Digital Logic VIII: Caching

CMSC 313
Sections 01, 02

Direct Mapping

6.4 Cache Memory

- The purpose of cache memory is to speed up accesses by storing recently used data closer to the CPU, instead of storing it in main memory.
- Although cache is much smaller than main memory, its access time is a fraction of that of main memory.
- Unlike main memory, which is accessed by address, cache is typically accessed by content; hence, it is often called *content addressable memory*.
- Because of this, a single large cache memory isn't always desirable-- it takes longer to search.

6.4 Cache Memory

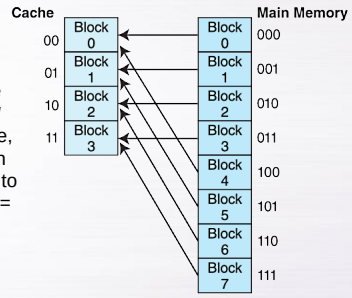
- The simplest cache mapping scheme is *direct mapped cache*.
- In a direct mapped cache consisting of *N* blocks of cache, block *X* of main memory maps to cache block *Y* = *X* mod *N*.
- Thus, if we have 10 blocks of cache, block 7 of cache may hold blocks 7, 17, 27, 37, . . . of main memory.

The next slide illustrates this mapping.

4

6.4 Cache Memory

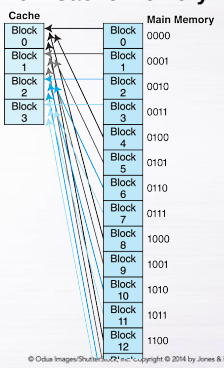
- With direct mapped cache consisting of *N* blocks of cache, block *X* of main memory maps to cache block *Y* = *X* mod *N*.



5

6.4 Cache Memory

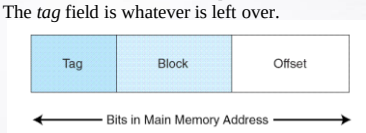
- A larger example.



6

6.4 Cache Memory

- To perform direct mapping, the binary main memory address is partitioned into the *fields* shown below.
  - The *offset* field uniquely identifies an address within a specific block.
  - The *block* field selects a unique block of cache.
  - The *tag* field is whatever is left over.

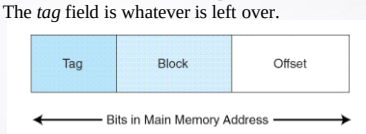


- The sizes of these fields are determined by characteristics of both memory and cache.

7

6.4 Cache Memory

- To perform direct mapping, the binary main memory address is partitioned into the *fields* shown below.
  - The *offset* field uniquely identifies an address within a specific block.
  - The *block* field selects a unique block of cache.
  - The *tag* field is whatever is left over.



- The sizes of these fields are determined by characteristics of both memory and cache.

8

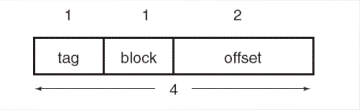
6.4 Cache Memory

- EXAMPLE 6.1 Consider a byte-addressable main memory consisting of 4 blocks, and a cache with 2 blocks, where each block is 4 bytes.
- This means Block 0 and 2 of main memory map to Block 0 of cache, and Blocks 1 and 3 of main memory map to Block 1 of cache.
- Using the tag, block, and offset fields, we can see how main memory maps to cache as follows.

9

6.4 Cache Memory

- EXAMPLE 6.1 Cont'd Consider a byte-addressable main memory consisting of 4 blocks, and a cache with 2 blocks, where each block is 4 bytes.
  - First, we need to determine the address format for mapping. Each block is 4 bytes, so the offset field must contain 2 bits; there are 2 blocks in cache, so the block field must contain 1 bit; this leaves 1 bit for the tag (as a main memory address has 4 bits because there are a total of 2^4=16 bytes).

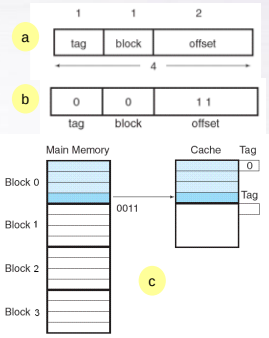


10

6.4 Cache Memory

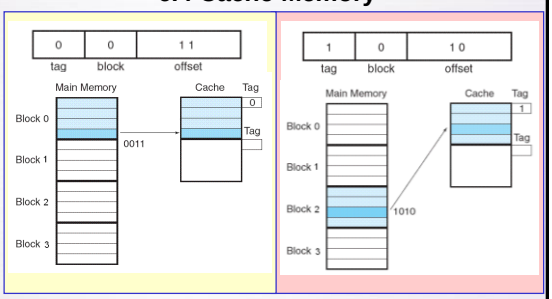
- EXAMPLE 6.1 Cont'd
  - Suppose we need to access main memory address 3_{16} (0x0011 in binary). If we partition 0x0011 using the address format from Figure a, we get Figure b.
  - Thus, the main memory address 0x0011 maps to cache block 0.
  - Figure c shows this mapping, along with the tag that is also stored with the data.

The next slide illustrates another mapping.



11

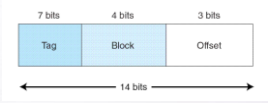
6.4 Cache Memory



12

6.4 Cache Memory

- EXAMPLE 6.2 Assume a byte-addressable memory consists of 2^{14} bytes, cache has 16 blocks, and each block has 8 bytes.
  - The number of memory blocks are: $\frac{2^{14}}{2^3} = 2^{11}$
  - Each main memory address requires14 bits. Of this 14-bit address field, the rightmost 3 bits reflect the offset field
  - We need 4 bits to select a specific block in cache, so the block field consists of the middle 4 bits.
  - The remaining 7 bits make up the tag field.



13

6.4 Cache Memory

- In summary, direct mapped cache maps main memory blocks in a modular fashion to cache blocks. The mapping depends on:
- The number of bits in the main memory address (how many addresses exist in main memory)
- The number of blocks are in cache (which determines the size of the block field)
- How many addresses (either bytes or words) are in a block (which determines the size of the offset field)

14

Fully Associative Mapping

15

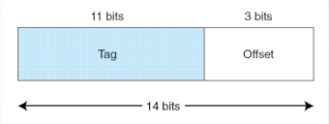
6.4 Cache Memory

- Suppose instead of placing memory blocks in specific cache locations based on memory address, we could allow a block to go anywhere in cache.
- In this way, cache would have to fill up before any blocks are evicted.
- This is how *fully associative* cache works.
- A memory address is partitioned into only two fields: the tag and the word.

16

6.4 Cache Memory

- Suppose, as before, we have 14-bit memory addresses and a cache with 16 blocks, each block of size 8. The field format of a memory reference is:



- When the cache is searched, all tags are searched in parallel to retrieve the data quickly.
- This requires special, costly hardware.

17

Set Associative Mapping

18

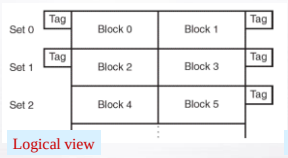
6.4 Cache Memory

- Set associative cache combines the ideas of direct mapped cache and fully associative cache.
- An *N*-way set associative cache mapping is like direct mapped cache in that a memory reference maps to a particular location in cache.
- Unlike direct mapped cache, a memory reference maps to a set of several cache blocks, similar to the way in which fully associative cache works.
- Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots.

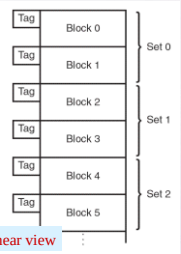
19

6.4 Cache Memory

- The number of cache blocks per set in set associative cache varies according to overall system design.
  - For example, a 2-way set associative cache can be conceptualized as shown in the schematic below.
  - Each set contains two different memory blocks.



Logical view



Linear view

20

6.4 Cache Memory

- In set associative cache mapping, a memory reference is divided into three fields: tag, set, and offset.
- As with direct-mapped cache, the offset field chooses the word within the cache block, and the tag field uniquely identifies the memory address.
- The set field determines the set to which the memory block maps.

21

6.4 Cache Memory

- EXAMPLE 6.5 Suppose we are using 2-way set associative mapping with a word-addressable main memory of 2^{14} words and a cache with 16 blocks, where each block contains 8 words.
  - Cache has a total of 16 blocks, and each set has 2 blocks, then there are 8 sets in cache.
  - Thus, the set field is 3 bits, the offset field is 3 bits, and the tag field is 8 bits.

22

Caching Policies

- Cache replacement policy
  - For fully associative and set associative mapping
  - Which cache block gets kicked out?
  - Some schemes: first-in first-out, least recently used, ...
- Cache write policy
  - Write through: always write to main memory
  - Write back: write to main memory when replaced

23

Cache Performance

24

6.4 Cache Memory

- The performance of hierarchical memory is measured by its *effective access time* (EAT).
- EAT is a weighted average that takes into account the hit ratio and relative access times of successive levels of memory.
- The EAT for a two-level memory is given by:

$$EAT = H \times Access_C + (1-H) \times Access_{MM}.$$

where H is the cache hit rate and $Access_C$ and $Access_{MM}$ are the access times for cache and main memory, respectively.

25

6.4 Cache Memory

- For example, consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.
- Suppose access to cache and main memory occurs concurrently. (The accesses overlap.)
- The EAT is:

0.99(10ns) + 0.01(200ns) = 9.9ns + 2ns = 11ns.

26

6.4 Cache Memory

- For example, consider a system with a main memory access time of 200ns supported by a cache having a 10ns access time and a hit rate of 99%.
- If the accesses do not overlap, the EAT is:

0.99(10ns) + 0.01(10ns + 200ns)
= 9.9ns + 2.01ns = 12ns.

- This equation for determining the effective access time can be extended to any number of memory levels, as we will see in later sections.

27