Homework 11 Solutions

Problem 1: Cache Tracing - Fully Associative (40 points)

You have a fully associative cache with 16 B of data (not counting metadata). Each cache block is 1 B (again, not counting metadata). Main memory addresses are 8 bits, and precise LRU is used as the replacement policy.

Assume that the cache is initially empty (all blocks are invalid), and fill in the table below to trace the following sequence of read accesses.

Part A: Tracing individual read accesses (24 points, 3 per row)

Complete the following table showing how each address is split into tag, index, and offset, as well as whether it is a hit or a miss and which addresses’ data, if any, it evicts. These accesses are starting in sequential order, with the first one accessing an empty cache.

Part B: Final cache state (16 points, EMRN)

Draw the final state of the cache, clearly labeling each block and set. Your drawing should clearly show all relevant valid, tag, and LRU bits in the cache. You can represent the LRU bits however you like, as long as the precise ordering of the blocks is clear.

Problem 1 Solutions (both parts)

*This is a fully associative cache, so there are no index bits. Each block is 1B, so there are also no offset bits (no need to choose from the multiple bytes in a block). This means everything is tag.*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Address** | **Tag** | **Index** | **Offset** | **H/M?** | **Address(es) evicted (if any)** |
| 1010 1111 | 1010 1111 | - | - |  |  |
| 1011 1111 | 1011 1111 | - | - |  |  |
| 1010 1111 | 1010 1111 | - | - |  |  |
| 1011 1110 | 1011 1110 | - | - |  |  |
| 1010 1111 | 1010 1111 | - | - |  |  |
| 1011 1111 | 1011 1111 | - | - |  |  |
| 1010 1110 | 1010 1110 | - | - |  |  |
| 1011 1111 | 1011 1111 | - | - |  |  |

*To fill out the other two columns, we need to trace the state of the cache. We should start by drawing it. Since there are 16 bytes of data stored in blocks that are 1 byte each, this means we have 16 blocks:*

Initial Cache State

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |
|  | 0 | - | 13th least (replace 13th) |
|  | 0 | - | 14th least (replace 14th) |
|  | 0 | - | 15th least (replace 15th) |
|  | 0 | - | 16th least (replace 16th) |

Cache State After First Access (1010 1111)

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
| Data from address 1010 1111 | 1 | 1010 1111 | 16th least (replace 16th) |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |
|  | 0 | - | 13th least (replace 13th) |
|  | 0 | - | 14th least (replace 14th) |
|  | 0 | - | 15th least (replace 15th) |

Cache State After Second Access (1011 1111)

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
| Data from address 1010 1111 | 1 | 1010 1111 | 15th least (replace 15th) |
| Data from address 1011 1111 | 1 | 1011 1111 | 16th least (replace 16th) |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |
|  | 0 | - | 13th least (replace 13th) |
|  | 0 | - | 14th least (replace 14th) |

Cache State After Third Access (1010 1111)

*The third access, to 1010 1111, is a hit, so we just need to update the LRU field.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
| Data from address 1010 1111 | 1 | 1010 1111 | 16th least (replace 16th) |
| Data from address 1011 1111 | 1 | 1011 1111 | 15th least (replace 15th) |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |
|  | 0 | - | 13th least (replace 13th) |
|  | 0 | - | 14th least (replace 14th) |

Cache State After Fourth Access (1011 1110)

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
| Data from address 1010 1111 | 1 | 1010 1111 | 15th least (replace 15th) |
| Data from address 1011 1111 | 1 | 1011 1111 | 14th least (replace 14th) |
| Data from address 1011 1110 | 1 | 1011 1110 | 16th least (replace 16th) |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |
|  | 0 | - | 13th least (replace 13th) |

Cache State After Fifth and Sixth Accesses (1010 1111 and 1011 1111)

*Both of these accesses are hits, so they just update LRU.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
| Data from address 1010 1111 | 1 | 1010 1111 | 15th least (replace 15th) |
| Data from address 1011 1111 | 1 | 1011 1111 | 16th least (replace 16th) |
| Data from address 1011 1110 | 1 | 1011 1110 | 14th least (replace 14th) |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |
|  | 0 | - | 13th least (replace 13th) |

Cache State After Seventh Access (1010 1110)

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
| Data from address 1010 1111 | 1 | 1010 1111 | 14th least (replace 14th) |
| Data from address 1011 1111 | 1 | 1011 1111 | 15th least (replace 15th) |
| Data from address 1011 1110 | 1 | 1011 1110 | 13th least (replace 13th) |
| Data from address 1010 1110 | 1 | 1010 1110 | 16th least (replace 16th) |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |

Final Cache State (Final answer to Part B)

*The last access is a hit, so once again, we just update LRU bits.*

|  |  |  |  |
| --- | --- | --- | --- |
| **Data** | **Valid** | **Tag** | **LRU** |
| Data from address 1010 1111 | 1 | 1010 1111 | 14th least (replace 14th) |
| Data from address 1011 1111 | 1 | 1011 1111 | 16th least (replace 16th) |
| Data from address 1011 1110 | 1 | 1011 1110 | 13th least (replace 13th) |
| Data from address 1010 1110 | 1 | 1010 1110 | 15th least (replace 15th) |
|  | 0 | - | Least recent (replace 1st) |
|  | 0 | - | 2nd least (replace 2nd) |
|  | 0 | - | 3rd least (replace 3rd) |
|  | 0 | - | 4th least (replace 4th) |
|  | 0 | - | 5th least (replace 5th) |
|  | 0 | - | 6th least (replace 6th) |
|  | 0 | - | 7th least (replace 7th) |
|  | 0 | - | 8th least (replace 8th) |
|  | 0 | - | 9th least (replace 9th) |
|  | 0 | - | 10th least (replace 10th) |
|  | 0 | - | 11th least (replace 11th) |
|  | 0 | - | 12th least (replace 12th) |

Final answer to Part A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Address** | **Tag** | **Index** | **Offset** | **H/M?** | **Address(es) evicted (if any)** |
| 1010 1111 | 1010 1111 | - | - | miss | - |
| 1011 1111 | 1011 1111 | - | - | miss | - |
| 1010 1111 | 1010 1111 | - | - | hit | - |
| 1011 1110 | 1011 1110 | - | - | miss | - |
| 1010 1111 | 1010 1111 | - | - | hit | - |
| 1011 1111 | 1011 1111 | - | - | hit | - |
| 1010 1110 | 1010 1110 | - | - | miss | - |
| 1011 1111 | 1011 1111 | - | - | hit | - |

Problem 2: Cache Tracing - FA with 2-byte blocks (40 points)

You have a fully associative cache with 16 B of data (not counting metadata). **Each cache block is now 2 B (again, not counting metadata).** Main memory addresses are 8 bits, and precise LRU is used as the replacement policy.

Assume that the cache is initially empty (all blocks are invalid), and fill in the table below to trace the following sequence of read accesses.

Part A: Tracing individual read accesses (24 points, 3 per row)

Complete the following table showing how each address is split into tag, index, and offset, as well as whether it is a hit or a miss and which addresses’ data, if any, it evicts. These accesses are starting in sequential order, with the first one accessing an empty cache.

Part B: Final cache state (16 points, EMRN)

Draw the final state of the cache, clearly labeling each block and set. Your drawing should clearly show all relevant valid, tag, and LRU bits in the cache. You can represent the LRU bits however you like, as long as the precise ordering of the blocks is clear.

Problem 2 Solutions (both parts)

*This is a fully associative cache, so there are no index bits. Each block is 2B, so we need one offset bit to choose between the 2 bytes in a block. The other 7 bits of the address will be the tag:*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Address** | **Tag** | **Index** | **Offset** | **H/M?** | **Address(es) evicted (if any)** |
| 1010 1111 | 1010 111 | - | 1 |  |  |
| 1011 1111 | 1011 111 | - | 1 |  |  |
| 1010 1111 | 1010 111 | - | 1 |  |  |
| 1011 1110 | 1011 111 | - | 0 |  |  |
| 1010 1111 | 1010 111 | - | 1 |  |  |
| 1011 1111 | 1011 111 | - | 1 |  |  |
| 1010 1110 | 1010 111 | - | 0 |  |  |
| 1011 1111 | 1011 111 | - | 1 |  |  |

Initial Cache State

*To fill out the other two columns, we need to trace the state of the cache. We should start by drawing it. Since there are 16 bytes of data stored in blocks that are 2 byte each, this means we have 8 blocks:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data[0]** | **Data[1]** | **V** | **Tag** | **LRU** |
|  |  | 0 | - | Least recent (replace 1st) |
|  |  | 0 | - | 2nd least (replace 2nd) |
|  |  | 0 | - | 3rd least (replace 3rd) |
|  |  | 0 | - | 4th least (replace 4th) |
|  |  | 0 | - | 5th least (replace 5th) |
|  |  | 0 | - | 6th least (replace 6th) |
|  |  | 0 | - | 7th least (replace 7th) |
|  |  | 0 | - | 8th least (replace 8th) |

Cache State After First Access (1010 1111)

*This is a miss, so we bring in the data at addresses 1010 1110 and 1010 1111 into the first block:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data[0]** | **Data[1]** | **V** | **Tag** | **LRU** |
| Data from 1010 1110 | Data from 1010 1111 | 1 | 1010 111 | 8th least (replace 8th) |
|  |  | 0 | - | Least recent (replace 1st) |
|  |  | 0 | - | 2nd least (replace 2nd) |
|  |  | 0 | - | 3rd least (replace 3rd) |
|  |  | 0 | - | 4th least (replace 4th) |
|  |  | 0 | - | 5th least (replace 5th) |
|  |  | 0 | - | 6th least (replace 6th) |
|  |  | 0 | - | 7th least (replace 7th) |

Cache State After Second Access (1011 1111)

*This is also a miss, since the tag doesn’t match any valid block.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data[0]** | **Data[1]** | **V** | **Tag** | **LRU** |
| Data from 1010 1110 | Data from 1010 1111 | 1 | 1010 111 | 7th least (replace 7th) |
| Data from 1011 1110 | Data from 1011 1111 | 1 | 1011 111 | 8th least (replace 8th) |
|  |  | 0 | - | Least recent (replace 1st) |
|  |  | 0 | - | 2nd least (replace 2nd) |
|  |  | 0 | - | 3rd least (replace 3rd) |
|  |  | 0 | - | 4th least (replace 4th) |
|  |  | 0 | - | 5th least (replace 5th) |
|  |  | 0 | - | 6th least (replace 6th) |

Cache State After Third Access (1010 1111)

*This is a hit, so we update LRU:*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data[0]** | **Data[1]** | **V** | **Tag** | **LRU** |
| Data from 1010 1110 | Data from 1010 1111 | 1 | 1010 111 | 8th least (replace 8th) |
| Data from 1011 1110 | Data from 1011 1111 | 1 | 1011 111 | 7th least (replace 7th) |
|  |  | 0 | - | Least recent (replace 1st) |
|  |  | 0 | - | 2nd least (replace 2nd) |
|  |  | 0 | - | 3rd least (replace 3rd) |
|  |  | 0 | - | 4th least (replace 4th) |
|  |  | 0 | - | 5th least (replace 5th) |
|  |  | 0 | - | 6th least (replace 6th) |

Cache State After Fourth Access (1011 1110)

*This is a hit, since its tag matches the tag of the second block. We just update LRU.*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Data[0]** | **Data[1]** | **V** | **Tag** | **LRU** |
| Data from 1010 1110 | Data from 1010 1111 | 1 | 1010 111 | 7th least (replace 7th) |
| Data from 1011 1110 | Data from 1011 1111 | 1 | 1011 111 | 8th least (replace 8th) |
|  |  | 0 | - | Least recent (replace 1st) |
|  |  | 0 | - | 2nd least (replace 2nd) |
|  |  | 0 | - | 3rd least (replace 3rd) |
|  |  | 0 | - | 4th least (replace 4th) |
|  |  | 0 | - | 5th least (replace 5th) |
|  |  | 0 | - | 6th least (replace 6th) |

Cache State After Remaining Accesses

*These are all hits. After the fifth access, the LRU fields flip to make the first block the most recent. After the sixth, they flip back to make the second block the most recent, and we’re back to where we were after the fourth access.*

*Same story with the seventh and eighth accesses: the LRU bits flip to make the first block the most recent, and back again to make the second block the most recent. So the cache diagram above is the final state.*

Final Answer to Part A

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Address** | **Tag** | **Index** | **Offset** | **H/M?** | **Address(es) evicted (if any)** |
| 1010 1111 | 1010 111 | - | 1 | miss |  |
| 1011 1111 | 1011 111 | - | 1 | miss |  |
| 1010 1111 | 1010 111 | - | 1 | hit |  |
| 1011 1110 | 1011 111 | - | 0 | hit |  |
| 1010 1111 | 1010 111 | - | 1 | hit |  |
| 1011 1111 | 1011 111 | - | 1 | hit |  |
| 1010 1110 | 1010 111 | - | 0 | hit |  |
| 1011 1111 | 1011 111 | - | 1 | hit |  |

Problem 3: Caches in the Real World (20 points, EMRN)

Look up the specifications for a specific model of CPU (not GPU) that was released in 2018 or later. I recommend this database: [https://www.techpowerup.com/cpu-specsLinks to an external site.](https://www.techpowerup.com/cpu-specs)

Answer these questions for your chosen processor model:

* Who is the manufacturer/designer (Intel, AMD, Apple, ARM…)?
* What’s the model number?
* What’s the microarchitecture (sometimes called codename)?
* What was the year of release?
* For each level of cache (L1, L2, L3…), state:
  + Is this cache private (per-core), shared among multiple cores (how many?), or shared among all cores?
  + How large is the cache? For caches that are not shared among all the cores, make clear whether this measurement is for *each* of these caches, or the *total* across the entire processor chip.
* Cite your sources!