Cache Mapping

COMP375 Computer Architecture and Organization

"The only problem in computer architecture that is really hard to overcome is not having enough address bits."

Gordon Bell

Exam on Wednesday

- The second exam in COMP375 will be on Wednesday, October 21, 2015
- It covers all the material since the first exam
  - Interrupts
  - Busses
  - Memory
  - VLSI
  - Cache
- You may have one 8½ by 11" page of notes

Design Project

- The project is due by 1:00pm (start of class) on **Monday, October 19**, 2015
- Be sure to include the names of both team members

Cache Challenges

- The challenge in cache design is to ensure that the desired data and instructions are in the cache. The cache should achieve a high hit ratio
- The cache system must be quickly searchable as it is checked every memory reference

Cache to Ram Ratio

- A processor might have 2 MB of cache and 2 GB of RAM
- There may be 1000 times more RAM than cache
- The cache algorithms have to carefully select the 0.1% of the memory that is likely to be most accessed

Tag Fields

- A cache line contains two fields
  - Data from RAM
  - The address of the block currently in the cache.
- The part of the cache line that stores the address of the block is called the tag field
- Many different addresses can be in any given cache line. The tag field specifies the address currently in the cache line
- Only the upper address bits are needed

Cache Array Showing full Tag

Tag	Data	Data	Data	Data
1234	from 1234	from 1235	from 1236	from 1237
2458	from 2458	form 2459	from 245A	from 245B
17B0	from 17B0	from 17B1	from 17B2	from 17B3
5244	from 5244	from 5245	from 5246	from 5247

- Addresses are 16 bytes in length (4 hex digits)
- In this example, each cache line contains four bytes.
- The upper 14 bits of each address in a line are the same
- This tag contains the full address of the first byte

Cache Array Showing Tag

Tag	Data	Data	Data	Data
48D	from 1234	from 1235	from 1236	from 1237
916	from 2458	form 2459	from 245A	from 245B
5EC	from 17B0	from 17B1	from 17B2	from 17B3
1491	from 5244	from 5245	from 5246	from 5247

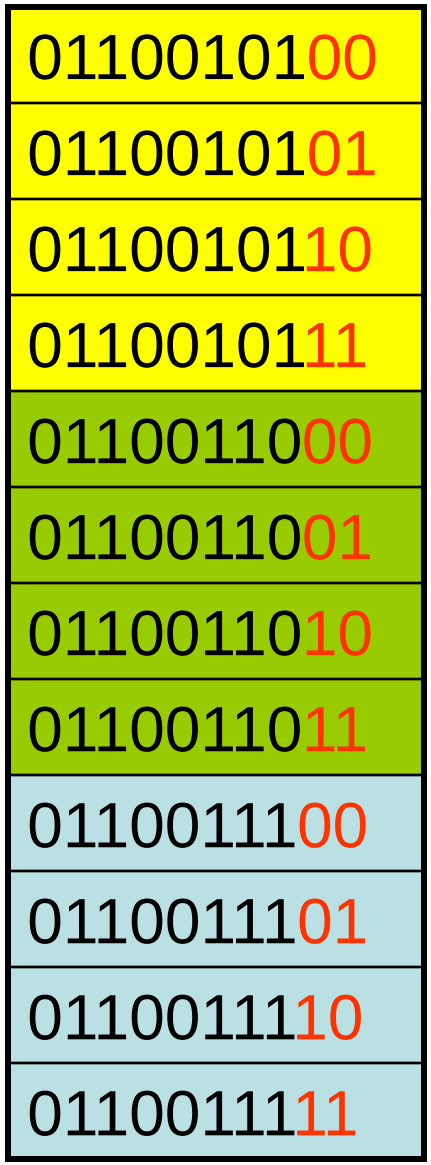
- Addresses are 16 bytes in length (4 hex digits)
- In this example, each cache line contains four bytes.
- The upper 14 bits of each address in a line are the same
- This tag contains only the upper address bits

Cache Lines

- The cache memory is divided into **blocks** or **lines**. Currently lines can range from 16 to 64 bytes
- Data is copied to and from the cache one line at a time
- The lower $\log_2$(line size) bits of an address specify a particular byte in a line

Line Example

These boxes represent RAM addresses



With a line size of 4, the offset is the $\log_2(4) = 2$ bits

The lower 2 bits specify which byte in the line

Computer Science Search

- If you ask COMP285 students how to search for a data item, they should be able to tell you

  - Linear Search – $O(n)$
  - Binary Search – $O(\log_2 n)$
  - Hashing – $O(1)$
  - Parallel Search – $O(n/p)$

Mapping

- The memory system has to quickly determine if a given address is in the cache
- There are three popular methods of mapping addresses to cache locations
  - **Fully Associative** – Search the entire cache for an address
  - **Direct** – Each address has a specific place in the cache
  - **Set Associative** – Each address can be in any of a small set of cache locations

Direct Mapping

- Each location in RAM has one specific place in cache where the data will be held
- Consider the cache to be like an array. Part of the address is used as index into the cache to identify where the data will be held
- Since a data block from RAM can only be in one specific line in the cache, it must always replace the one block that was already there. There is no need for a replacement algorithm.

Direct Cache Addressing

- The lower $\log_2$(line size) bits define which byte in the block
- The next $\log_2$(number of lines) bits defines which line of the cache
- The remaining upper bits are the tag field

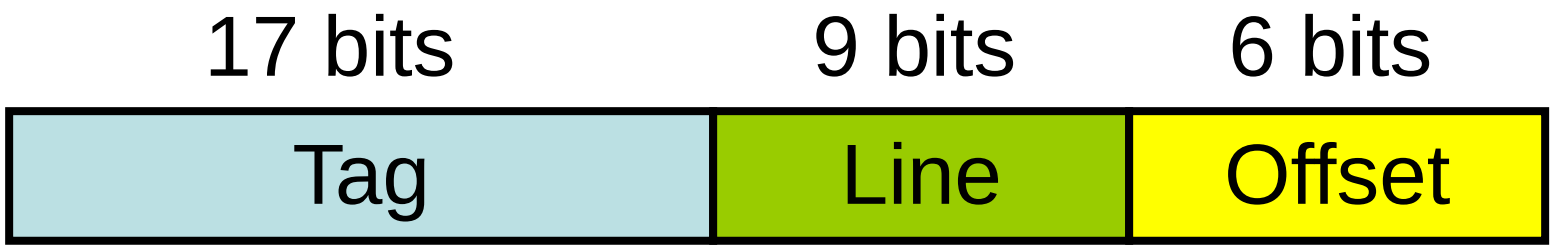
Cache Constants

- cache size / line size = number of lines
- $\log_2$(line size) = bits for offset
- $\log_2$(number of lines) = bits for cache index
- remaining upper bits = tag address bits

Example direct address

Assume you have

- 32 bit addresses (can address 4 GB)
- 64 byte lines (offset is 6 bits)
- 32 KB of cache
- Number of lines = 32 KB / 64 = 512
- Bits to specify which line = $\log_2(512) = 9$



Example Address

- Using the previous direct mapping scheme with 17 bit tag, 9 bit index and 6 bit offset

  01111101011101110001101100111000

01111101011101110	**001101100**	**111000**
Tag	Index	offset

- Compare the tag field of line 001101100

(108_{10}) for the value 01111101011101110. If it matches, return byte 111000 (56_{10}) of the line

How many bits are in the tag, line and offset fields?

Direct Mapping

24 bit addresses

64K bytes of cache

16 byte cache lines

A. tag=4, line=16, offset=4
B. tag=4, line=14, offset=6
C. tag=8, line=12, offset=4
D. tag=6, line=12, offset=6

Associative Mapping

- In associative cache mapping, the data from any location in RAM can be stored in any location in cache
- When the processor wants an address, all tag fields in the cache as checked to determine if the data is already in the cache
- Each tag line requires circuitry to compare the desired address with the tag field
- All tag fields are checked in parallel

Associative Cache Mapping

- The lower $\log_2$(line size) bits define which byte in the block
- The remaining upper bits are the tag field.
- For a 4 GB address space with 128 KB cache and 32 byte blocks:

Example Address

- Using the previous associate mapping scheme with 27 bit tag and 5 bit offset

  011111010111011100011011001110000

011111010111011100011011001	**11000**
Tag	offset

- Compare all tag fields for the value 011111010111011100011011001. If a match is found, return byte 11000 (24_{10}) of the line

How many bits are in the tag and offset fields?

Associative Mapping

24 bit addresses

128K bytes of cache

64 byte cache lines

A. tag= 20, offset=4
B. tag=19, offset=5
C. tag=18, offset=6
D. tag=16, offset=8

Set Associative Mapping

- Set associative mapping is a mixture of direct and associative mapping
- The cache lines are grouped into sets
- The number of lines in a set can vary from 2 to 16
- A portion of the address is used to specify which set will hold an address
- The data can be stored in any of the lines in the set

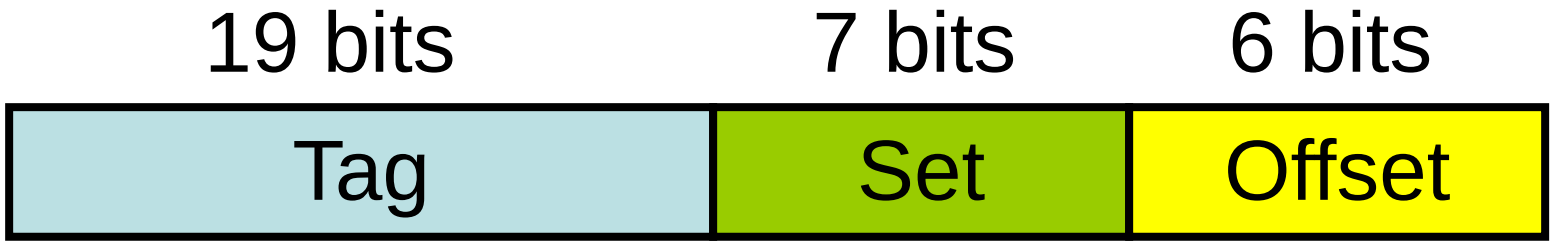
Set Associative Mapping

- When the processor wants an address, it indexes to the set and then searches the tag fields of all lines in the set for the desired address
- n = cache size / line size = number of lines
- b = $\log_2$(line size) = bit for offset
- w = number of lines / set
- s = n / w = number of sets

Example Set Associative

Assume you have

- 32 bit addresses
- 32 KB of cache 64 byte lines
- Number of lines = 32 KB / 64 = 512
- 4 way set associative
- Number of sets = 512 / 4 = 128
- Set bits = $\log_2(128) = 7$



Example Address

- Using the previous set-associate mapping with 19 bit tag, 7 bit index and 6 bit offset

  01111101011101110001101100111000

0111110101110111000	**1101100**	**111000**
Tag	Index	offset

- Compare the tag fields of lines 110110000 to 110110011 for the value 01111101011101110000. If a match is found, return byte 111000 (56) of that line

How many bits are in the tag, set and offset fields?

2-way Set Associative

24 bit addresses

128K bytes of cache

16 byte cache lines

A. tag=8, set = 12, offset=4
B. tag=16, set = 4, offset=4
C. tag=12, set = 8, offset=4
D. tag=10, set = 10, offset=4

Replacement policy

- When a cache miss occurs, data is copied into some location in cache
- With Set Associative or Fully Associative mapping, the system must decide where to put the data and what values will be replaced
- Cache performance is greatly affected by properly choosing data that is unlikely to be referenced again

Replacement Options

- First In First Out (FIFO)
- Least Recently Used (LRU)
- Pseudo LRU
- Random

LRU replacement

- LRU is easy to implement for 2 way set associative
- You only need one bit per set to indicate which line in the set was most recently used
- LRU is difficult to implement for larger ways
- For an N way mapping, there are N! different permutations of use orders
- It would require $\log_2(N!)$ bits to keep track

Pseudo LRU

- Pseudo LRU is frequently used in set associative mapping
- In pseudo LRU there is a bit for each half of a group indicating which half was most recently used
- For 4 way set associative, one bit indicates that the upper two or lower two was most recently used. For each half another bit specifies which of the two lines was most recently used. 3 bits total

Comparison of Mapping Fully Associative

- Associative mapping works the best, but is complex to implement. Each tag line requires circuitry to compare the desired address with the tag field
- Some special purpose cache, such as the virtual memory Translation Lookaside Buffer (TLB) is a fully associative cache

Comparison of Mapping Direct

- Direct has the lowest performance, but is easiest to implement
- Direct is often used for instruction cache
- Sequential addresses fill a cache line and then go to the next cache line

Comparison of Mapping Set Associative

- Set associative is a compromise between the other two. The bigger the “way” the better the performance, but the more complex and expensive
- Intel Pentium uses 8 way set associative caching for level 1 data and 4, 8, 16 or 24 way set associative for level 2

Who Cares?

What good is it to know about cache memory?

Better grade on the next COMP375 exam

Squeezing the last nanosecond

- Programs that conform to locality of reference will have a higher probability of a cache hit
- Sequential access of RAM will be faster than the same number of random accesses

- With large cache memories, most of a reasonably sized program will be in cache

Address Bits

- It is a valuable Computer Science ability to consider a collection of bits as a number
- Hash systems often use the bits of the key similar to the mapping algorithms

Design Project

- The project is due by 1:00pm (start of class) on **Monday, October 19**, 2015
- Be sure to include the names of both team members

Exam on Friday

- The second exam in COMP375 will be on Wednesday, October 21, 2015
- It covers all the material since the first exam
  - Interrupts
  - Busses
  - Memory
  - VLSI
  - Cache
- You may have one 8½ by 11" page of notes