ECE-585 -FINAL PROJECT

DESIGN AND SIMULATION OF A SPLIT L1 CACHE

GROUP MEMBERS-

Minoti Sanjay Karkhanis

Tanaya Nandkishor Vichare

Pavithra Yuvaraja

1. BLOCK DIAGRAM

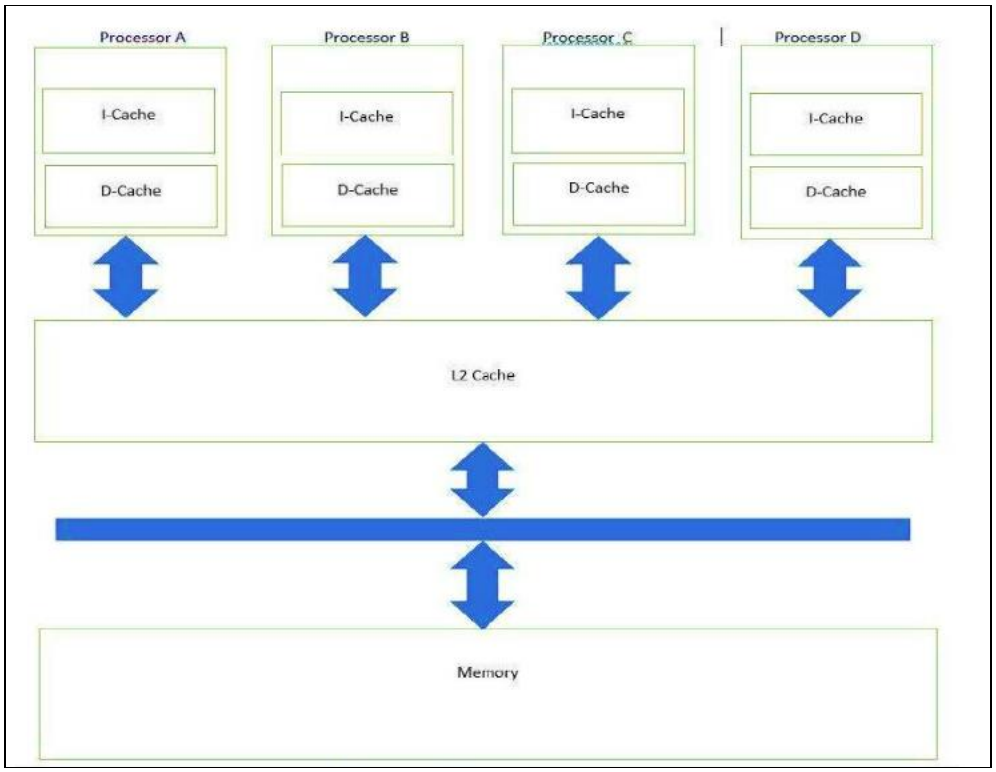

Figure 1 – BLOCK DIAGRAM

Description-

- The L1 split cache as the name suggests consists of two physically separate parts- where one part, called the Instruction cache, is dedicated for holding instructions and the other, called the Data cache, is dedicated for holding data.
- L1 instruction cache is four-way set associative and consists of 16K sets and 64-byte lines. The L1 data cache is eight-way set associative and consists of 16K sets of 64-byte lines. Both cache employ LRU replacement policy and are backed by a L2 cache.
- The L1 Cache can be used with up to 3 other processors with shared memory.
- The L1 and L2 cache employs cache inclusivity (meaning data stored in the L1 cache is also duplicated in the L2 cache)

2. DESIGN SPECIFICATIONS

2.1 Addressing

The processor is a 32-bit processor

As there are 64-byte lines = 2^6, so there are 6 address bits for Byte Select bits (bit [5:0])

16K sets cache = 2^14, so there are 14 address bits for index (bit [19:6])

So, Tag bits = 32 – 6 – 14 = 12 bits (bit [31:20])

2.2 Cache Design

The L1 split cache is split into Data Cache and Instruction Cache.
Data cache is 8 way set associative with 16K sets and instruction cache is 4 way set associative with 16K sets.
The first write should be a write through i.e. we need to write the data to L1,L2 and main memory.
The next write is a write back that means whenever a modified cache line needs to be evicted the data should be written back to main memory.
The cache is implemented using MESI protocol and True LRU.

Cache hit: Tag bits match and Data/Instruction is found in cache.

Cache miss: Tag bits do not match. Data/Instruction not found in cache. So Read from L2 cache.

We are taking the inputs from a file named “trace.txt” The “N” (opcode) and “address” is then bifurcated.

The description of N is given below:

N=0 Read data request to L1 data cache

- The CPU makes a read data request to L1 data cache. First the code will check if the Valid bit is 0 or 1 and then compare the Tag bits.
- If the tag bits match it is a hit. Then we check if the cache line was in which MESI state.
- If it is in Invalid state then data is read from L2 cache and written to L1 data cache. The CPU then reads the data. As only our processor has the data now the MESI bits go from

Invalid to Exclusive.

- If the state is not invalid and the tags match, then it is a hit.
- The code checks the previous MESI bits and accordingly changes the MESI bits.
- If the previous MESI bits are Exclusive and it is a hit then the other processor reads the data in the cache and the MESI bits go from Exclusive to Shared.
- If the previous MESI bits are Shared and it is a hit then the other processor reads the data in the cache and the MESI bits remain in the Shared state for that cache line.
- If the previous MESI bits are Modified and it is a hit then the other processor reads the data in the cache and the MESI bits remain in the Modified state for that cache line.
- If any of the above conditions are not satisfied then it might be a compulsory miss that means there is no data in the cache. In this loop the data is read from L2. The CPU then reads the data from L1 and the cache line goes to Exclusive state.
- If that condition is also not satisfied then it is a Capacity miss meaning the Set is full and the LRU line needs to be evicted. If the LRU line is modified then the data is written back to the main memory so that the data is not lost. Then read the new data from L2 and then the CPU reads the data from L1. If the LRU line which is to be evicted is in Exclusive or Shared state then simply evict the line (no need to write back) and then read the new data from L2. Once the data is available the CPU reads the data from L1.

N=1 Write data to L1 data cache

- The CPU makes a write data request to L1 data cache. First the code will check if the Valid bit is 0 or 1 and then compare the Tag bits.
- If the tag bits match it is a hit. Then we check if the cache line was in which MESI state previously.
- If it is in Invalid state then the cache line is read from L2 and written to the cache. The CPU can now write to this cache line. As the data is now changed then the cache line will go from Invalid to Modified state.
- If the state is not invalid and the tags match, then it is a hit it means that the line is present in the cache and now we can write to that line
- The code checks the previous MESI bits and accordingly changes the MESI bits.
- If the previous MESI bits are Exclusive and it is a hit then the line is present in cache and the CPU can write to that line. The cache line goes from Exclusive to Modified state
- If the previous MESI bits are Shared and it is a hit then the line is present in cache and the CPU can write to that line. The cache line goes from Shared to Modified state
- If the previous MESI bits are Modified and it is a hit then the line is present in cache and the CPU can write to that line. The cache line stays in Modified state.
- If any of the above conditions are not satisfied then it might be a compulsory miss that means there is no data in the cache. In this loop the cache line is read from L2. The CPU then writes the data to the L1 cache and the MESI bits change to Modified state.
- If that condition is also not satisfied then it is a Capacity miss meaning the Set is full and the LRU line needs to be evicted. If the LRU line is modified then the data is written back to the main memory so that the data is not lost. Then read the cache line from L2

and then the CPU writes data to L1. If the LRU line which is to be evicted is in Exclusive or Shared state then simply evict the line (no need to write back) and then write new data to L1. The MESI bits change to Modified state.

N=2 Instruction fetch (A read request to L1 instruction cache)

- The CPU makes a read request to L1 data cache. First the code will check if the Valid bit is 0 or 1 and then compare the Tag bits.
- If the tag bits match it is a hit. Then we check if the cache line was in which MESI state.
- If it is in Invalid state, then instruction is read from L2 cache and written to L1 instruction cache. The CPU then reads the instruction. As only our processor has the data now the MESI bits go from Invalid to Exclusive.
- If the state is not invalid and the tags match, then it is a hit.
- The code checks the previous MESI bits and accordingly changes the MESI bits.
- If the previous MESI bits are Exclusive and it is a hit then the CPU reads the instruction in the cache and the MESI bits stay in the Exclusive state.
- If the previous MESI bits are Shared and it is a hit then the other processor reads the instruction in the cache and the MESI bits remain in the Shared state for that cache line.
- If the previous MESI bits are Modified and it is a hit then the other processor reads the instruction in the cache and the MESI bits remain in the Modified state for that cache line.
- If any of the above conditions are not satisfied, then it might be a compulsory miss that means there is no instruction in the cache. In this loop the instruction is read from L2. The CPU then reads the instruction from L1 and the cache line goes to Exclusive state.
- If that condition is also not satisfied then it is a Capacity miss meaning the Set is full and the LRU line needs to be evicted. If the LRU line is modified then the instruction is written back to the main memory so that the instruction is not lost. Then read the new instruction from L2 and then the CPU reads the instruction from L1. If the LRU line which is to be evicted is in Exclusive or Shared state then simply evict the line (no need to write back) and then read the new instruction from L2. Once the instruction is available the CPU reads the instruction from L1. The L1 cache line MESI bits go from Modified, Exclusive or Shared to Exclusive.

N=3 Invalidate command from L2

- If the cache line is in shared MESI state and the other processor modifies the data or instruction of that line then the data/instruction in our processor becomes invalid.
- The code first checks if the line is valid and the tags match or not.
- If it is valid and the tags match then the data/instruction has been modified by the other processor and the L2 should send our processor the Invalidate command and also make that cache line Invalid (00).

N=4 Data request to L2 (in response to snoop)

- In this case the other processor wants an RFO to a particular data line.
- We first check if the line is Valid and the tags match or not.
- If the Tags match then the line is in our cache .
- We first need to check the MESI state of the data line in our Processor.
- If the cache data line is in modified state then we need to write this line to the main memory so that the data is not lost. Then the Processor gets the RFO and then modifies the cache data line. As the data line is now modified for the other processor the data present in our cache is now invalid and thus our Processor's cache data line goes to Invalid state.
- If the cache data line is in Exclusive state then no need to write back to the memory. The other Processor gets the RFO and changes the data. Now, our Processor's data line goes to Invalid state.
- If the cache data line is in Shared state then no need to write back to the memory. The other Processor gets the RFO and changes the data. Now, our Processor's data line goes to Invalid state.

N=8 Clear the cache and reset all states

- For this opcode clear the cache and all its states.
- Clear the cache hit, cache miss and cache hit ratio.
- The code goes to initialize() function to clear all the states.

N=9 Print contents and state of the cache

- If opcode is 9 print contents and state of the cache

3. LRU REPLACEMENT POLICY

- We have implemented the True LRU policy in our design.
- As the Data Cache is 8 way set associative, we need 3 bits LRU and for Instruction Cache we need 2 bits LRU as it is a 4 way set associative.
- For Data Cache- Whenever we reference a line in the set, we make 000 as MRU, we reset that line's counter to 000 and we increment other lines' counter if their value > current counter. The same logic is used for the Instruction cache as well using bits.
- We have used two different functions for Data cache and Instruction cache.

4. MESI PROTOCOL

- We have used two functions to implement the MESI protocol, one for instruction cache and the other one for data cache.
- The transition of MESI bits is discussed in section 2.2 of this report.
- We have initialized each state by giving a parameter

MESI state	Parameter
Invalid	00
Modified	01
Exclusive	10
Shared	11

5. ALGORITHM

- Initialize the cache parameters
- Open and check if file valid
- If valid, get first character then one of the cases are selected:
- Case 0: If opcode ‘0’ then read L1 data cache

  Check index, valid and tag bits.

  If valid=1 and tags match increase hit.

  If all ways are filled and tags don’t match then it is conflict miss, get data from L2 and evict with respect to LRU.

  If all ways are not filled and tags do not match then it is compulsory miss.

  Increment the number of reads.

- Case 1: If opcode ‘1’, write data to L1 data cache.

  Check index, valid, tag bits.

  If tags match and valid=1, that is a hit. Thus increase number of writes and hits.

  If all valid ways fill and tags do not match

  Then evict a line based on LRU.

  Get the required line and update it with new data.

- Case 2: If opcode ‘2’, read from L1 instruction cache.
- Case 3: If opcode ‘3’, invalidate corresponding line from cache.

  Check if tag bits match and valid bit=0

  If they match set valid=0

- Case 4: If opcode ‘4’ then check the state, give RFO to the other processor and make the cache line invalid.
- Case 8: If opcode ‘8’, clear cache and its states.
- Case 9: If opcode ‘9’, print contents and state of the cache.
- Display and stop.

6. TEST CASES

TRACE 1

0 984DE132
0 116DE12F
0 100DE130
0 999DE12E
0 645DE10A
0 846DE107
0 211DE128
0 777DE133
0 999DE132
1 116DE123
1 666DE135
1 333DE12C
0 846DE10C
9 846DE10C

TRANSCRIPT 1

Transcript

```
#
#
#
# Tag_DC  LRU_DC  MESI_DC
# 666     010     01
# 116     011     01
# 333     001     01
# 999     100     11
# 645     111     10
# 846     000     11
# 211     110     10
# 777     101     10
#
#
#
#
# Total number of cache reads:          10
#
# Total number of cache writes:          3
#
# Total number of cache hits:            3
#
# Total number of cache miss :          10
#
# Hit ratio:  0.230769
#
```

TRACE 2

0 984DE132
0 116DE12F
0 100DE130
0 999DE12E
0 645DE10A
0 846DE107
0 211DE128
0 777DE133
0 999DE132
1 116DE123
1 666DE135
1 333DE12C
0 846DE10C
3 777DE136
8 ABCDE128
0 777DE136
1 ABCDE128
1 777DE136
1 ABCDE128
0 116DE101
1 100DE101
1 AAADE101
1 EDCDE101
2 116DE12F
2 100DE130
2 999DE12E
2 645DE10A
2 846DE107
2 116DE12F
4 AAADE101
9 846DE107

TRANSCRIPT 2