las cos caches de prueba, y recibe un mensaje con un readmis del CPU 1 en la address 0xD, con el Protocolo MESI y MOESI, tendra el mismo funcionamiento puesto que la variante entre los protocolos es un estado y las transiciones a ese estado, por lo cual solo afecta en el caso anterior en los demás no.

Las caches tienen valores, en la cache 0 en la línea 0 tiene el estado de SHARED y en la cache 1 en la línea 3 tiene el estado MODIFIED, y como la petición la realiza el CPU 1 cambia los estados de las líneas de caches que contengan la misma address a SHARED, tal y como se evidencia en los print de la imagen.

Caso 3 Writemiss:



En este caso el mensaje o petición que recibe el bus es un writemiss del CPU 1 en el address 0xD del bloque 3 con el valor 34, entonces en primera instancia el revisa si hay el bloque que se va a escribir tiene estado MODIFIED o OWNED, si si, manda a guardar ese dato a memoria y luego en la caché guarda el nuevo valor de la solicitud, revisando las demás caches, viendo si alguna línea tiene la address del mensaje y si si hay cambia el estado a INVALID.

Pruebas Integración (Integration Testing):

Pruebas de integración de CPU y caché:

```
CPU 1 - Ejecutando instrucción:
  Operación: WRITE
  Dirección: 0x1
  Datos: 0
State: 3, Cache: 0x1
Write si estaba en cache y se modifico
Instruccion Recibida = Write
```

Como se ha visto en las pruebas anteriores, vemos los dos resultados de CPU y de la caché, realizando un write en una dirección con estado modificado por lo que directamente se llegó a cambiar el valor a la caché, dando una integración correcta entre el módulo de CPU y Caché privada.

```
CPU 1 - Ejecutando instrucción:
  Operación: READ
  Dirección: 0x1
  Datos: 0
Read esta en cache
Instruccion Recibida = Read
```

En este caso, da lo mismo, con las mismas condiciones pero con la instrucción READ, dónde valida que el bloque necesario si está en cache con la address de la instrucción procediendo a tomar el valor desde la caché.

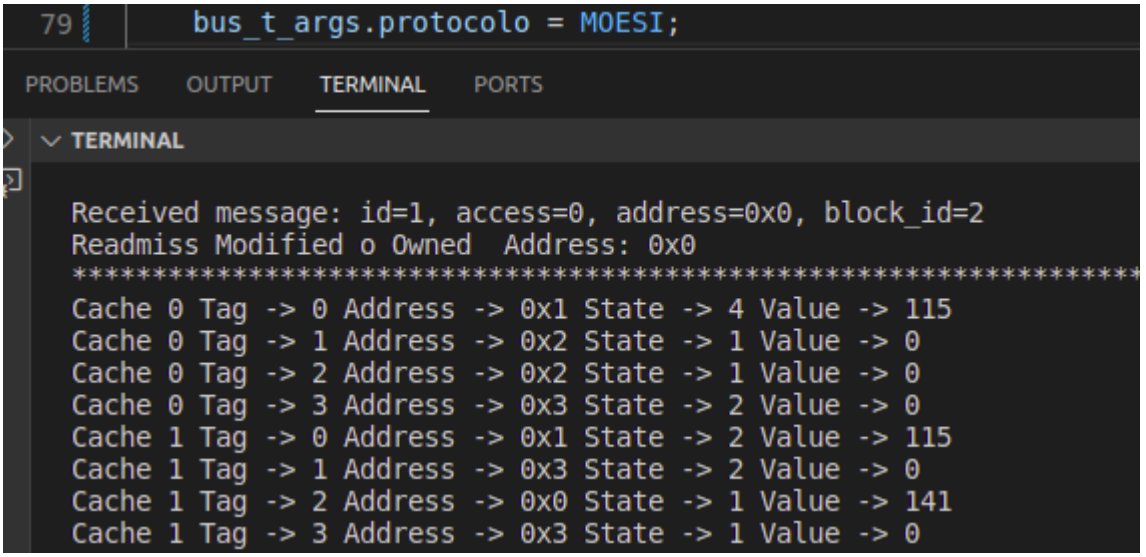
Pruebas de integración del bus con los CPUs:

```
Executing instruction WRITE from core 0
CPU 0 - Ejecutando instrucción:
  Operación: WRITE
  Dirección: 0x9
  Datos: 115
Write no esta en cache
Executing instruction READ from core 1
CPU 1 - Ejecutando instrucción:
  Operación: READ
  Dirección: 0xC
  Datos: 0
77777777---Cache x Tag -> 0 Address -> -1 State -> 0 Value -> 0
77777777---Cache x Tag -> 1 Address -> -1 State -> 0 Value -> 0
77777777---Cache x Tag -> 2 Address -> -1 State -> 0 Value -> 0
77777777---Cache x Tag -> 3 Address -> -1 State -> 0 Value -> 0
Received message: id=0, access=1, address=0x9, block_id=0
Writemiss Invalidate  Address: 0x9
Cache 0 Tag -> 0 Address -> 0x9 State -> 3 Value -> 115
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 0 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Read no esta en cache
Received message: id=1, access=0, address=0xC, block_id=0
Readmiss Trae de memoria Address: 0xC
Cache 0 Tag -> 0 Address -> 0x9 State -> 3 Value -> 115
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 0 Address -> 0xC State -> 1 Value -> -1
Cache 1 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 3 Address -> -1 State -> 0 Value -> 0
```

En esta etapa se busca la integración entre el bus que es el receptor de mensajes y los CPUs que son los emisores de los mensajes, primero de ejecutan las instrucciones una por cada CPU , en la impresión con las etiquetas de 7 son la inicialización de las caches, y luego se ven dos secciones de impresiones de los contenidos de las 2 caches, donde a partir del mensaje recibido se ejecuta un Writemiss para el procesador 0 y un Readmiss para el procesador 1 cambiando respectivamente sus caches, dependiendo de las instrucciones ejecutadas.

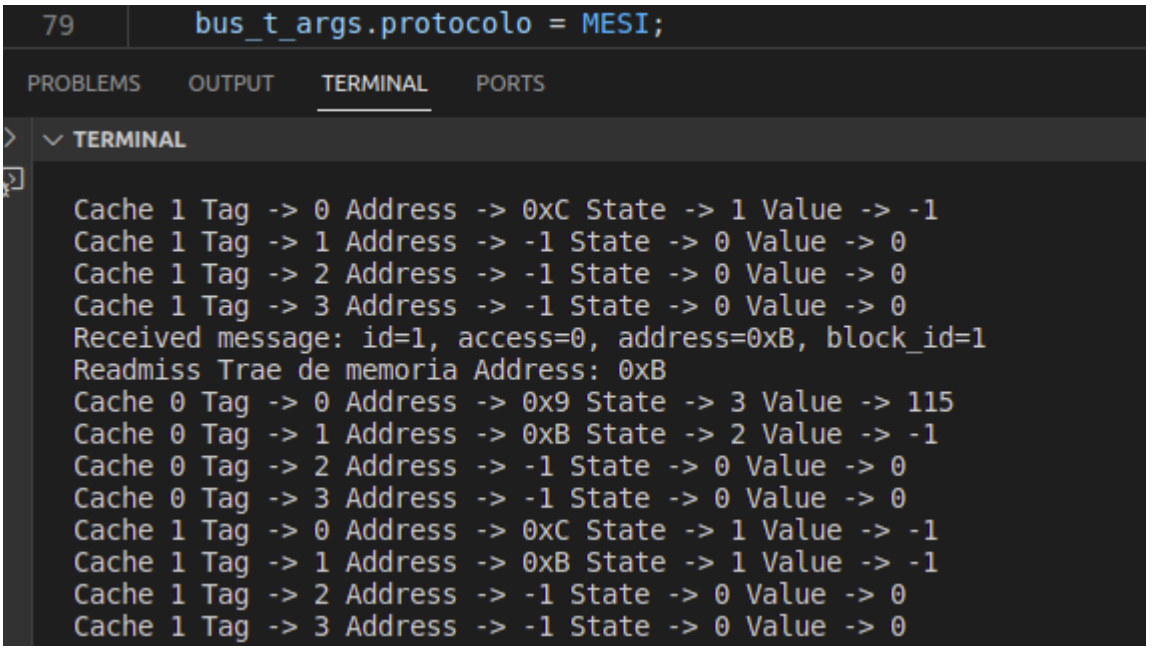
Cobertura de Funcionalidades Específicas:

Cobertura de estados MOESI:



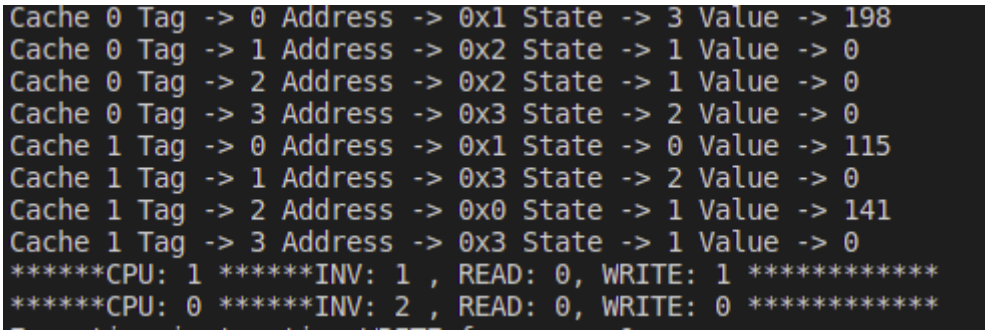
En la imagen anterior se puede observar las dos caches utilizando el protocolo MOESI, puesto que contiene sus estados contando con el estado OWNED o 4.

Cobertura de estados MESI:



A comparación de lo anterior el protocolo MESI no tiene el estado 4, por lo que después de una serie de ejecuciones, este estado nunca llega a ser ingresado , cuando se corra el simulador con el protocolo MESI.

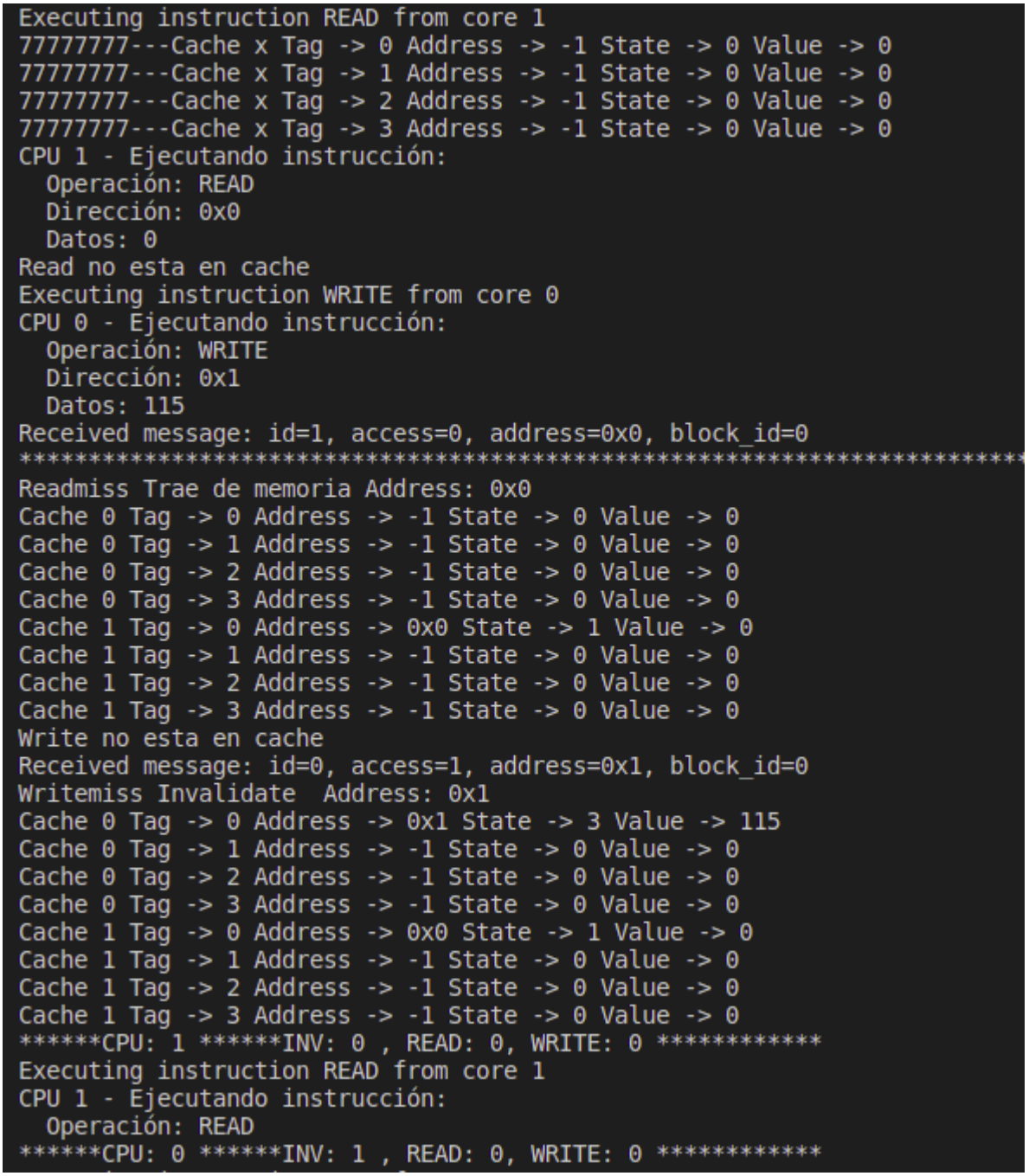
Cobertura de transiciones:



A nivel de transiciones, se colocaron contadores en los puntos específicos que eran considerados condiciones para poder aumentar esta estadística, y como se ve en la imagen, cada CPU tiene sus propias estadísticas.

Pruebas de Sistema (System Testing):

Pruebas de sistema sin Interfaz:



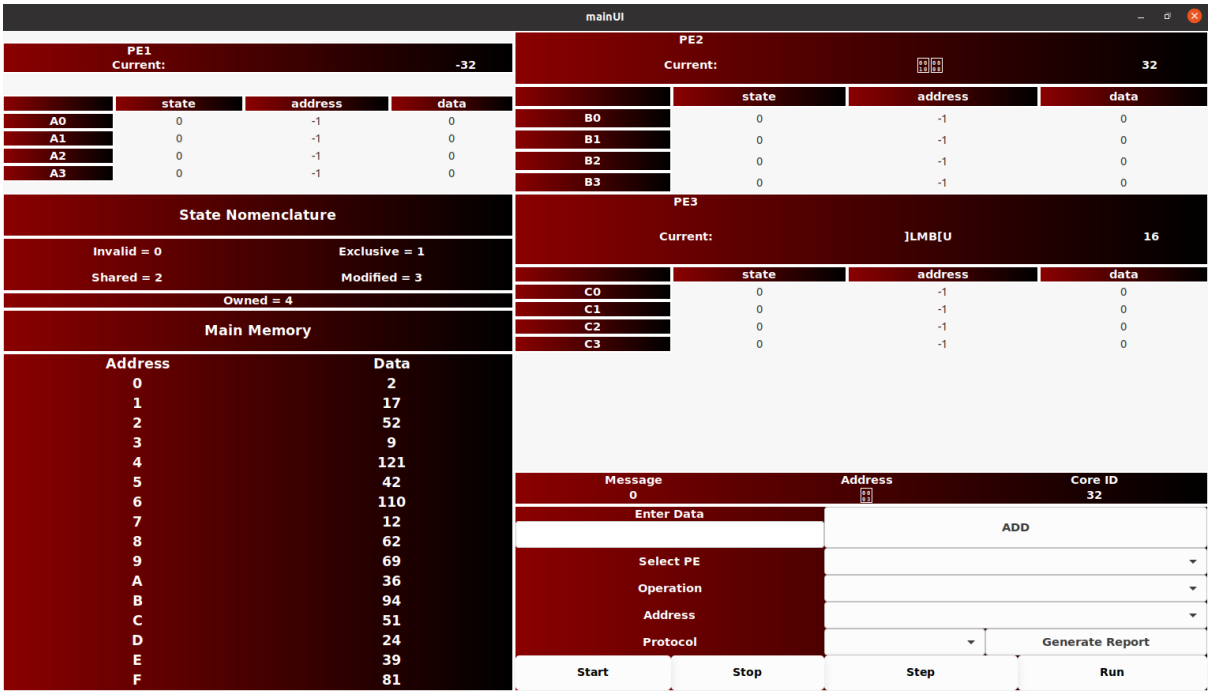
```
Executing instruction READ from core 1
77777777---Cache x Tag -> 0 Address -> -1 State -> 0 Value -> 0
77777777---Cache x Tag -> 1 Address -> -1 State -> 0 Value -> 0
77777777---Cache x Tag -> 2 Address -> -1 State -> 0 Value -> 0
77777777---Cache x Tag -> 3 Address -> -1 State -> 0 Value -> 0
CPU 1 - Ejecutando instrucción:
  Operación: READ
  Dirección: 0x0
  Datos: 0
Read no esta en cache
Executing instruction WRITE from core 0
CPU 0 - Ejecutando instrucción:
  Operación: WRITE
  Dirección: 0x1
  Datos: 115
Received message: id=1, access=0, address=0x0, block_id=0
**********
Readmiss Trae de memoria Address: 0x0
Cache 0 Tag -> 0 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 0 Address -> 0x0 State -> 1 Value -> 0
Cache 1 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Write no esta en cache
Received message: id=0, access=1, address=0x1, block_id=0
Writemiss Invalidate  Address: 0x1
Cache 0 Tag -> 0 Address -> 0x1 State -> 3 Value -> 115
Cache 0 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 0 Address -> 0x0 State -> 1 Value -> 0
Cache 1 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 3 Address -> -1 State -> 0 Value -> 0
******CPU: 1 ******INV: 0 , READ: 0, WRITE: 0 ************
Executing instruction READ from core 1
CPU 1 - Ejecutando instrucción:
  Operación: READ
******CPU: 0 ******INV: 1 , READ: 0, WRITE: 0 ************
```

En esta primera etapa se tiene la inicialización de las memorias caches, luego inicia ejecutando 2 CPU simultáneos con sus instrucciones específicas, haciendo un print de las caches en su estado actual, donde se puede ver el movimiento de sus address, además después de que el CPU ejecuta , el bus recibe el mensaje y en ciertas ocasiones cambia valores de las otras cachés, también mostrando los valores de las estadísticas por cada CPU.

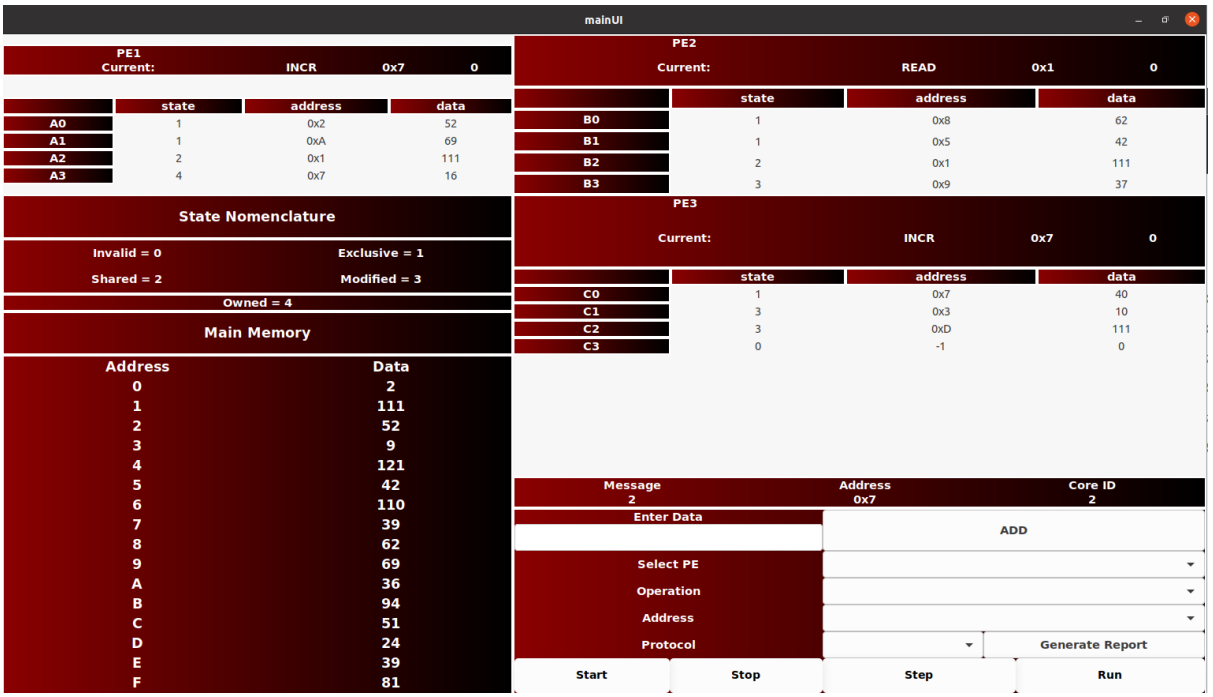
```
Executing instruction READ from core 1
CPU 1 - Ejecutando instrucción:
  Operación: READ
******CPU: 0 ******INV: 1 , READ: 0, WRITE: 0 ************
Executing instruction READ from core 0
CPU 0 - Ejecutando instrucción:
  Operación: READ
  Dirección: 0x3
  Datos: 0
Read no esta en cache
  Dirección: 0x3
Received message: id=0, access=0, address=0x3, block_id=1
**************************************************************************
Readmiss Trae de memoria Address: 0x3
Cache 0 Tag -> 0 Address -> 0x1 State -> 3 Value -> 115
Cache 0 Tag -> 1 Address -> 0x3 State -> 1 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 0 Address -> 0x0 State -> 1 Value -> 0
Cache 1 Tag -> 1 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 2 Address -> -1 State -> 0 Value -> 0
  Datos: 0
Read no esta en cache
Cache 1 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Received message: id=1, access=0, address=0x3, block_id=1
**************************************************************************
Readmiss NO hay otro bloque en Owned pero si en Shared  Address: 0x3
Cache 0 Tag -> 0 Address -> 0x1 State -> 3 Value -> 115
Cache 0 Tag -> 1 Address -> 0x3 State -> 2 Value -> 0
Cache 0 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 0 Tag -> 3 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 0 Address -> 0x0 State -> 1 Value -> 0
Cache 1 Tag -> 1 Address -> 0x3 State -> 2 Value -> 0
Cache 1 Tag -> 2 Address -> -1 State -> 0 Value -> 0
Cache 1 Tag -> 3 Address -> -1 State -> 0 Value -> 0
******CPU: 0 ******INV: 1 , READ: 0, WRITE: 0 ************
Executing instruction WRITE from core 0
CPU 0 - Ejecutando instrucción:
  Operación: WRITE
  Dirección: 0x0
  Datos: 194
Write no esta en cache
Received message: id=0, access=1, address=0x0, block_id=2
Writemiss Invalidate  Address: 0x0
```

El código continúa haciendo más ejecuciones en los CPUs , aumentando las estadísticas y alterando los valores en las caches, cumpliendo con los estados y transiciones tal y como se han evidenciado en los test de cada componente.

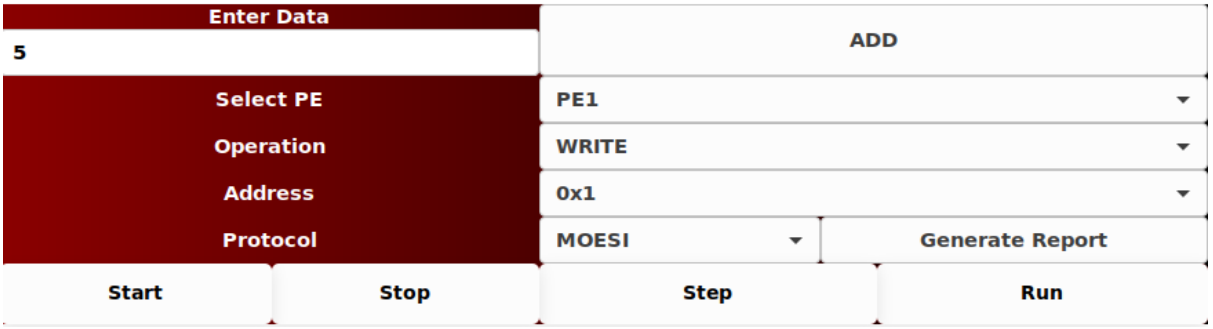
Pruebas del Sistema con Interfaz:



En la imagen anterior se muestra la versión finalizada de la interfaz gráfica, donde al iniciar, inicializa la memoria principal con la diferencia que ahora generamos números aleatorios y en las caches se ven en su estado inicial con direcciones inválidas.



Basados en los botones, el primero , llamado Start, inicia el proceso de ejecución cíclica y se detiene hasta que se presione el botón Stop.
Como se aprecia en la imagen ya se corrieron varias ejecuciones y se ven los estados de las líneas de caches, sus direcciones actuales y el valor que contienen.y en la imagen se aprecian los mensajes recibidos por el bus, mostrando el mensaje 1 significa Write Miss y el 0 significa Readmiss, conteniendo también el core id del PE que lo envió.



En esta sección se ingresa una instrucción a un respectivo PE y se agrega con el botón add, agrega la instrucción en una lista para poder ejecutarlas al momento de darle click al botón run, demostrando en los primeros pasos que tiene el mismo funcionamiento de la simulación en consola, sin embargo ahora es más fácil de identificar el flujo de datos y las acciones producida por cada ejecución de cada CPU.