Scenario 1: Load all the Data Cache Lines of a particular Set (index-0x3784)

0 984DE132
0 116DE12F
0 100DE130
0 999DE12E
0 645DE10A
0 846DE107
0 211DE128
0 777DE133

(Note: Initially all the cache lines are initialized to invalid state. And since the first write is write through it goes to Exclusive state)

Also during the first time processor reads data, it goes to exclusive state because no other processor is having any shared data. During a Read miss the processor first checks the other processor's L1 cache first and if no other processor has the data then it fetches copy of data from main memory and writes it to cache. If all the cache lines are filled at this point in the cache, then go for least recently used cache line and evict it to accommodate the new data. If the evicted line is in modified state then it has to be written back to the memory before eviction

Assume : LRU- 000 , MRU -111

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	984	116	100	999	645	846	211	777
LRU bits	000	001	010	011	100	101	110	111
MESI	E	E	E	E	E	E	E	E

Scenario 1 Explanation:

Step 1: 0 984DE132 - Read Operation -**Miss**

Binary Representation of 984DE132 is 1001 1000 0100 1101 1110 0001 0011 0010.

Byte offset: 11 0010
Index: 1101 1110 0001 00 (0x3784)
Tag: 1001 1000 0100 (0x984)

This loads Way-1 with 984

Step 2: 0 116DE12F - Read Operation - **Miss**

Binary Representation of 116DE12F is 0001 0001 0110 1101 1110 0001 0010 1111.

Byte offset: 10 1111.
Index: 1101 1110 0001 00 (0x3784, note this is the same set as set in Step 1.)
Tag: 0001 0001 0110 (0x116) This loads Way-2 with 116

Step 3: 0 100DE130 - Read Operation - **Miss**
Binary Representation of 100DE130 is 0001 0000 0000 1101 1110 0001 0011 0000.

Byte offset: 11 0000.
Index: 1101 1110 0001 00 (0x3784)
Tag: 0001 0000 0000 (0x100)

This loads Way-3 with 100

Step 4: 0 999DE12E - Read Operation - **Miss**

Binary Representation of 999DE12 is 1001 1001 1001 1101 1110 0001 0010 1111

Byte offset: 01 1111
Index: 1101 1110 0001 00 (0x3784)
Tag: 1001 1001 1001 (0x999)

This loads Way-4 with 999

Step 5: 0 645DE10A - Read Operation - **Miss**

Binary Representation of 645DE10A is 0110 0100 0101 1101 1110 0001 0000 1010

Byte offset: 00 1010
Index: 1101 1110 0001 00 (0x3784)
Tag: 0110 0100 0101 (0x645)

This loads Way-5 with 645

Step 6: 0 846DE107 - Read Operation - **Miss**

Binary Representation of 846DE107 is 1000 0100 0110 1101 1110 0001 0000 0111

Byte offset: 00 0111
Index: 1101 1110 0001 00 (0x3784)
Tag: 1000 0100 0110 (0x846)

This loads Way-6 with 846

Step 7: 0 211DE128 - Read Operation - Miss
Binary Representation of 211DE128 is 0010 0001 0001 1101 1110 0001 0010 1000

Byte offset: 10 1000
Index: 1101 1110 0001 00 (0x3784)
Tag: 0010 0001 0001 (0x211)

This loads Way-7 with 211

Step 8: 0 777DE133 - Read Operation - **Miss**

Binary Representation of 777DE133 is 0111 0111 0111 1101 1110 0001 0011 0011

Byte offset: 11 0011
Index: 1101 1110 0001 00 (0x3784)
Tag: 0111 0111 0111 (0x777)

This loads Way-8 with 777

Note in this scenario, all the eight access has the same index address of 1101 1110 0001 00 (or 0x3784). Assumes LRU bit: 000, MRU bit: 111

Scenario 2: Read and Write into a particular Set for data cache (index-3784)

0 999DE132
1 116DE123
1 666DE135
1 333DE12C
0 846DE10C
0 777DE136
1 ABCDE128
0 116DE101
1 100DE101
1 AAADE101
1 EDCDE101
4 AAADE101

Scenario 2 Explanation:

Step 1 0 999DE132 - Read Operation - **Hit**

Binary Representation of 999DE132 is 1001 1001 1001 1101 1110 0001 0011 0010

Byte offset: 11 0010
Index: 1101 1110 0001 00 (0x3784)
Tag: 1001 1001 1001 (0x999)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	984	116	100	999	645	846	211	777
LRU bits	000	001	010	111	011	100	101	110
MESI	E	E	E	S	E	E	E	E

Step 2: 1 116DE123 – Write Operation - **Hit**

Binary Representation of 116DE12F is 0001 0001 0110 1101 1110 0001 0010 1111.

Byte offset: 10 1111.
Index: 1101 1110 0001 00 (0x3784)
Tag: 0001 0001 0110(0x116)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	984	116	100	999	645	846	211	777
LRU bits	000	111	001	110	010	011	100	101
MESI	E	M	E	S	E	E	E	E

Step 3: 1 666DE135 - Write Operation – **Miss. So replace using LRU. (No Write back)**

Binary Representation of 666DE135 0110 0110 0110 1101 1110 0001 0011 0101.

Byte offset: 11 0101.
Index: 1101 1110 0001 00 (0x3784)
Tag: 0110 0110 0110 (0x666)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	666	116	100	999	645	846	211	777
LRU bits	111	110	000	101	001	010	011	100

MESI	M	M	E	S	E	E	E	E

Step 4: 1 333DE12C - Write Operation - **Miss. So replace using LRU. (No Write back)**

Binary Representation of 333DE12C is 0011 0011 0011 1101 1110 0001 0010 1100

Byte offset: 10 1100
Index: 1101 1110 0001 00 (0x3784)
Tag: 0011 0011 0011 (0x333)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	666	116	333	999	645	846	211	777
LRU bits	110	101	111	100	000	001	010	011
MESI	M	M	M	S	E	E	E	E

Step 5: 0 846DE10C - Read Operation - **Hit**

Binary Representation of 846DE10C is 1000 0100 0110 1101 1110 0001 0000 1100

Byte offset: 00 1100
Index: 1101 1110 0001 00 (0x3784)
Tag: 1000 0100 0110 (0x846)

(Assume that during every read operation if the cache line is previously in Exclusive state, then consider read is done by another processor. So in this scenario the MESI bits goes from Exclusive to Shared. This is assumed so that all MESI states can be covered. If you are considering it as read from the same processor then it will remain in the Exclusive state itself and will never go to the shared state. **So consider this read from other processors.)**

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	666	116	333	999	645	846	211	777
LRU bits	101	100	110	011	000	111	001	010
MESI	M	M	M	S	E	S	E	E

Step 6: 0 777DE136- Read Operation – **Hit (Same as previous case)**

Binary Representation of 777DE136 is 0111 0111 0111 1101 1110 0001 0011 0110

Byte offset: 11 0110
Index: 1101 1110 0001 00 (0x3784)
Tag: 0111 0111 0111(0x777)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	666	116	333	999	645	846	211	777

LRU bits	100	011	101	010	000	110	001	111
MESI	M	M	M	S	E	S	E	S

Step 7: 1 ABCDE128 - Write Operation - **Miss. So replace using LRU. (No Write back)**

Binary Representation of ABCDE128 is 1010 1011 1100 1101 1110 0001 0010 1000

Byte offset: 10 1000
Index: 1101 1110 0001 00(0x3784)
Tag: 1010 1011 1100 (0xABC)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	666	116	333	999	ABC	846	211	777

LRU bits	011	010	100	001	111	101	000	110
MESI	M	M	M	S	M	S	E	S

Step 8: 0 116DE101 - Read Operation - **Hit**

Binary Representation of 116DE101 is 0001 0001 0110 1101 1110 0001 0000 0001

Byte offset: 00 0001
Index: 1101 1110 0001 00 (0x3784)
Tag: 0001 0001 0110 (0x116)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	666	116	333	999	ABC	846	211	777

LRU bits	010	111	011	001	110	100	000	101
MESI	M	M	M	S	M	S	E	S

Step 9: 1 100DE101 - Write Operation - **Miss. So replace using LRU. (No Write back)**

Binary Representation of 100DE101 is 0001 0000 0000 1101 1110 0001 0000 0001

Byte offset: 00 0001
Index: 1101 1110 0001 00 (0x3784)
Tag: 0001 0000 0000 (0x100)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
	666	116	333	999	ABC	846	100	777

Index

LRU bits	001	110	010	000	101	011	111	100
MESI	M	M	M	S	M	S	M	S

Step 10: 1 AAADE101 - Write Operation - **Miss. So replace using LRU. (No Write back)**

Binary Representation of AAADE101 is 1010 1010 1010 1101 1110 0001 0000 0001

Byte offset: 00 0001
Index: 1101 1110 0001 00 (0x3784)
Tag: 1010 1010 1010 (0xAAA)

Index	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
	666	116	333	AAA	ABC	846	100	777

LRU bits	000	101	001	111	100	010	110	011
MESI	M	M	M	M	M	S	M	S

Step 11: 1 EDCDE101 - Write Operation - **Miss. So replace using LRU. (Write back)**

Binary Representation of EDCDE101 is 1110 1101 1100 1101 1110 0001 0000 0001

Byte offset: 00 0001
Index: 1101 1110 0001 00 (0x3784)
Tag: 1110 1101 1100 (0xEDC)

Index	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
	EDC	116	333	AAA	ABC	846	100	777

LRU bits	111	100	000	110	011	001	101	010
MESI	M	M	M	M	M	S	M	S

In this case, Way-1 was previously in M state and again a Write operation happens in this step. So the previously modified content is written back to the memory before step 11 happens.

Step 12: 4 AAADE101 - **SNOOP (RFO scenario)**

This is a two step process. AAA in previous case points to Modified state. The modified data pointed by AAA is being requested by another processor(P2) for ownership. So AAA first writes back data to Memory and shares the data to the requested processor thereby going to shared state. Once the requested processor gets the shared data it modifies it(Read For Ownership first, then write) and goes to Exclusive state(P2) and broadcasts the Invalidate signal to all other processors. So AAA of processor(P1) goes to Invalid state finally .

Binary Representation of AAADE101 is 1010 1010 1010 1101 1110 0001 0000 0001

Byte offset: 00 0001
Index: 1101 1110 0001 00 (0x3784)
Tag: 1010 1010 1010 (0xAAA)

	Way-1	Way-2	Way-3	Way-4	Way-5	Way-6	Way-7	Way-8
Index	EDC	116	333	AAA	ABC	846	100	777
LRU bits	110	100	000	111	011	001	101	010
MESI	M	M	M	I	M	S	M	S

Note to students:

In this case when the cache line is Invalid keep the LRU as 111. The next time when a write miss occurs check for the cache line with MESI bit as "Invalid". If there is no Invalid cache line go for the cache line with LRU bits 000 and evict and replace it (if needed write it back to memory).

If there is just one invalid cache line in that particular set during a read/write miss then new data is written to the cache line with invalid MESI bit instead of evicting any other useful data in the cache. Now update the LRU of this to 111.

If there are multiple cache lines with Invalid MESI bits during a read/write miss then again instead of evicting useful data, you can write the new data to one of the invalid cache lines which has the least LRU Value.

So in the above scenario the LRU bits of AAA will be 111. The next time a read/write miss occurs , instead of evicting 333 in way3 , you will write the new data to Way-4 as it is in invalid state. LRU bits have to be updated accordingly if needed. Give preference to cache line that is invalid before checking the LRU bits during the read/write miss.

P.S -> The above two scenarios are explained for different operations only to a particular index (3784 in decimal or 1101 1110 0001 00 in HEX) in the D-cache.

Additional Notes:

1) If this processor (acting processor and the processor we are modeling/simulating) has a cache line in M state, there is a read to the same cache line, we assume this read is from the acting processor (my own processor), NOT from any other processor. So this project **won't have a case** that another processor is doing a read to the same cache line of the acting processor while the acting processor's cache line is in M state, and this acting processor's cache in M state changes to S state since the assumption is reads are from the acting processor (my own processor). **In summary, there won't be a transition of M->S in this case.**

2) If the acting processor is in E state, there is a read to the same cache line, **here we assume this read is from the other processor**, and the acting processor will change cache line state from E to S. That is **the ONLY case in this project** that we assume the read is from another processor.