Chapter 09: Caches

Lesson 06:

Cache Implementation

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Objective

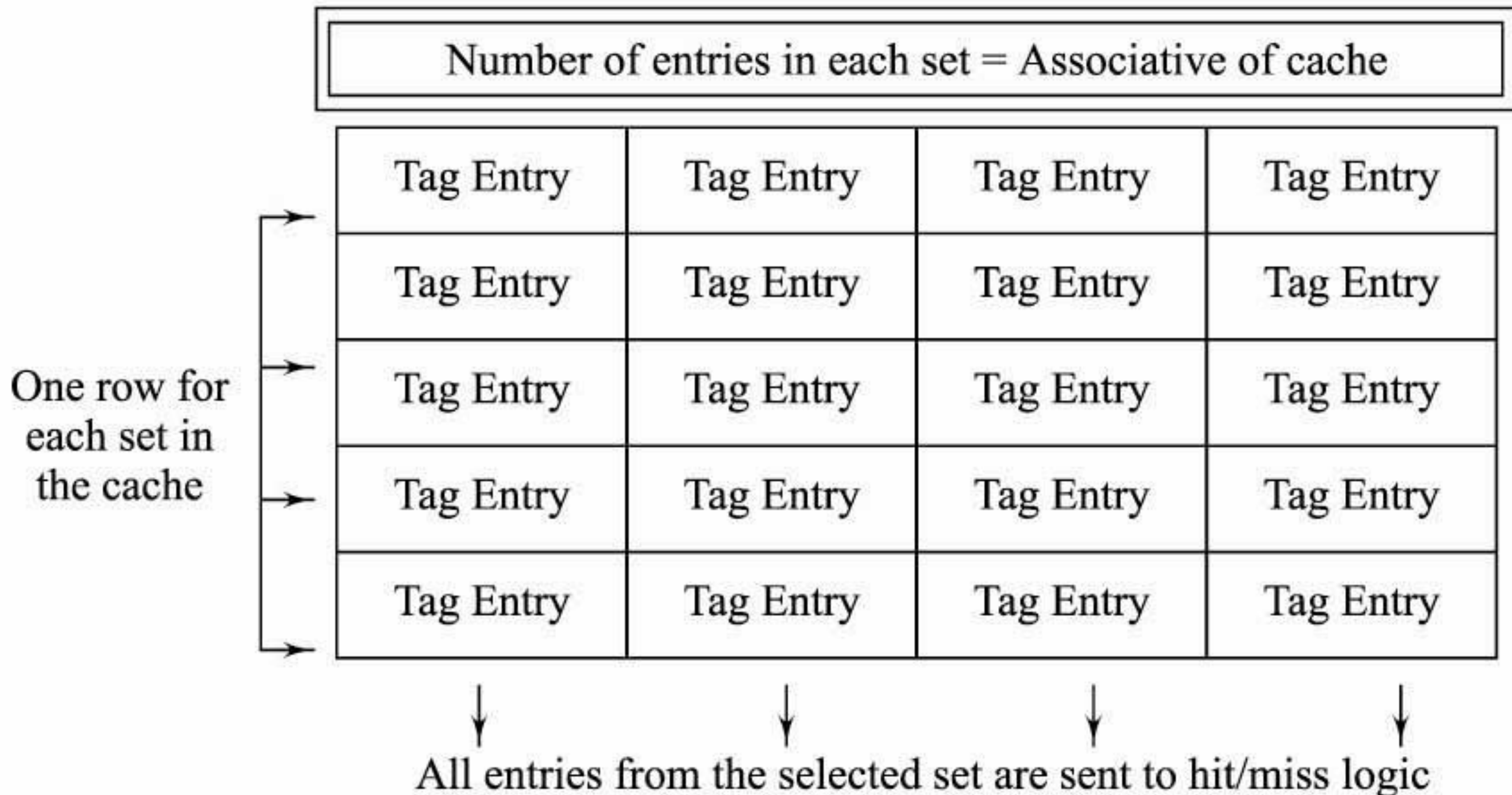
- Learn implementation of cache design by using the tag entry, and valid and dirty-bit fields
- Learn Design of hit/miss logic
- Use of data arrays
-

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Implementing tag entries

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Arrays of tag entries in Four Way Set Associative Cache



Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Tag Entry

- A tag entry contains the information necessary to record which line of data is stored in the line of the data cache that is associated with the entry
- Each entry describes one cache line of data

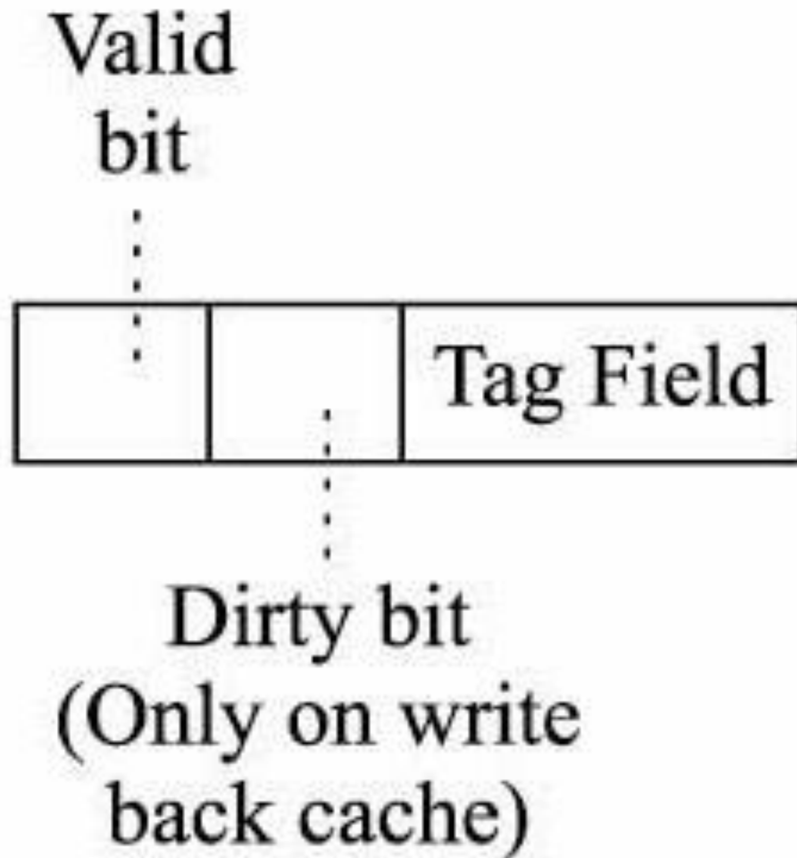
Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Tag Entry

- A tag entry consists of a tag field that contains the portion of the address of the line that is not used to select a set (the "remainder" field)
- A valid bit that records whether or not the line associated with this tag array entry contains valid data
- A dirty bit (for write- back caches)

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

A tag-array entry



Number of Entries in Each set = Associativity of cache

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Example

- Assume— the tag array of a 32-kB cache with 256-byte cache lines and four-way set-associativity
- Assume— the cache is write-back but does not require any additional bits of data in the tag array to implement the write-back policy
- Assume— system containing the cache uses 32-bit addresses
- Find how many bits of storage are required Assume that the

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Solution

- Number of cache lines = 32-kB ÷ with 256-byte = 32× 1024 ÷ 256 = 128 lines
- Number of sets in Four-way set-associative= 128/4 =32
- Number of addressing bits $m = \log_2 32 = 5 = 5$ bits
- Number of addressing bits n in the lines that are 256 bytes long = 8
- m + n = 13 bits of the address used to select a set and determine the byte within the line that an address points to

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Solution

- The tag field of each tag array entry is 32 – 13 =19 bits long
- Adding 2 bits for dirty and valid bits, we get 21 bits per tag entry
- Multiplying by the 128 lines in the cache gives 2688 bits of storage in the tag array

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Tag entry additional bits in case of using LRU or NMRU replacement policy

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Tag Entry

- Depending on the replacement policy of the cache, the tag entry might also contain one or more additional bits
- LRU replacement, each tag entry must be able to record how many of the other lines in the set have been referenced since the last time that the line it corresponds to was referenced, so that the replacement policy can locate the least recently used line when a replacement is required

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Initialization of tag entries

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Initialization when a computer is first powered up

- All of the valid bits in the tag array are set to 0
- Records the fact that there is no data in the cache
- Whenever a line is brought into the cache, the valid bit in the corresponding entry of the tag array is set to 1, recording that the line now contains valid data

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Initialization

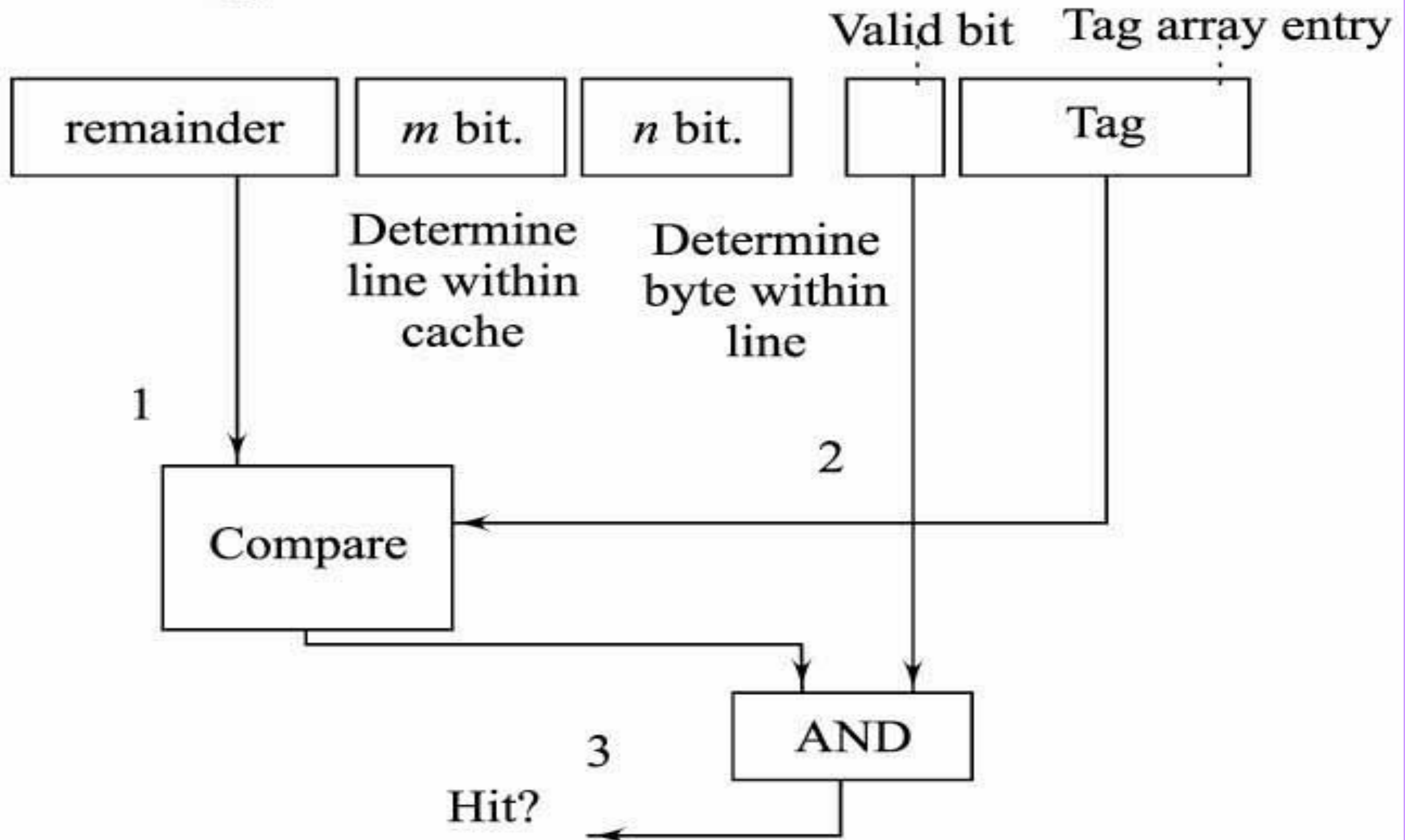
- In general, once a line becomes full, it remains full, because the data that was loaded into it remains in the cache until it is replaced with another line of data
- The exception to this is when a program deliberately removes a line of data from the cache (most processors provide instructions to do this), in which case the line becomes empty and its valid bit is reset to 0

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Implementation of hit/miss logic

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Logic for finding hit/miss in cache

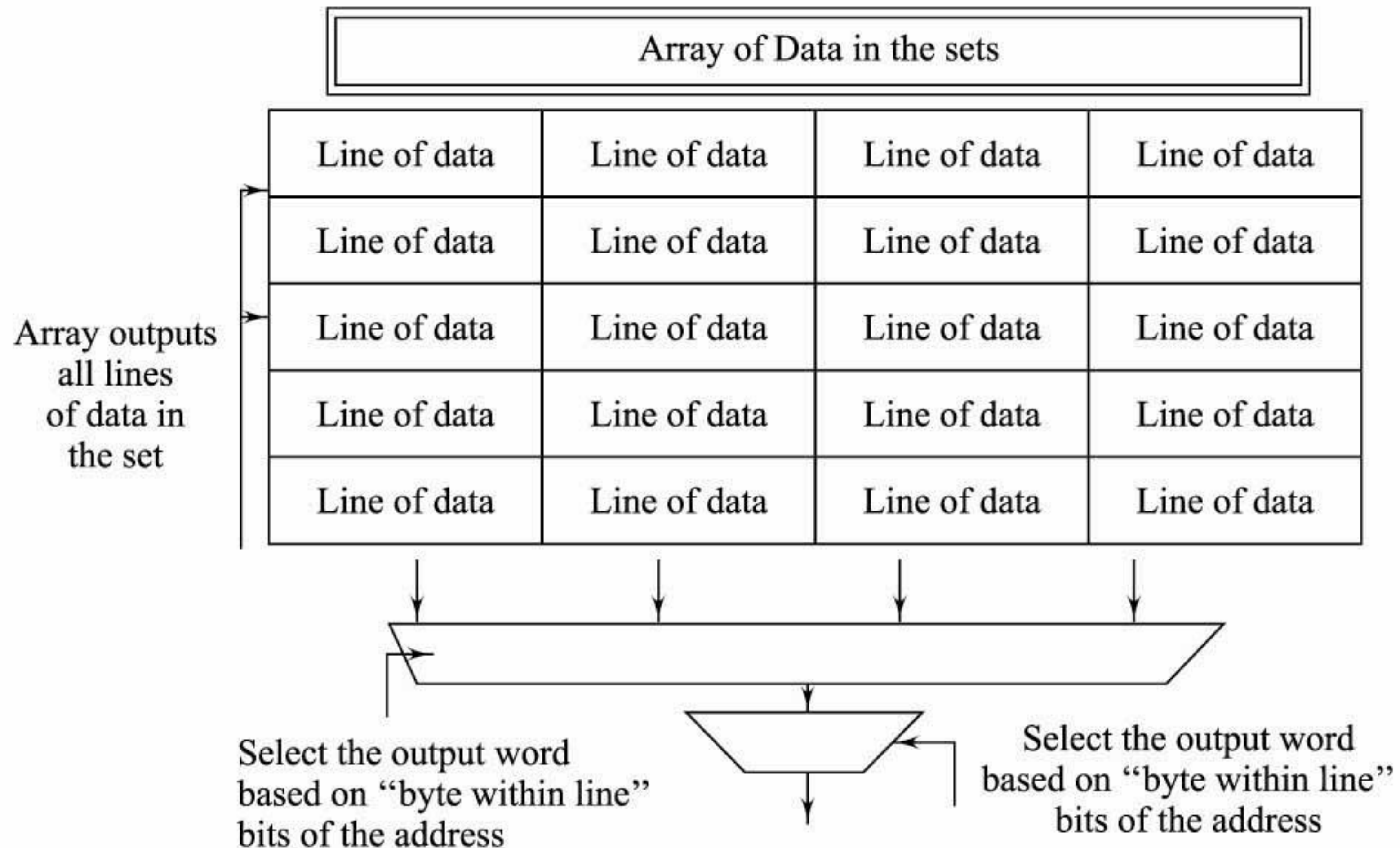


Schaum's Outline of Theory and Problems of Computer Architecture
Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Implementation of data array

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Data array organisation



Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

Summary

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

We learnt

- Cache implementation using the tag entry, and valid and dirty-bit fields
- Design of hit/miss logic
- Using data arrays

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009

End of Lesson 06 on

Cache Implementation

Copyright © The McGraw-Hill Companies Inc. Indian Special Edition 2009