**THE UNIVERSITY OF DANANG**

**UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**Faculty of Advanced Science and Technology**

**FINAL PROJECT**

**REPORT**

**Cache Simulation**

**Instructor: Bui Thi Thanh Thanh**

**Class: 21ES**

**Student: Le Dinh Bao Tin**

**Tran Chi Thanh**

**Nguyen Duc Hung**

1. **Cache Operations and Pseudo code**:
2. **Read from L1 Data Cache**:
3. **Read cases**:

* Hit case:
* Found a matched **tag** in the set and **Valid=1**.
* Miss cases:
* Found a matched **tag** in the set but **Valid=0**: L1 Data Cache evict the invalid line and **read** the block from L2 again.
* No **tag** matches found in the set; But the set have **empty space**: **Read** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; there is line with **Valid=0**: L1 Data Cache evict the invalid line and **read** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; There is **NO invalid line**; LRU line have **dirty=0**: L1 Data Cache evict LRU line and **read** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; There is **NO invalid line**; LRU line have **dirty=1**: L1 Data Cache writeback to L2 LRU line and **read** the corresponding block from L2.

1. **Pseudo code**:

if (Address\_Tag == A\_Line\_in\_Set->Tag)

{

if (A\_Line\_in\_Set->Valid == true)

{

READ HIT

}

Else

{

READ MISS

L1 Evict the INVALID line

Read from L2 block contain request address

}

}

else

{

if (Set\_have\_empty\_line = true)

{

READ MISS

Read from L2 block contain request address

}

else

{

READ MISS

if (There is INVALID line in set)

{

L1 Evict the INVALID line

Read from L2 block contain request address

}

else

{

Find LRU line

if (LRU\_line\_in\_Set->Dirty == false)

{

L1 Evict the LRU = 0 line

Read from L2 block contain request address

}

else

{

L1 Write back the LRU = 0 + dirty line to L2

Read from L2 block contain request address

}

}

}

}

1. **Write to L1 Data Cache**:
2. **Write cases**:

* Hit case:
* Found a matched **tag** in the set and **Valid=1**.
* Miss cases:
* Found a matched **tag** in the set but **Valid=0**: L1 Data Cache evict the invalid line and **read for ownership** the block from L2 again.
* No **tag** matches found in the set; But the set have **empty space**: **Read for ownership** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; there is line with **Valid=0**: L1 Data Cache evict the invalid line and **read for ownership** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; There is **NO invalid line**; LRU line have **dirty=0**: L1 Data Cache evict LRU line and **read for ownership** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; There is **NO invalid line**; LRU line have **dirty=1**: L1 Data Cache writeback to L2 LRU line and **read for ownership** the corresponding block from L2.

1. **Pseudo code**:

if (Address\_Tag == A\_Line\_in\_Set->Tag)

{

if (A\_Line\_in\_Set->Valid == true)

    {

WRITE HIT

Line -> Dirty = 1

    }

    else

    {

        WRITE MISS

        L1 Evict the INVALID line

        Read for Ownership from L2 block contain request address

Line -> Dirty = 1

    }

}

else

{

    if (Set\_have\_empty\_line = true)

    {

        WRITE MISS

        Read for Ownership from L2 block contain request address

Line -> Dirty = 1

    }

    else

    {

        WRITE MISS

if (There is INVALID line in set)

{

            L1 Evict the INVALID line

            Read for Ownership from L2 block contain request address

Line -> Dirty = 1

}

else

{

      if (LRU\_line\_in\_Set->Dirty == false)

        {

          L1 Evict the LRU = 0 line

            Read for Ownership from L2 block contain request address

Line -> Dirty = 1

        }

        else

        {

            L1 Write back the dirty line to L2

            Read for Ownership from L2 block contain request address

Line -> Dirty = 1

        }

}

    }

}

1. **Read from Instruction Cache**:
2. **Instruction fetch cases**:

* Hit case:
* Found a matched **tag** in the set and **Valid=1.**
* Miss cases:
* Found a matched **tag** in the set but **Valid=0**: L1 Data Cache evict the invalid line and **read** the block from L2 again.
* No **tag** matches found in the set; But the set have **empty space**: **Read** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; there is line with **Valid=0**: L1 Instruction Cache evict the invalid line and **read** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; There is **NO invalid line**; LRU line have **dirty=0**: L1 Instruction Cache evict LRU line and **read** the corresponding block from L2.
* No **tag** matches found in the set; The set have **NO empty space**; There is **NO invalid line**; LRU line have **dirty=1**: ERROR => Instruction cache cannot have dirty line

1. **Pseudo code**:

if (Address\_Tag == A\_Line\_in\_Set->Tag)

{

if (A\_Line\_in\_Set->Valid == true)

    {

    READ HIT

    }

    else

    {

        READ MISS

        L1 instruction Evict the INVALID line

        Read from L2 block contain request address

    }

}

else

{

    if (Set\_have\_empty\_line = true)

    {

        READ MISS

        Read from L2 block contain request address

    }

    else

    {

        READ MISS

if (There is INVALID line in set)

{

            L1 Evict the INVALID line

            Read from L2 block contain request address

}

else

{

      if (LRU\_line\_in\_Set->Dirty == false)

        {

            L1 instruction Evict the LRU = 0 line

            Read from L2 block contain request address

        }

        else

        {

            ERROR: Dirty bits in INSTRUCTION Cache are always false (0)

        }

}

    }

}

1. **Eviction Command from L2**:
2. **Eviction Cases**:

* Found a **matched Tag** in L1; The line has **Valid=0**: L1 can just ignore the line
* Found a **matched Tag** in L1; The line has **Valid=1**; **Dirty=0**: L1 set **Valid=0**
* Found a **matched Tag** in L1; The line has **Valid=1**; **Dirty=1**: L1 **write back** the line to L2 and set **Valid=0**
* **NO Tag matched found** in L1: ERROR: The evicted line in L2 not found in L1

1. **Pseudo code**:

if (Evicted\_Tag == Tag\_in\_Set)

{

if (Line with match Tag -> Valid == 1)

    {

        if (Line with match Tag -> Dirty == 0)

        {

            L1 set the line Valid=0;

        }

        else

        {

            L1 Write back the line to L2;

            L1 set the line Valid=0;

        }

    }

    else

    {

        L1 ignore the line; //As it's already INVALID!

    }

}

else

{

    ERROR: The evicted line in L2 not found in L1;

}

1. **LRU implementation**:
2. **LRU cases consideration**:

* In this project, we ignore the line validity as it is already considered in a read/write accesses. Valid and Invalid lines are treated equally when calculate the LRU index.
* Empty lines are always the latter lines in the set. For example, Set 0 initialize empty; The first request will fill line 1; The second request fill line 2; The thirst request fill line 3; Therefore, the empty line are always at the higher line index; If for example, line 2 is invalidate due to eviction command from L2, because we treat all line equally => the empty space is still line 4 of the set; When a miss reference, the invalid line is take into account first => invalid will be removed first.
* Example:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Set x | Initialize | Miss reference | Miss reference | Miss reference | Invalidate Line 2 | Miss reference |
| Line 0 | 0 | 3 | 2 | 1 | 1 | 1 |
| Line 1 | 0 | 0 | 3 | 2 | 2  (invalid) | 3 |
| Line 2 | 0 | 0 | 0 | 3 | 3 | 2 |
| Line 3 | 0 | 0 | 0 | 0 | 0 | 0 |

* Step in calculating the LRU index for each line:
* Get the current reference line LRU state
* Loop from line 0 to the last line
* If the line in each iteration has LRU state > reference line LRU state; minus the LRU state of that line by 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Set x | Initialize | Miss reference | Miss reference | Miss reference | Ref to line 2 |
| Line 0 | 0 | 3 | 2 | 1 | 1 |
| Line 1 | 0 | 0 | 3 | 2 | **3** (largest) |
| Line 2 | 0 | 0 | 0 | 3 | 2  (3-1) |
| Line 3 | 0 | 0 | 0 | 0 | 0 |

1. **Pseudo code**:

* A “Empty\_Flag” is used. This flag is set only when a miss occurs and there is empty line in cache => The data will be loaded into that empty line. This is to avoid empty line with LRU\_State=0 get minus 1 and become negative.

if (Miss occur && Empty line)

{

for (int i = 0; i < Empty\_line\_index; i++)

    {

        Line i -> LRU\_State--;

    }

    Set Empty\_line -> LRU\_State = 3;

}

else

{

    Get the reference line CURRENT LRU state;

    for (int i = 0; i < 4; i++)

    {

        if (Line i -> LRU\_State > Current reference line LRU state)

        {

            Line i -> LRU\_State --;

        }

        else

        {

            nop;

        }

    }

    Referenced\_line = 3;

}

1. **Testing and Verification (Results):**

* In evert test:
* We have a sequence table which show the test sequence:

<operation> <address>

* Expected result which include valid, dirty and LRU data:

S0\_L1:V1/D0/0||L2:V1/D0/1||L3:V0/D0/3||L4:V1/D1/2 means:

* S0: Set 0
* Line 1: Valid=1, Dirty=0, LRU index=0
* Line 2: Valid=1, Dirty=0, LRU index=1
* Line 3: Valid=0, Dirty=0, LRU index=3
* Line 4: Valid=1, Dirty=1, LRU index=2
* If nothing is change, the expected result is left intact. For example: A hit to the current MRU (most recently used) in the set.
* The comment section is used to provide further information or the purpose of some test vector. The comment can have notation such as: UM (Unoccupied miss), CM (Conflict miss), Lx\_H (Hit on Line x). When comment miss, we expect L1 ⬄ L2 message if choosing mode 1 in information displaying (describe in section II.6 above)

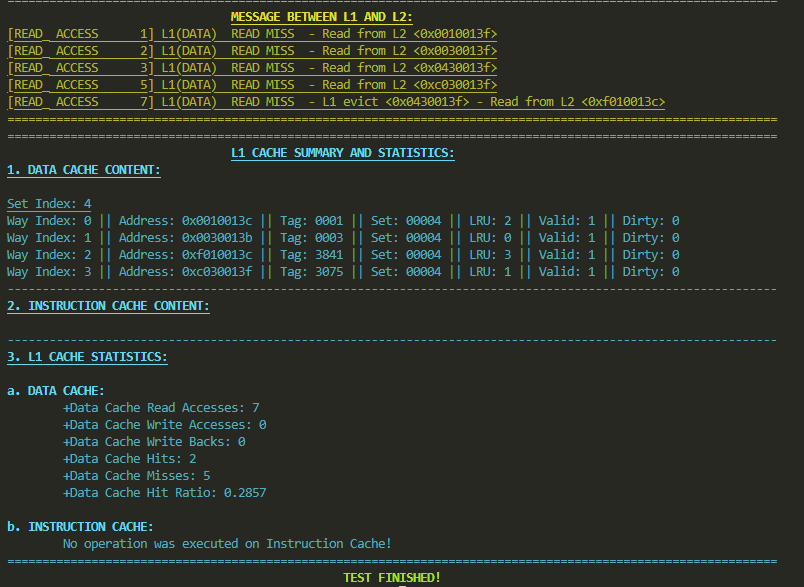
1. **L1 Data Cache – Read Test:**
2. **Test sequences**:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Sequence | Expected Result | Comment |
| 1 | 0 0x0010013F | S4\_L0:V1/D0/3||L1:V0/D0/0||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 2 | 0 0x0030013F | S4\_L0:V1/D0/2||L1:V0/D0/3||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 3 | 0 0x0430013F | S4\_L0:V1/D0/1||L1:V0/D0/2||L2:V0/D0/3||L3:V0/D0/0 | UM |
| 4 | 0 0x0030013B | S4\_L0:V1/D0/1||L1:V0/D0/3||L2:V0/D0/2||L3:V0/D0/0 | L1\_H |
| 5 | 0 0xC030013F | S4\_L0:V1/D0/0||L1:V1/D0/2||L2:V1/D0/1||L3:V1/D0/3 | UM |
| 6 | 0 0x0010013C | S4\_L0:V1/D0/3||L1:V1/D0/1||L2:V1/D0/0||L3:V1/D0/2 | L0\_H |
| 7 | 0 0xF010013C | S4\_L0:V1/D0/2||L1:V1/D0/0||L2:V1/D0/3||L3:V1/D0/1 | CM |

1. **Trace file**:

|  |
| --- |
| 0 0010013F  0 0030013F  0 0430013F  0 0030013B  0 C030013F  0 0010013C  0 F010013C  9 |

1. **Test Result**:



1. **Comment**:

* The simulated result is the same as what we expected!

1. **L1 Data Cache – Write Test:**

* Some notation uses in the Comment field:
* WB\_Lx: Write back line x to L2

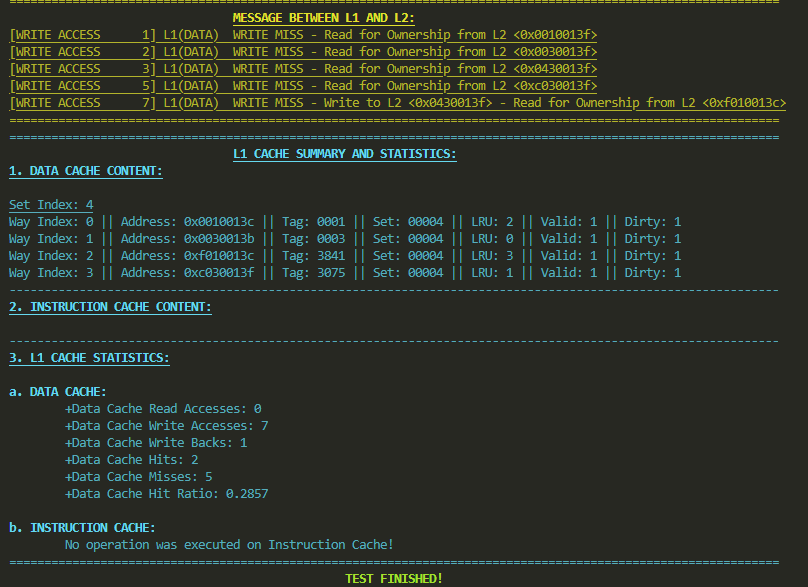
1. **Test Sequence**:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Sequence | Expected Result | Comment |
| 1 | 1 0x0010013F | S4\_L0:V1/D1/3||L1:V0/D0/0||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 2 | 1 0x0030013F | S4\_L0:V1/D1/2||L1:V1/D1/3||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 3 | 1 0x0430013F | S4\_L0:V1/D1/1||L1:V1/D1/2||L2:V1/D1/3||L3:V0/D0/0 | UM |
| 4 | 1 0x0030013B | S4\_L0:V1/D1/1||L1:V1/D1/3||L2:V1/D1/2||L3:V0/D0/0 | L1\_H |
| 5 | 1 0xC030013F | S4\_L0:V1/D1/0||L1:V1/D1/2||L2:V1/D1/1||L3:V1/D1/3 | UM |
| 6 | 1 0x0010013C | S4\_L0:V1/D1/3||L1:V1/D1/1||L2:V1/D1/0||L3:V1/D1/2 | L0\_H |
| 7 | 1 0xF010013C | S4\_L0:V1/D1/2||L1:V1/D1/0||L2:V1/D1/3||L3:V1/D1/1 | CM, WB\_L2 |

1. **Trace file**:

|  |
| --- |
| 1 0010013F  1 0030013F  1 0430013F  1 0030013B  1 C030013F  1 0010013C  1 F010013C  9 |

1. **Test Result**:



1. **Comment**:

* The simulated result is the same as what we expected!

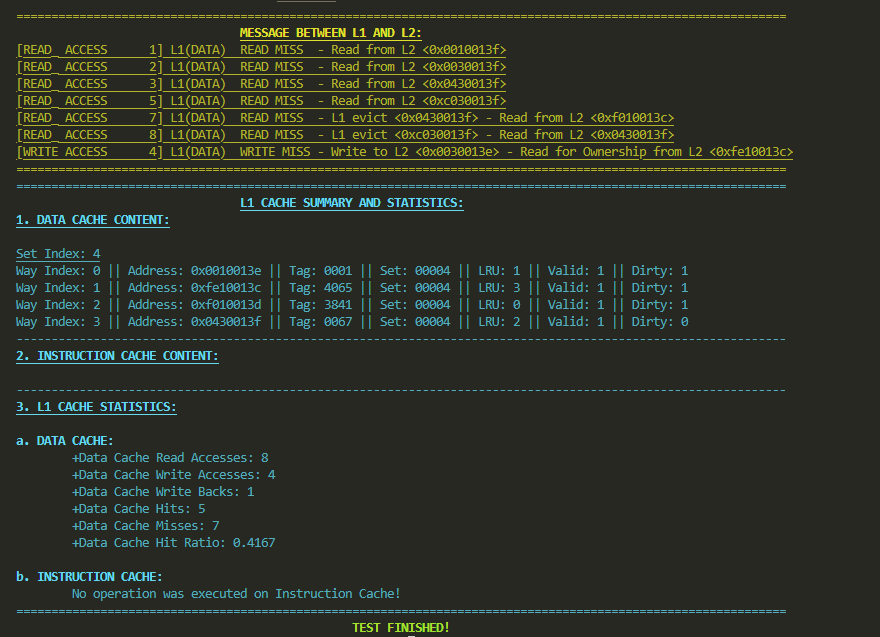
1. **L1 Data Cache – Read/Write Test**:
2. **Test Sequence**:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Sequence | Expected Result | Comment |
| 1 | 0 0x0010013F | S4\_L0:V1/D0/3||L1:V0/D0/0||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 2 | 0 0x0030013F | S4\_L0:V1/D0/2||L1:V1/D0/3||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 3 | 0 0x0430013F | S4\_L0:V1/D0/1||L1:V1/D0/2||L2:V1/D0/3||L3:V0/D0/0 | UM |
| 4 | 0 0x0030013B | S4\_L0:V1/D0/1||L1:V1/D0/3||L2:V1/D0/2||L3:V0/D0/0 | L1\_H |
| 5 | 0 0xC030013F | S4\_L0:V1/D0/0||L1:V1/D0/2||L2:V1/D0/1||L3:V1/D0/3 | UM |
| 6 | 0 0x0010013C | S4\_L0:V1/D0/3||L1:V1/D0/1||L2:V1/D0/0||L3:V1/D0/2 | L0\_H |
| 7 | 0 0xF010013C | S4\_L0:V1/D0/2||L1:V1/D0/0||L2:V1/D0/3||L3:V1/D0/1 | CM |
| 8 | 1 0x0030013E | S4\_L0:V1/D0/1||L1:V1/D1/3||L2:V1/D0/2||L3:V1/D0/0 | L1\_H |
| 9 | 1 0xF010013D | S4\_L0:V1/D0/1||L1:V1/D1/2||L2:V1/D1/3||L3:V1/D1/0 | L2\_H |
| 10 | 1 0x0010013E | S4\_L0:V1/D1/3||L1:V1/D1/1||L2:V1/D0/2||L3:V1/D1/0 | L0\_H |
| 11 | 0 0x0430013F | S4\_L0:V1/D1/2||L1:V1/D1/0||L2:V1/D0/1||L3:V1/D1/3 | CM |
| 12 | 1 0xFE10013C | S4\_L0:V1/D1/1||L1:V1/D1/3||L2:V1/D0/0||L3:V1/D1/2 | CM, WB\_L1 |

1. **Trace file**:

|  |
| --- |
| 0 0010013F  0 0030013F  0 0430013F  0 0030013B  0 C030013F  0 0010013C  0 F010013C  1 0030013E  1 F010013D  1 0010013E  0 0430013F  1 FE10013C  9 |

1. **Test Result**:



1. **Comment**:

* The simulated result is the same as what we expected!

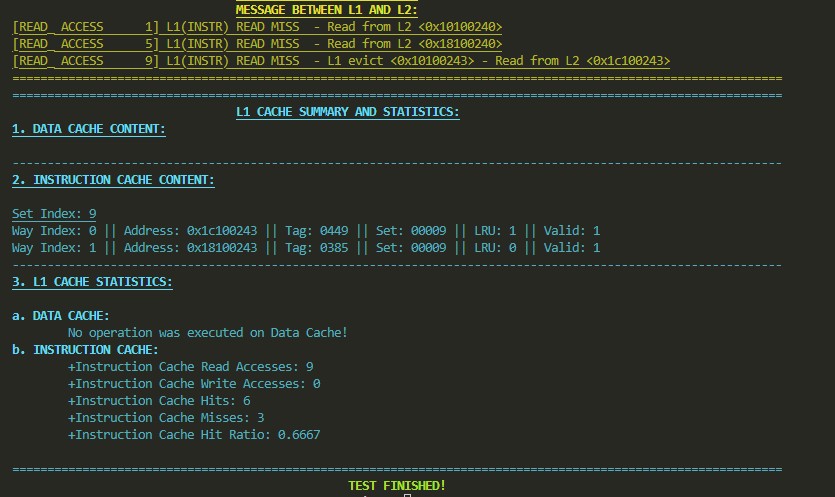
1. **L1 Instruction Cache – Fetch Test:**
2. **Test Sequence**:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Sequence | Expected Result | Comment |
| 1 | 2 0x10100240 | S9\_L0:V1/1||L1:V0/0 | UM |
| 2 | 2 0x10100241 | S9\_L0:V1/1||L1:V0/0 | L0\_H |
| 3 | 2 0x10100242 | S9\_L0:V1/1||L1:V0/0 | L0\_H |
| 4 | 2 0x10100243 | S9\_L0:V1/1||L1:V0/0 | L0\_H |
| 5 | 2 0x18100240 | S9\_L0:V1/0||L1:V1/1 | UM |
| 6 | 2 0x18100241 | S9\_L0:V1/0||L1:V1/1 | L1\_H |
| 7 | 2 0x18100242 | S9\_L0:V1/0||L1:V1/1 | L1\_H |
| 8 | 2 0x18100243 | S9\_L0:V1/0||L1:V1/1 | L1\_H |
| 9 | 2 0x1C100243 | S9\_L0:V1/1||L1:V1/0 | CM |

1. **Trace file**:

|  |
| --- |
| 2 10100240  2 10100241  2 10100242  2 10100243  2 18100240  2 18100241  2 18100242  2 18100243  2 1C100243  9 |

1. **Test Result**:



1. **Comment**:

* The simulated result is the same as what we expected!

1. **Eviction from (shared) L2 Cache Test:**

* Notation Lx\_MBI: Miss on line x due to invalid line (This is equal to unoccupied miss ~ UM)
* Notation Lx\_INV: Invalidate line x

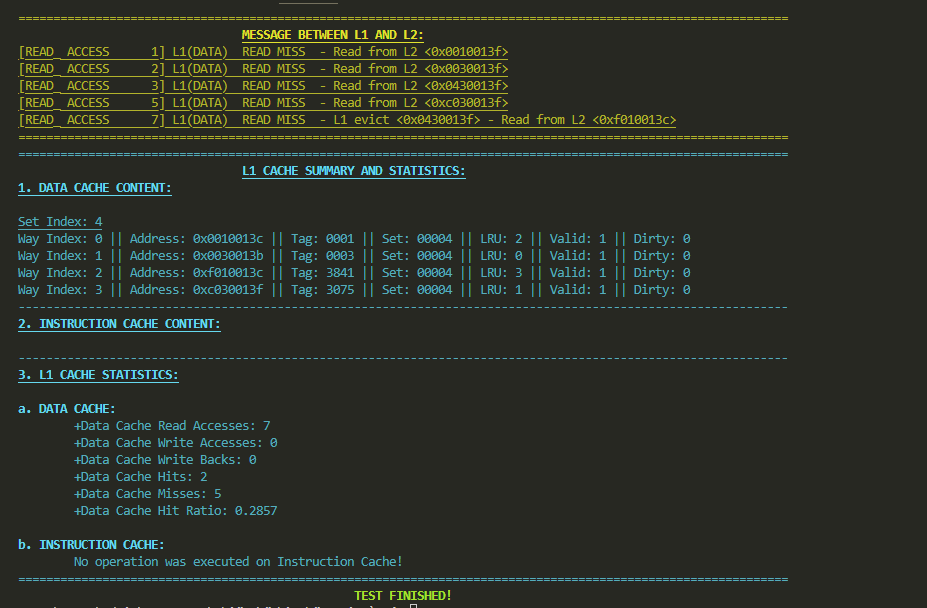
1. **Test Sequence**:

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Sequence | Expected Result | Comment |
| 1 | 0 0x0010013F | S4\_L0:V1/D0/3||L1:V0/D0/0||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 2 | 0 0x0030013F | S4\_L0:V1/D0/2||L1:V0/D0/3||L2:V0/D0/0||L3:V0/D0/0 | UM |
| 3 | 0 0x0430013F | S4\_L0:V1/D0/1||L1:V0/D0/2||L2:V0/D0/3||L3:V0/D0/0 | UM |
| 4 | 0 0x0030013B | S4\_L0:V1/D0/1||L1:V0/D0/3||L2:V0/D0/2||L3:V0/D0/0 | L1\_H |
| 5 | 0 0xC030013F | S4\_L0:V1/D0/0||L1:V1/D0/2||L2:V1/D0/1||L3:V1/D0/3 | UM |
| 6 | 0 0x0010013C | S4\_L0:V1/D0/3||L1:V1/D0/1||L2:V1/D0/0||L3:V1/D0/2 | L0\_H |
| 7 | 0 0xF010013C | S4\_L0:V1/D0/2||L1:V1/D0/0||L2:V1/D0/3||L3:V1/D0/1 | CM |
| 8 | 3 0x0010013E | S4\_L0:V0/D0/2||L1:V1/D0/0||L2:V1/D0/3||L3:V1/D0/1 | L0\_INV |
| 9 | 0 0x0010013D | S4\_L0:V1/D0/3||L1:V1/D0/0||L2:V1/D0/2||L3:V1/D0/1 | L0\_MBI |

1. **Trace file**:

|  |
| --- |
| 0 0010013F  0 0030013F  0 0430013F  0 0030013B  0 C030013F  0 0010013C  0 F010013C  3 0010013E  0 0010013D  9 |

1. **Test Result**:



1. **Comment**:

* The simulated result is the same as what we expected!

1. **Conclusion**:

As we can see, the simulated results match all the expected result in test II.1 to II.5. In the last test, the simulated result is similar to the executed program in Ripes. This proves the reliability and accuracy of our cache simulation.