EECE 321: Computer Organization

Mohammad M. Mansour
Dept. of Electrical and Compute Engineering
American University of Beirut

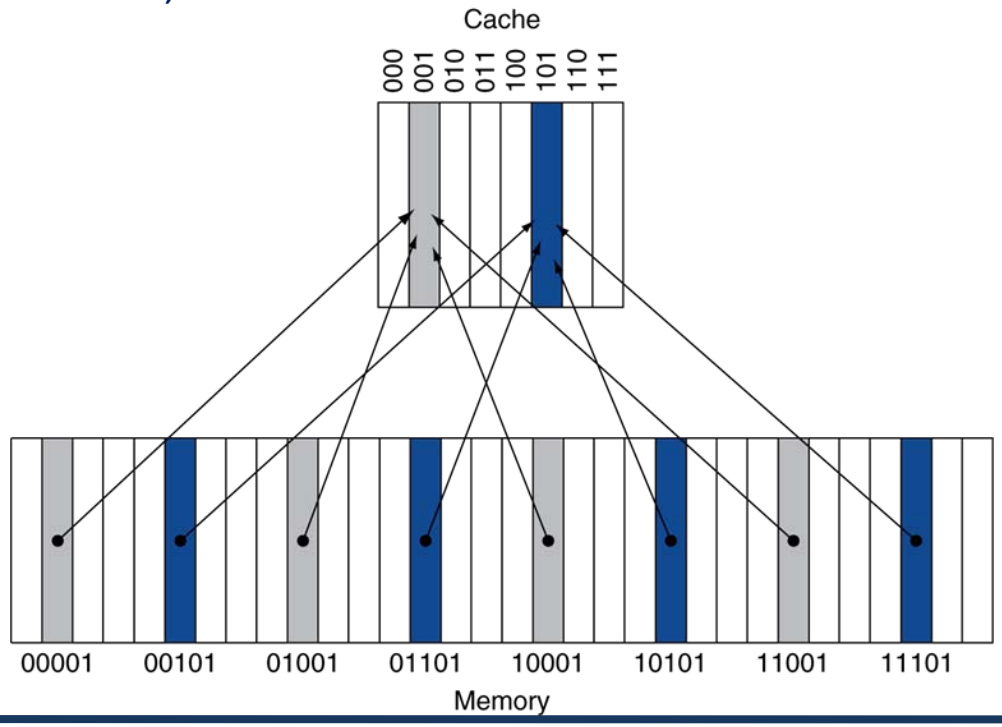
Lecture 30: Caches

Direct-Mapped Caches

- Since size of main memory is much larger than the size of cache, how do we map a block from main memory to cache?
- How do we know if a block is in cache? If it is, how do we find it?
- The simplest way to assign a location in cache for each block in memory is to use ***direct mapping***: (we'll examine other alternatives later)

(Block address) modulo (Number of blocks in cache)

- Example: 8-block cache, each block is 1 word

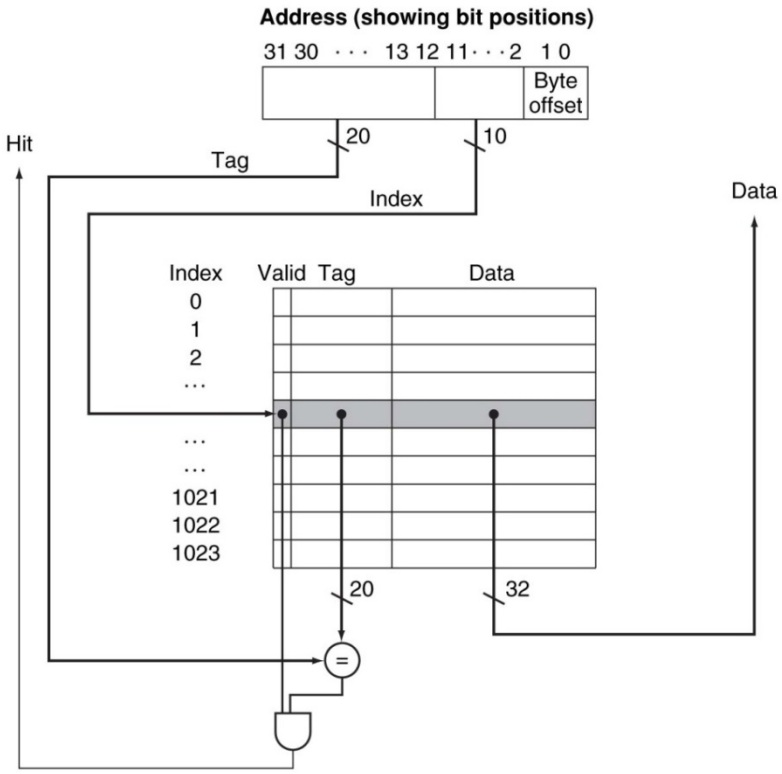


Direct-Mapped Caches

- How do we differentiate between the different blocks that map to the same cache location?
  - Add a set of tags to the cache.
  - A tag contains the address information required to identify whether a block in cache corresponds to the request block.
- In the previous example, least three bits are used to index cache. The upper most two bits are the tag bits.

- **How do we detect that a block entry in cache contains valid data?**
  - **Add a valid bit.**
- **Cache entries include**
  - **Data**
  - **Tag**
  - **Valid bit**

32-bit addresses
1K blocks
Block=word
Index: 10 bits
Tags: 20 Bits



Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example (cont'd)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	**Y**	**10**	**Mem[10110]**
111	N		

Cache Example (cont'd)

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	**Y**	**11**	**Mem[11010]**
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (cont'd)

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (cont'd)

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

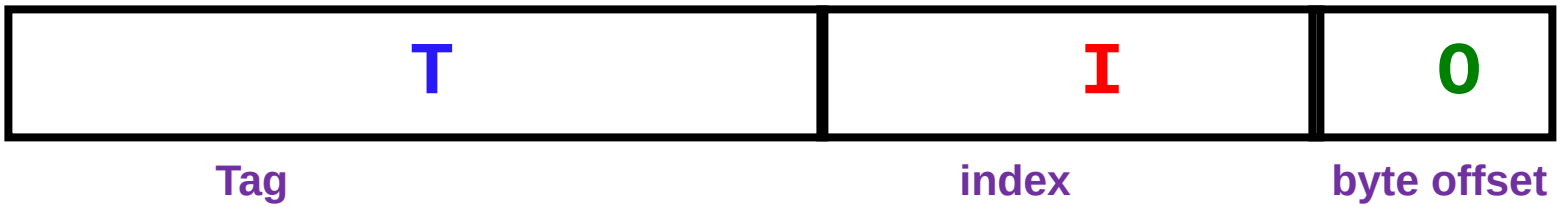
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Example (cont'd)

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache Size



- Cache includes storage for both data and tags, so the total number of bits needed for a cache is a function of the cache size (# blocks) and the address size.
- Assuming 32-bit byte addresses and a 32-bit processor, a direct-mapped cache of size 2^n words with one-word blocks.
  - An index field of size *n* because number of blocks is 2^n.
  - Size of byte offset field is 2
  - A tag field of size 32 - (*n*+2)
  - Therefore, cache size in bits= 2^n (32 + 32 - (n+2) +1)
- Example: How many bits are required for a direct-mapped cache with 64KB of data and one-word blocks, assuming 32-bit address?
- Answer = 2^{14}(32 + 32-14-2 + 1) bits
- What about 2-word blocks?
  - Answer: 2^{13}(32x2+ 32-13-3 +1) bits.

Determining Cache Size

- Assume byte addressable main memory with B-bit byte addresses
- Need to determine out of the B-bits, what the size of the offset field (O), index field (I), and tag field (T) are
- To get the size of cache in bits, we determine the size (in bits) of each entry in cache, and then multiply by the number of entries
- Divide main memory and cache in units called blocks. These blocks will hold data.
  - Each entry in cache will hold a block of data plus other book-keeping bits (tag, valid, …)
  - Assume the size of each block in bytes is 2^O. Then the size of the offset field is O.
- Next determine how many blocks there are in main memory (2^m) and in cache (2^c).
  - Index field: I = m – n.
- Size of tag field in bits is simply T = (B – I – O) = (B – (m – n) – O).
- The size of each entry in cache is: 8×2^O + T + 1 + (# other book-keeping bits).
- Total size of cache in bits is $2^c\times$ (8×2^O + T + 1 + …).

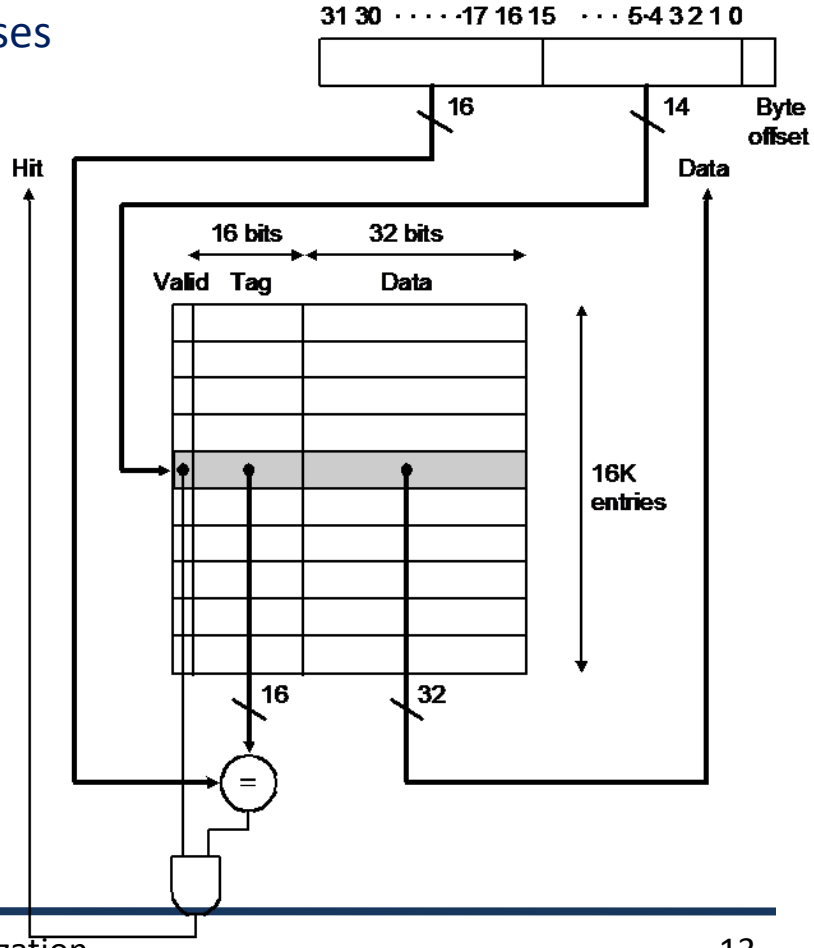
Handling Cache Misses

- The controller must detect cache misses and process these misses by fetching data/instructions from memory.
- For our pipelined MIPS processor, the basic approach is to stall the pipeline upon a miss.
  - Assume instructions are stored in I-cache, data words are stored in D-cache
- Steps taken on an instruction miss (steps for data misses are similar)
  - Send original PC to the main memory (one level below I-cache)
  - Instruct main memory to perform a read and wait for the memory to complete its access.
  - Write the cache entry, putting the data from memory in the data portion of the entry, writing the bits of the address (from PC) into the tag field, and turning the valid bit on.
  - Restart the instruction execution at the first step, which will re-fetch the instruction, this time finding it in I-cache.
- When dealing with caches, always need to separate between read and write accesses:
  - What happens on a read hit, read miss?
  - What happens on a write hit, write miss?
- Block placement and replacement:
  - A block is fetched into cache from memory according to a placement policy.
    - Example: Direct-mapping is one type of a block placement policy.
  - Blocks are returned from cache into memory to make room for other blocks according to a replacement policy.

An Example Cache: The DECStation 3100

- This machine uses a pipelined MIPS R2000 processor similar to the one we considered in class.
- Direct-Mapped cache organization:
- Two separate caches, I-Cache for instructions, D-Cache for Data
- Each is 64KB, one-word block, 32-bit addresses

- **Steps for a read request on both caches:**
  - **Send address to cache**
  - **If cache hits, requested word is retrieved. If cache misses, send address to main memory. When memory returns with data, write it into cache.**
- **What about writes?**
- **Suppose a store writes the data only into D-cache without changing main memory, then cache and main memory become inconsistent!**



Write-Through and Write-Back Caches

- The simplest way to keep main memory and caches consistent is to always write the data into both cache and main memory.
  - This scheme is called **write-through**.
- What happens on a write miss, i.e. when store wants to write data into cache that is not present in cache?
  - Since the data is going to be overwritten any way by the processor, there is no reason to read a word from memory.
  - For the simple cache organization on the DEC machine, we do not consider write misses. Write misses are relevant when a block contains more than one word (more on that later)
- Write-through caches negatively impact performance. Every write causes data to be written to main memory which takes a long time:
  - EX: In GCC, 13% of the instructions are stores. The CPI without cache misses for a program like GCC is 1.2. So spending for example 10 cycles on every write to memory would increase the CPI to 1.2 + 10 x 13% = 2.5.

Write-Through and Write-Back Caches

- One solution is to use a small write-buffer, which stores the data while it is waiting to be written to memory. After writing the data into the buffer and cache, the processor resumes execution.
- Another solution is to delay writing to main memory until the block in cache *is to be replaced* with a new block from memory. This scheme is called write-back.
  - Write-back caches improve performance, but are more complex.