CS429: Computer Organization and Architecture

Cache II

Warren Hunt, Jr. and Bill Young
Department of Computer Sciences
University of Texas at Austin

Last updated: August 26, 2014 at 08:54

Cache Vocabulary

- Capacity
- Cache block (aka cache line)
- Associativity
- Cache set
- Index
- Tag
- Hit rate
- Miss rate
- Placement policy
- Replacement policy

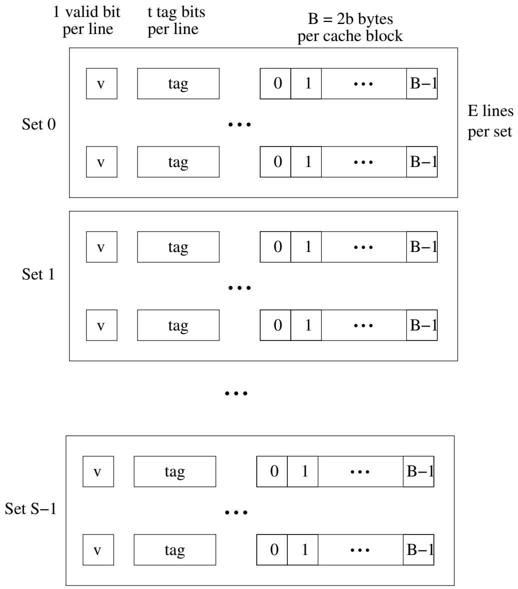
Organization of Cache Memory

Cache is an array of $S = 2^s$ sets.

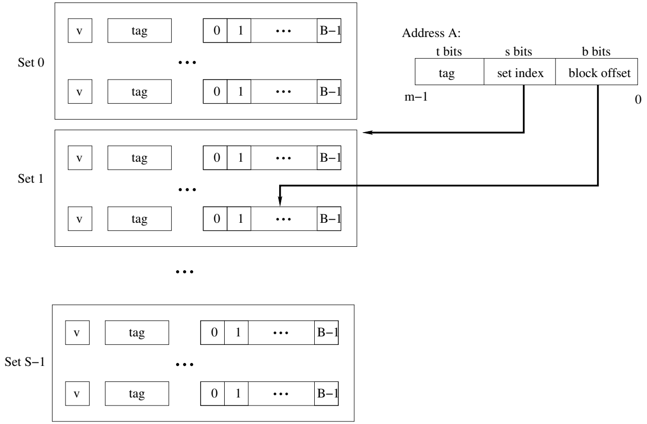
Each set contains $E \geq 1$ lines.

Each line holds a block of data containing $B = 2^b$ bytes

Cache size: $C = B \times E \times S$ data bytes.



Addressing Caches

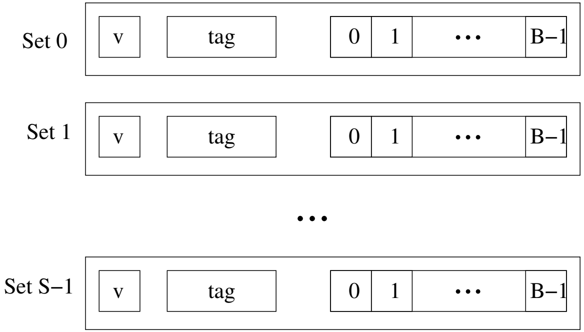


The word at address A is in the cache if the tag bits in one of the *valid* lines in set *set_index* match *tag* for that line.

The word contents begin at offset *block offset* from the beginning of the block.

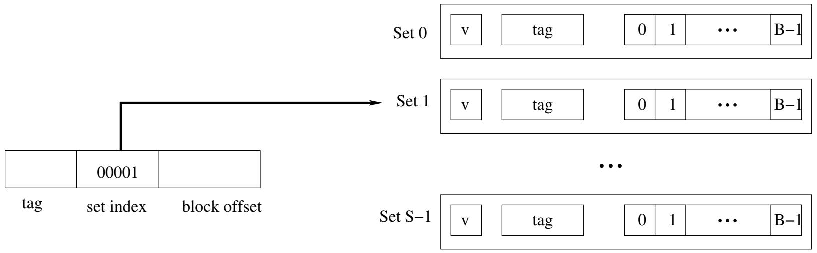
Direct-Mapped Cache

This is the simplest kind of cache, characterized by exactly one line per set (i.e, $E = 1$).



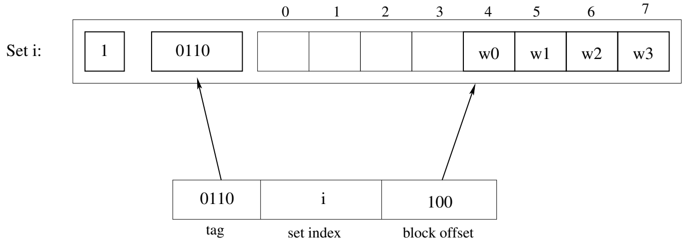
Accessing Direct-Mapped Caches

Use the set index bits to determine the set of interest.



Direct-Mapped Caches: Matching and Selection

- *Line matching:* Find a valid line in the selected set with a matching tag.
- *Word selection:* Extract the word using the block offset.



1. The valid bit must be set.
2. The tag bits in the cache line must match the tag bits in the address.
3. If (1) and (2), then cache hit, and block offset selects starting bits.

Direct-Mapped Cache Simulation

Suppose:

- $M = 16$ byte addresses;
- $B = 2$ bytes/block;
- $S = 4$ sets;
- $E = 1$ line/set.

Address trace (reads):

1. $0 = [0000_2]$
2. $1 = [0001_2]$
3. $13 = [1101_2]$
4. $8 = [1000_2]$
5. $0 = [0000_2]$

Address	x	xx	x

(1) 0 [0000] (miss)

v	tag	data
1	0	M[0–1]

(3) 13 [1101] (miss)

v	tag	data
1	0	M[0–1]
1	1	M[12–13]

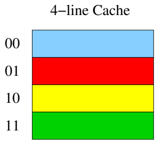
(4) 8 [1000] (miss)

v	tag	data
1	1	M[8–9]
1	1	M[12–13]

(5) 0 [0000] (miss)

v	tag	data
1	0	M[0–1]
1	1	M[12–13]

Why Use Middle Bits as Index?

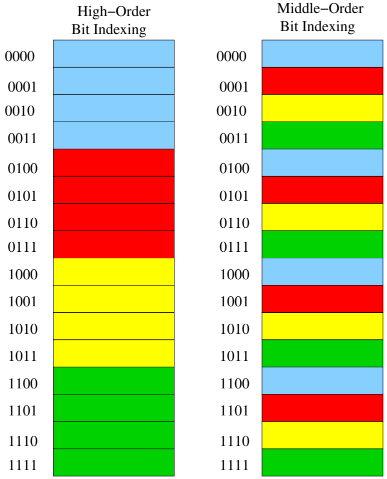


High-Order Bit Indexing

- Adjacent memory lines map to same cache entry.
- Poor use of spatial locality.

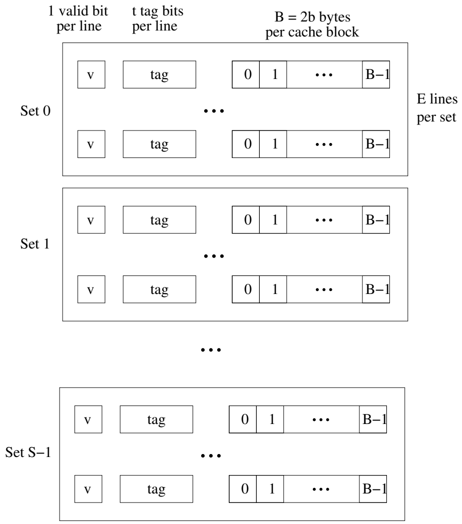
Middle-Order Bit Indexing

- Consecutive memory lines map to different cache lines.
- Can hold a C-byte region of address space in cache at one time.



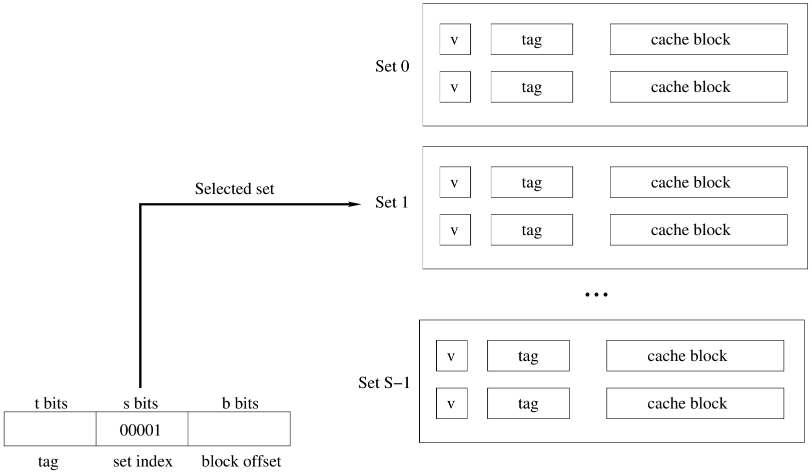
Set Associative Caches

These are characterized by more than one line per set.



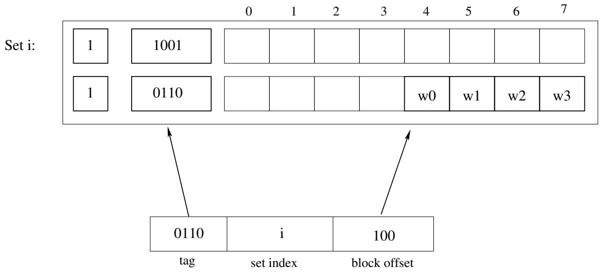
Accessing Set Associative Caches

Set selection is identical to that for direct-mapped cache.



Accessing Set Associative Caches

For *Line Matching* and *Word Selection*, we must compare the tag in each valid line in the selected set.



1. The valid bit must be set.
2. The tag bits in one of the cache lines must match the tag bits in the address.
3. If (1) and (2), than cache hit, and block offset selects starting byte.

Cache Performance Metrics

Miss Rate

- Fraction of memory references not found in the cache (misses / references)
- Typical numbers: 3-10% for L1; can be quite small (e.g., $< 1\%$) for L2, depending on size, etc.

Hit Time

- Time to deliver a line in the cache to the processor (including time to determine whether the line is in the cache).
- Typical numbers: 1-3 clock cycles for L1; 5-12 clock cycles for L2.

Miss Penalty

- Additional time required because of a miss.
- Typically 100-300 cycles for main memory.

Memory System Performance

Average Memory Access Time (AMAT)

$$T_{access} = (1 - p_{miss}) \cdot t_{hit} + p_{miss} \cdot t_{miss}$$
$$t_{miss} = t_{hit} + t_{penalty}$$

Assume 1-level cache, 90% hit rate, 1 cycle hit time, 200 cycle miss penalty.

AMAT = 21 cycles, even though 90% only take one cycle. This shows the importance of a high hit rate.

Memory System Performance II

How does AMAT affect overall performance? Recall the CPI equation (pipeline efficiency).

$$CPI = 1.0 + lp + mp + rp$$

- load/use penalty (lp) assumed memory access of 1 cycle.
- Further, we assumed all load instructions were 1 cycle.
- More realistic AMAT (20+ cycles) really hurts CPI and overall performance.

Cause	Name	Instr. Freq.	Cond. Freq.	Stalls	Product
Load	lp	0.30	0.7	21	4.41
Load/Use	lp	0.30	0.3	21+1	1.98
Mispredict	mp	0.20	0.4	2	0.16
Return	rp	0.02	1.0	3	0.06
Total					6.61

Memory System Performance III

$$T_{access} = (1 - p_{miss}) \cdot t_{hit} + p_{miss} \cdot t_{miss}$$
$$t_{miss} = t_{hit} + t_{penalty}$$

How can we reduce AMAT?

- Reduce the miss rate.
- Reduce the miss penalty.
- Reduce the hit time.

There have been numerous inventions targeting each of these.

Issues with Writes

If you *write* to an item in cache, the cached value becomes *inconsistent* with the values stored at lower levels of the memory hierarchy.

There are two main approaches to dealing with this:

Write-through:	immediately write the cache block to the next lowest level.
Write-back:	only write to lower levels when the block is evicted from the cache.

Write-through requires updating multiple levels of the memory hierarchy (causes bus traffic) on every write.

Write-back reduces bus traffic, but requires that each cache line have a *dirty bit*, indicating that the line has been modified.

Write Strategies

How to deal with write misses?

Write-allocate loads the line from the next level and updates the cache block.

No-write-allocate bypasses the cache and updates directly in the lower level of the memory hierarchy.

Write-through caches are typically no-write-allocate. Write-back caches are typically write-allocate.

Writing Cache Friendly Code

- Can write code to improve miss rate.
- Repeated references to variables are good (temporal locality).
- Stride-1 reference patterns are good (spatial locality).

Examples: Assume cold cache, 4-byte words, 4-word cache blocks.

```c
int sumarrayrows (int a[M][N])
{
    int i, j, sum = 0;
    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Miss rate $= 1/4 = 25\%$

```c
int sumarraycols (int a[M][N])
{
    int i, j, sum = 0;
    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Miss rate $= 100\%$

Some Questions to Consider

- What happens where there is a miss and the cache has no free lines? What should we evict?
- What happens on a store miss?
- What if we have a multicore chip where cores share the L2 cache but have private L1 caches? What bad things could happen?

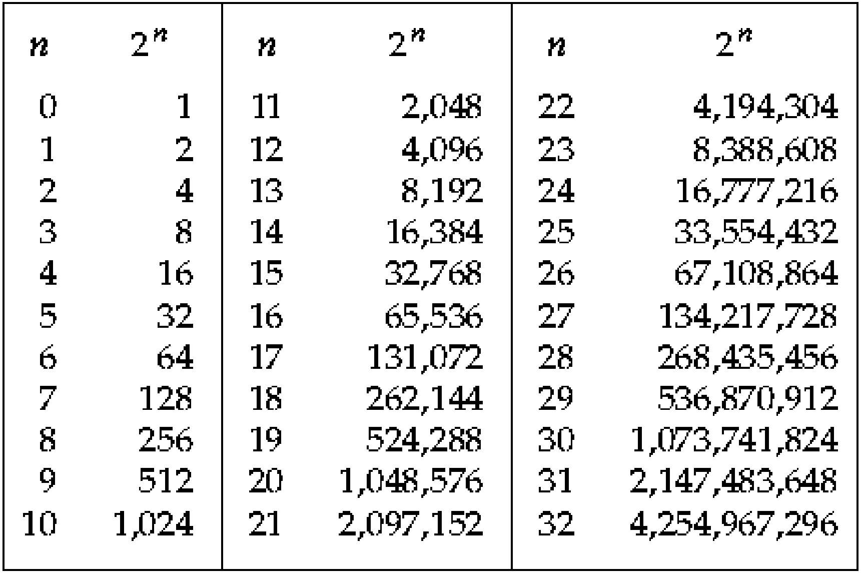
Concluding Observations

A programmer can optimize for cache performance.

- How data structures are organized.
- How data are accessed.
- Nested loop structure.
- Blocking is a general technique.

All systems favor "cache friendly code."

- Getting absolute optimum performance is very platform specific (cache sizes, line sizes, associativities, etc.)
- But you can get most of the advantage with generic code.
- Keep the working set reasonably small (temporal locality).
- Use small strides (spatial locality).



n	2^n	n	2^n	n	2^n
0	1	11	2,048	22	4,194,304
1	2	12	4,096	23	8,388,608
2	4	13	8,192	24	16,777,216
3	8	14	16,384	25	33,554,432
4	16	15	32,768	26	67,108,864
5	32	16	65,536	27	134,217,728
6	64	17	131,072	28	268,435,456
7	128	18	262,144	29	536,870,912
8	256	19	524,288	30	1,073,741,824
9	512	20	1,048,576	31	2,147,483,648
10	1,024	21	2,097,152	32	4,254,967,296