Example

```
int A[1024];
int B[1024];
int C[1024];

    for (int i = 0; i < 1024; i++) {
        A[i] = B[i] + C[i];
    }
```

- Where is the temporal locality (if any)?
- Where is the spatial locality (if any)?

Chapter 5

Large and Fast: Exploiting Memory Hierarchy

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

Principle of Locality(局部性)

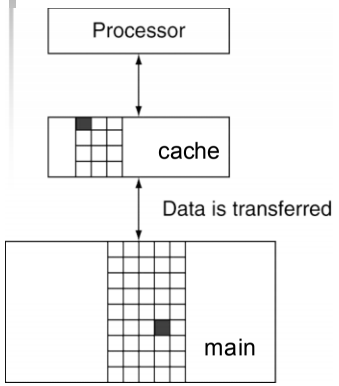
- Programs access a small proportion of their address space at any time
- Temporal locality(时间局部性)
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality(空间局部性)
  - Items near those accessed recently are likely to be accessed soon
  - E.g., sequential instruction access, array data

https://en.wikipedia.org/wiki/Induction_variable

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- How do we know if the data is present?
- Where do we look?

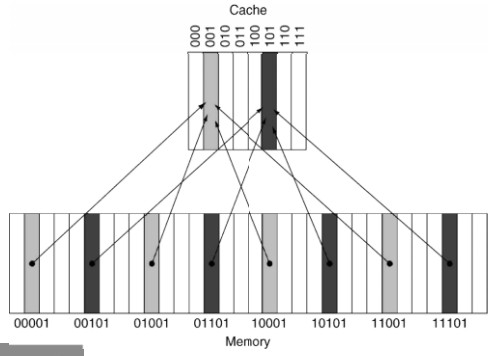
Memory Hierarchy Levels



- Block (aka line): unit of copying
  - May be multiple words **Why?**
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Extra time taken: miss penalty
    - Miss ratio: misses/accesses = 1 – hit ratio
  - Then accessed data supplied from upper level

Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
  - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits

Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, \$500 – \$1000 per GB

×100

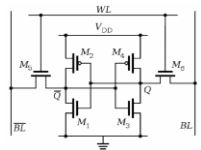
- Dynamic RAM (DRAM)
  - 50ns – 70ns, \$10 – \$20 per GB

×100

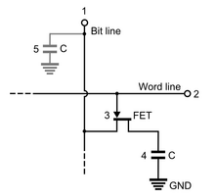
- Flash
  - 5μs – 50μs, \$0.75 – \$1 per GB

×1000

- Magnetic disk
  - 5ms – 20ms, \$0.05 – \$0.10 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk



SRAM



DRAM

Illusion is created by **cache** memory and **virtual** memory

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, only need the high-order bits
  - Called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- Main memory 32 words
- Cache 8-blocks, 1 word/block, direct mapped
- Initial state:

Index	V	Tag	Data
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	0		
111	0		

Cache Example

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

Index	V	Tag	Data
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	0		
111	0		

Cache Example

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

Index	V	Tag	Data
000	0		
001	0		
010	0		
011	0		
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

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

Index	V	Tag	Data
000	1	10	Mem[10000]
001	0		
010	1	11	Mem[11010]
011	1	00	Mem[00011]
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

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

Index	V	Tag	Data
000	0		
001	0		
010	1	11	Mem[11010]
011	0		
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

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

Index	V	Tag	Data
000	1	10	Mem[10000]
001	0		
010	1	**10**	**Mem[10010]**
011	1	00	Mem[00011]
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Cache Example

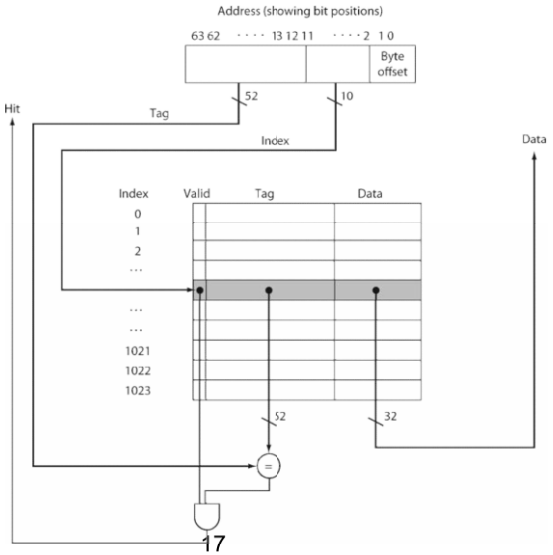
Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	**1**	**10**	**Mem[10000]**
001	0		
010	1	11	Mem[11010]
011	**1**	**00**	**Mem[00011]**
100	0		
101	0		
110	1	10	Mem[10110]
111	0		

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
  - Larger blocks ⇒ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help (see page 406)

Address Subdivision



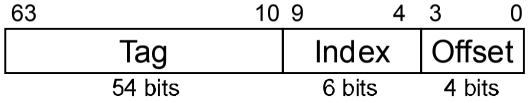
Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Example: Larger Block Size

Why?

- 64 blocks, 16 bytes/block
  - To what block number (index) does address 1200 map?
- Block address = ⌊1200/16⌋ = 75
- Block number = 75 modulo 64 = 11



Alternative 1200 = 0..0100 1011 0000

Tag = 1 Index = 11 Offset = 0

Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don't fetch the block
    - Since programs often write a large block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full

Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

$$\text{Memory stall cycles} = \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit time × Hit rate
    + (Hit time + Miss penalty) × Miss rate
  - AMAT = Hit time × (1 – Miss rate)
    + (Hit time + Miss penalty) × Miss rate
  - AMAT = Hit time – Hit time × Miss rate
    + Hit time × Miss rate + Miss penalty × Miss rate
  - AMAT = Hit time + Miss rate × Miss penalty

See book page 416

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles Realistic value, see sheet 8
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions
- Calculate speedup due to addition of cache
- Calculate maximum speedup (if main memory would be as fast as cache)

Performance Summary

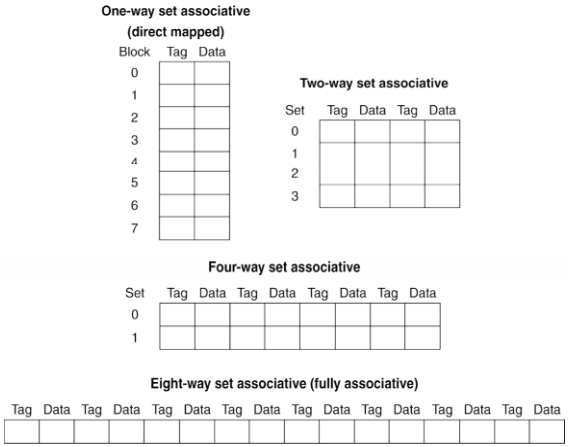
- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance

Cache Performance Example

- CPI without cache
  - = 2 + 100 + 0.36 × 100 = 138
- CPI with cache
  - = 2 + 0.02 × 100 + 0.36 × 0.04 × 100 = 5.44
- Speedup
  - = 138 / 5.44 = 25.4
- Maximum speedup
  - = 138 / 2 = 69
  - = 69 / 25.4 = 2.7 faster than with cache

Spectrum of Associativity

- For a cache with 8 entries



Associative Caches

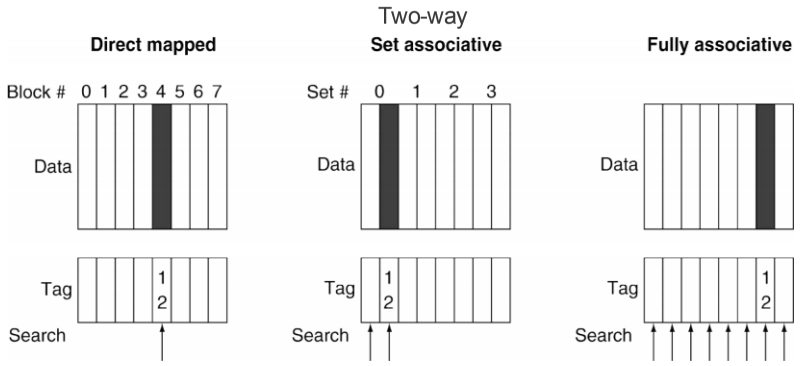
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			

Associative Cache Example



How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Associativity Example

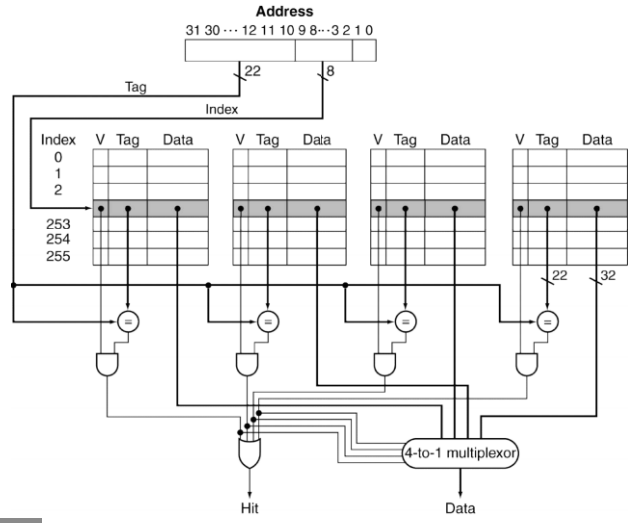
- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			

- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			

Set Associative Cache Organization



20220418

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Calculate speedup
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Speedup = 9/3.4 = 2.6

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache

Software Optimization via Blocking

- Goal: maximize accesses to data before it is replaced
- See book page 427

Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Concluding Remarks

- Fast memories are expensive and therefore small, cheap memories are slow but can be large
  - We really want fast, large memories ☹
  - Caching gives this illusion ☺
- Principle of locality
  - Programs use a small part of their memory space frequently
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk

Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation