Memory Hierarchy: The motivation

- The gap between CPU performance and main memory has been widening with higher performance CPUs creating performance bottlenecks for memory access instructions.

- The memory hierarchy is organized into several levels of memory with the smaller, more expensive, and faster memory levels closer to the CPU: registers, then primary Cache Level ($L_1$), then additional secondary cache levels ($L_2$, $L_3$…), then main memory, then mass storage (virtual memory).

- Each level of the hierarchy is a subset of the level below: data found in a level is also found in the level below but at lower speed.

- Each level maps addresses from a larger physical memory to a smaller level of physical memory.

- This concept is greatly aided by the principal of locality both temporal and spatial which indicates that programs tend to reuse data and instructions that they have used recently or those stored in their vicinity leading to working set of a program.
## From Recent Technology Trends

<table>
<thead>
<tr>
<th></th>
<th>Capacity</th>
<th>Speed (latency)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic:</td>
<td>2x in 3 years</td>
<td>2x in 3 years</td>
</tr>
<tr>
<td>DRAM:</td>
<td>4x in 3 years</td>
<td>2x in 10 years</td>
</tr>
<tr>
<td>Disk:</td>
<td>4x in 3 years</td>
<td>2x in 10 years</td>
</tr>
</tbody>
</table>

### DRAM

<table>
<thead>
<tr>
<th>Year</th>
<th>Size</th>
<th>Cycle Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1980</td>
<td>64 Kb</td>
<td>250 ns</td>
</tr>
<tr>
<td>1983</td>
<td>256 Kb</td>
<td>220 ns</td>
</tr>
<tr>
<td>1986</td>
<td>1 Mb</td>
<td>190 ns</td>
</tr>
<tr>
<td>1989</td>
<td>4 Mb</td>
<td>165 ns</td>
</tr>
<tr>
<td>1992</td>
<td>16 Mb</td>
<td>145 ns</td>
</tr>
<tr>
<td>1995</td>
<td>64 Mb</td>
<td>120 ns</td>
</tr>
</tbody>
</table>

1000:1 2:1
Memory Hierarchy: Motivation
Processor-Memory (DRAM) Performance Gap

- Processor-Memory Performance Gap: (grows 50% / year)
- DRAM 7%/yr.
- μProc 60%/yr.
Processor-DRAM Performance Gap Impact: Example

- To illustrate the performance impact, assume a pipelined RISC CPU with CPI = 1 using non-ideal memory.
- Over an 10 year period, ignoring other factors, the cost of a full memory access in terms of number of wasted instructions:

<table>
<thead>
<tr>
<th>Year</th>
<th>CPU speed MHZ</th>
<th>CPU cycle ns</th>
<th>Memory Access ns</th>
<th>Minimum CPU cycles or instructions wasted</th>
</tr>
</thead>
<tbody>
<tr>
<td>1986:</td>
<td>8</td>
<td>125</td>
<td>190</td>
<td>190/125 = 1.5</td>
</tr>
<tr>
<td>1988:</td>
<td>33</td>
<td>30</td>
<td>175</td>
<td>175/30 = 5.8</td>
</tr>
<tr>
<td>1991:</td>
<td>75</td>
<td>13.3</td>
<td>155</td>
<td>155/13.3 = 11.65</td>
</tr>
<tr>
<td>1994:</td>
<td>200</td>
<td>5</td>
<td>130</td>
<td>130/5 = 26</td>
</tr>
<tr>
<td>1996:</td>
<td>300</td>
<td>3.33</td>
<td>100</td>
<td>110/3.33 = 33</td>
</tr>
</tbody>
</table>
Memory Hierarchy: Motivation

The Principle Of Locality

• Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time (program working set).

• Two Types of locality:
  – Temporal Locality: If an item is referenced, it will tend to be referenced again soon.
  – Spatial locality: If an item is referenced, items whose addresses are close will tend to be referenced soon.

• The presence of locality in program behavior, makes it possible to satisfy a large percentage of program access needs (both instructions and operands) using memory levels with much less capacity than program address space.
Levels of The Memory Hierarchy

Part of The On-chip CPU Datapath
16-256 Registers

One or more levels (Static RAM):
Level 1: On-chip 16-64K
Level 2: On or Off-chip 128-512K
Level 3: Off-chip 128K-8M

Registers

Cache

Main Memory

Magnetic Disc

Optical Disk or Magnetic Tape

Farther away from The CPU
Lower Cost/Bit
Higher Capacity
 Increased Access Time/Latency
Lower Throughput

DRAM, RDRAM
16M-16G

Interface:
SCSI, RAID,
IDE, 1394
4G-100G
A Typical Memory Hierarchy
(With Two Levels of Cache)

Faster
Larger Capacity

Virtual Memory, Secondary Storage (Disk)

Main Memory (DRAM)

Second Level Cache (SRAM) L₂

On-Chip Level One Cache L₁

Datapath

Registers

Processor

Control

Tertiary Storage (Tape)

Speed (ns): 1s 10s 100s 10,000,000s (10s ms) 10,000,000,000s (10s sec)

Size (bytes): 100s Ks Ms Gs Ts
# Levels of The Memory Hierarchy

<table>
<thead>
<tr>
<th>Level</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Called</td>
<td>Registers</td>
<td>Cache</td>
<td>Main memory</td>
<td>Disk storage</td>
</tr>
<tr>
<td>Typical size</td>
<td>&lt; 1 KB</td>
<td>&lt; 4 MB</td>
<td>&lt; 4 GB</td>
<td>&gt; 1 GB</td>
</tr>
<tr>
<td>Implementation tech.</td>
<td>Custom memory with</td>
<td>On-chip or off-</td>
<td>CMOS DRAM</td>
<td>Magnetic disk</td>
</tr>
<tr>
<td></td>
<td>multiple ports,</td>
<td>chip CMOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CMOS or BiCMOS</td>
<td>SRAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Access time (in ns)</td>
<td>2–5</td>
<td>3–10</td>
<td>80–400</td>
<td>5,000,000</td>
</tr>
<tr>
<td>Bandwidth (in MB/sec)</td>
<td>4000–32,000</td>
<td>800–5000</td>
<td>400–2000</td>
<td>4–32</td>
</tr>
<tr>
<td>Managed by</td>
<td>Compiler</td>
<td>Hardware</td>
<td>Operating system</td>
<td>Operating system/user</td>
</tr>
<tr>
<td>Backed by</td>
<td>Cache</td>
<td>Main memory</td>
<td>Disk</td>
<td>Tape</td>
</tr>
</tbody>
</table>
SRAM Organization Example
4 words X 3 bits each
Memory Hierarchy Operation

• If an instruction or operand is required by the CPU, the levels of the memory hierarchy are searched for the item starting with the level closest to the CPU (Level 1 cache):
  – If the item is found, it’s delivered to the CPU resulting in a cache hit without searching lower levels.
  – If the item is missing from an upper level, resulting in a miss, the level just below is searched.
  – For systems with several levels of cache, the search continues with cache level 2, 3 etc.
  – If all levels of cache report a miss then main memory is accessed for the item.
    • CPU ↔ cache ↔ memory: Managed by hardware.
    – If the item is not found in main memory resulting in a page fault, then disk (virtual memory), is accessed for the item.
    • Memory ↔ disk: Managed by hardware and the operating system.
Memory Hierarchy: Terminology

- **A Block**: The smallest unit of information transferred between two levels.
- **Hit**: Item is found in some block in the upper level (example: Block X)
  - **Hit Rate**: The fraction of memory access found in the upper level.
  - **Hit Time**: Time to access the upper level which consists of RAM access time + Time to determine hit/miss
- **Miss**: Item needs to be retrieved from a block in the lower level (Block Y)
  - **Miss Rate** = 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in the upper level + Time to deliver the block the processor
- **Hit Time << Miss Penalty**

![Memory Hierarchy Diagram]

To Processor

From Processor

Upper Level Memory

Blk X

Lower Level Memory

Blk Y
Cache Concepts

• Cache is the first level of the memory hierarchy once the address leaves the CPU and is searched first for the requested data.

• If the data requested by the CPU is present in the cache, it is retrieved from cache and the data access is a cache hit otherwise a cache miss and data must be read from main memory.

• On a cache miss a block of data must be brought in from main memory to cache to possibly replace an existing cache block.

• The allowed block addresses where blocks can be mapped into cache from main memory is determined by cache placement strategy.

• Locating a block of data in cache is handled by cache block identification mechanism.

• On a cache miss the cache block being removed is handled by the block replacement strategy in place.

• When a write to cache is requested, a number of main memory update strategies exist as part of the cache write policy.
Cache Design & Operation Issues

• Q1: Where can a block be placed cache?
  *(Block placement strategy & Cache organization)*
  – Fully Associative, Set Associative, Direct Mapped.

• Q2: How is a block found if it is in cache?
  *(Block identification)*
  – Tag/Block.

• Q3: Which block should be replaced on a miss?
  *(Block replacement)*
  – Random, LRU.

• Q4: What happens on a write?
  *(Cache write policy)*
  – Write through, write back.
We will examine:

- Cache Placement Strategies
  - Cache Organization
- Locating A Data Block in Cache
- Cache Replacement Policy
- What happens on cache Reads/Writes
- Cache write strategies
- Cache write miss policies
- Cache performance
Cache Organization & Placement Strategies

Placement strategies or mapping of a main memory data block onto cache block frame addresses divide cache into three organizations:

1. **Direct mapped cache**: A block can be placed in one location only, given by:
   
   $$(\text{Block address}) \mod (\text{Number of blocks in cache})$$

2. **Fully associative cache**: A block can be placed anywhere in cache.

3. **Set associative cache**: A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:
   
   $$(\text{Block address}) \mod (\text{Number of sets in cache})$$

   If there are $n$ blocks in a set the cache placement is called $n$-way set-associative.
Cache Organization: Direct Mapped Cache

A block can be placed in one location only, given by:
(Block address) \( \text{MOD} \) (Number of blocks in cache)

In this case:  (Block address) \( \text{MOD} \) (8)
Direct-Mapped Cache Example

1024 Blocks
Each block = one word
Can cache up to $2^{32}$ bytes of memory
Direct Mapped Cache Example

4K blocks
Each block = four words

Takes better advantage of spatial locality (possibly lower miss rate)
Alpha AXP 21064 Data Cache Organization

Direct Mapped Organization
Cache Organization:
Set Associative Cache

One-way set associative
(direct mapped)

Four-way set associative

Eight-way set associative (fully associative)
Cache Organization Example

Fully associative: block 12 can go anywhere

Direct mapped: block 12 can go only into block 4 (12 mod 8)

Set associative: block 12 can go anywhere in set 0 (12 mod 4)

Block frame address

Block no. 0 1 2 3 4 5 6 7

Memory

Set 0 1 2 3

FIGURE 5.2 This example cache has eight block frames and memory has 32 blocks.
Locating A Data Block in Cache

- Each block frame in cache has an address tag.
- The tags of every cache block that might contain the required data are checked in parallel.
- A valid bit is added to the tag to indicate whether this entry contains a valid address.
- The address from the CPU to cache is divided into:
  - A block address, further divided into:
    - An index field to choose a block set in cache. (no index field when fully associative).
    - A tag field to search and match addresses in the selected set.
  - A block offset to select the data from the block.
Address Field Sizes

Block Address

Tag

Index

Block Offset

Block offset size = log2(block size)

Index size = log2(Total number of blocks/associativity)

Tag size = address size - index size - offset size
Four-Way Set Associative Cache: DLX Implementation Example

256 sets
1024 block frames
FIGURE 5.8 A two-way set-associative version of the 8-KB cache of Figure 5.5, showing the extra multiplexer in the path.
Cache Organization/Addressing Example

- Given the following:
  - A single-level cache with 128 cache block frames
    - Each block frame contains four words (16 bytes)
  - 16-bit memory addresses (64K bytes main memory or 4096 blocks)

- Show the cache organization/mapping and cache address fields for:
  - Fully Associative cache
  - Direct mapped cache
  - 2-way set-associative cache
Cache Example: Fully Associative Case

- Block offset = 4 bits
- Block Address = 12 bits
- Tag = 12 bits

All 128 tags must be checked in parallel by hardware to locate a data block.
Cache Example: Direct Mapped Case

Block offset = 4 bits

Block Address = 12 bits
Tag = 5 bits
Index = 7 bits

Main Memory

| BLOCK 0 | BLOCK 1 | BLOCK 2 |
| BLOCK 127 | BLOCK 128 | BLOCK 129 |
| BLOCK 255 | BLOCK 256 | BLOCK 257 |
| BLOCK 4095 |

Only a single tag must be checked in parallel to locate a data block.
Cache Example: 2-Way Set-Associative

Block offset = 4 bits
Tag = 6 bits
Index = 6 bits
Block Address = 12 bits

Two tags in a set must be checked in parallel to locate a data block

Valid bits not shown
Cache Replacement Policy

- When a cache miss occurs the cache controller may have to select a block of cache data to be removed from a cache block frame and replaced with the requested data, such a block is selected by one of two methods:
  - Random:
    - Any block is randomly selected for replacement providing uniform allocation.
    - Simple to build in hardware.
    - The most widely used cache replacement strategy.
  - Least-recently used (LRU):
    - Accesses to blocks are recorded and the block replaced is the one that was not used for the longest period of time.
    - LRU is expensive to implement, as the number of blocks to be tracked increases, and is usually approximated.
## Miss Rates for Caches with Different Size, Associativity & Replacement Algorithm

### Sample Data

<table>
<thead>
<tr>
<th>Associativity:</th>
<th>2-way</th>
<th>4-way</th>
<th>8-way</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LRU</td>
<td>Random</td>
<td>LRU</td>
</tr>
<tr>
<td>Size</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 KB</td>
<td>5.18%</td>
<td>5.69%</td>
<td>4.67%</td>
</tr>
<tr>
<td>64 KB</td>
<td>1.88%</td>
<td>2.01%</td>
<td>1.54%</td>
</tr>
<tr>
<td>256 KB</td>
<td>1.15%</td>
<td>1.17%</td>
<td>1.13%</td>
</tr>
</tbody>
</table>
Unified vs. Separate Level 1 Cache

• Unified Level 1 Cache (Princeton Memory Architecture).
  A single level 1 cache is used for both instructions and data.

• Separate instruction/data Level 1 caches (Harvard Memory Architecture):
  The level 1 ($L_1$) cache is split into two caches, one for instructions (instruction cache, $L_1$ I-cache) and the other for data (data cache, $L_1$ D-cache).
Cache Performance:

Average Memory Access Time (AMAT), Memory Stall cycles

- The Average Memory Access Time (AMAT): The number of cycles required to complete an average memory access request by the CPU.
- Memory stall cycles per memory access: The number of stall cycles added to CPU execution cycles for one memory access.
- For ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.
- Memory stall cycles per average memory access = (AMAT - 1)
- Memory stall cycles per average instruction =
  Memory stall cycles per average memory access
  \( \times \) Number of memory accesses per instruction
  \( = (AMAT - 1) \times (1 + \text{fraction of loads/stores}) \)

Instruction Fetch
Cache Performance

Princeton (Unified) Memory Architecture

For a CPU with a single level (L1) of cache for both instructions and data (Princeton memory architecture) and no stalls for cache hits:

With ideal memory

Total CPU time = (CPU execution clock cycles +
Memory stall clock cycles) x clock cycle time

Memory stall clock cycles =
(Reads x Read miss rate x Read miss penalty) +
(Writes x Write miss rate x Write miss penalty)

If write and read miss penalties are the same:
Memory stall clock cycles =
Memory accesses x Miss rate x Miss penalty
Cache Performance

Princeton (Unified) Memory Architecture

CPU time = Instruction count × CPI × Clock cycle time

CPI_{execution} = CPI with ideal memory

CPI = CPI_{execution} + Mem Stall cycles per instruction

CPU time = Instruction Count × (CPI_{execution} +
Mem Stall cycles per instruction) × Clock cycle time

Mem Stall cycles per instruction =
Mem accesses per instruction × Miss rate × Miss penalty

CPU time = IC × (CPI_{execution} + Mem accesses per instruction ×
Miss rate × Miss penalty) × Clock cycle time

Misses per instruction = Memory accesses per instruction × Miss rate

CPU time = IC × (CPI_{execution} + Misses per instruction × Miss penalty) ×
Clock cycle time
Memory Access Tree
For Unified Level 1 Cache

CPU Memory Access

L1 Hit:
\[ \text{L1 Hit:} \quad \% = \text{Hit Rate} = H1 \]
Access Time = 1
Stalls = H1 x 0 = 0
(No Stall)

L1 Miss:
\[ \% = (1\text{- Hit rate}) = (1-H1) \]
Access time = M + 1
Stall cycles per access = M x (1-H1)

AMAT = H1 x 1 + (1-H1) x (M+ 1) = 1 + M x (1-H1)

Stall Cycles Per Access = AMAT - 1 = M x (1 -H1)
Cache Impact On Performance: 

Assuming the following execution and cache parameters:

- Cache miss penalty = 50 cycles
- Normal instruction execution CPI ignoring memory stalls = 2.0 cycles
- Miss rate = 2%
- Average memory references/instruction = 1.33

\[
\text{CPU time} = IC \times [\text{CPI}_{\text{execution}} + \text{Memory accesses/instruction} \times \text{Miss rate} \times \text{Miss penalty}] \times \text{Clock cycle time}
\]

\[
\text{CPUtime}_{\text{with cache}} = IC \times (2.0 + (1.33 \times 2\% \times 50)) \times \text{clock cycle time}
\]

\[
= IC \times 3.33 \times \text{Clock cycle time}
\]

\rightarrow \text{Lower CPI}_{\text{execution}} \text{ increases the impact of cache miss clock cycles}

\rightarrow \text{CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI}
Cache Performance Example

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.
- \( \text{CPI}_{\text{execution}} = 1.1 \)
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{mem stalls per instruction}
\]

\[
\text{Mem Stalls per instruction} = \text{Mem accesses per instruction} \times \text{Miss rate} \times \text{Miss penalty}
\]

\[
\text{Mem accesses per instruction} = 1 + 0.3 = 1.3
\]

\[
\text{Mem Stalls per instruction} = 1.3 \times 0.015 \times 50 = 0.975
\]

\[
\text{CPI} = 1.1 + 0.975 = 2.075
\]

The ideal memory CPU with no misses is \( 2.075/1.1 = 1.88 \) times faster
Cache Performance
Harvard Memory Architecture

For a CPU with separate or split level one (L1) caches for instructions and data (Harvard memory architecture) and no stalls for cache hits:

CPU\text{time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}

\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}

CPU\text{time} = \text{Instruction Count} \times \left( \text{CPI}_{\text{execution}} + \text{Mem Stall \ cycles per instruction} \right) \times \text{Clock cycle time}

\text{Mem Stall \ cycles per instruction} = \text{Instruction Fetch Miss rate} \times \text{Miss Penalty} + \text{Data Memory Accesses Per Instruction} \times \text{Data Miss Rate} \times \text{Miss Penalty}
Memory Access Tree
For Separate Level 1 Caches

CPU Memory Access

Instruction

Data

L_1

Instruction L1 Hit:
Access Time = 1
Stalls = 0

Instruction L1 Miss:
Access Time = M + 1
Stalls Per access

% instructions \times (1 - Instruction H1) \times M

Data L1 Hit:
Access Time: 1
Stalls = 0

Data L1 Miss:
Access Time: M + 1
Stalls per access:
% data \times (1 - Data H1) \times M

\text{Stall Cycles Per Access}\ = \% \text{ Instructions} \ \times (1 - \text{Instruction H1}) \ \times M \ + \% \text{ data} \ \times (1 - \text{Data H1}) \ \times M

\text{AMAT} = 1 + \text{Stall Cycles per access}
Typical Cache Performance Data
Using SPEC92

<table>
<thead>
<tr>
<th>Size</th>
<th>Instruction cache</th>
<th>Data cache</th>
<th>Unified cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 KB</td>
<td>3.06%</td>
<td>24.61%</td>
<td>13.34%</td>
</tr>
<tr>
<td>2 KB</td>
<td>2.26%</td>
<td>20.57%</td>
<td>9.78%</td>
</tr>
<tr>
<td>4 KB</td>
<td>1.78%</td>
<td>15.94%</td>
<td>7.24%</td>
</tr>
<tr>
<td>8 KB</td>
<td>1.10%</td>
<td>10.19%</td>
<td>4.57%</td>
</tr>
<tr>
<td>16 KB</td>
<td>0.64%</td>
<td>6.47%</td>
<td>2.87%</td>
</tr>
<tr>
<td>32 KB</td>
<td>0.39%</td>
<td>4.82%</td>
<td>1.99%</td>
</tr>
<tr>
<td>64 KB</td>
<td>0.15%</td>
<td>3.77%</td>
<td>1.35%</td>
</tr>
<tr>
<td>128 KB</td>
<td>0.02%</td>
<td>2.88%</td>
<td>0.95%</td>
</tr>
</tbody>
</table>

FIGURE 5.7  Miss rates for instruction, data, and unified caches of different sizes.
Cache Performance Example

To compare the performance of either using a 16-KB instruction cache and a 16-KB data cache as opposed to using a unified 32-KB cache, we assume a hit to take one clock cycle and a miss to take 50 clock cycles, and a load or store to take one extra clock cycle on a unified cache, and that 75% of memory accesses are instruction references. Using the miss rates for SPEC92 we get:

Overall miss rate for a split cache = \((75\% \times 0.64\%) + (25\% \times 6.74\%)\) = 2.1%

From SPEC92 data a unified cache would have a miss rate of 1.99%

Average memory access time = 1 + stall cycles per access

= 1 + \% instructions \times (Instruction miss rate \times Miss penalty) + \% data \times (Data miss rate \times Miss penalty)

For split cache:

Average memory access time_{split} = 1 + 75\% \times (0.64\% \times 50) + 25\% \times (6.47\% \times 50) = 2.05

For unified cache:

Average memory access time_{unified} = 1 + 75\% \times (1.99\% \times 50) + 25\% \times (1 + 1.99\% \times 50)

= 2.24 cycles
Cache Read/Write Operations

- Statistical data suggest that reads (including instruction fetches) dominate processor cache accesses (writes account for 25% of data cache traffic).

- In cache reads, a block is read at the same time while the tag is being compared with the block address. If the read is a hit the data is passed to the CPU, if a miss it ignores it.

- In cache writes, modifying the block cannot begin until the tag is checked to see if the address is a hit.

- Thus for cache writes, tag checking cannot take place in parallel, and only the specific data (between 1 and 8 bytes) requested by the CPU can be modified.

- Cache is classified according to the write and memory update strategy in place: write through, or write back.
Cache Write Strategies

1 Write Though: Data is written to both the cache block and to a block of main memory.
   - The lower level always has the most updated data; an important feature for I/O and multiprocessing.
   - Easier to implement than write back.
   - A write buffer is often used to reduce CPU write stall while data is written to memory.

2 Write back: Data is written or updated only to the cache block. The modified or dirty cache block is written to main memory when it’s being replaced from cache.
   - Writes occur at the speed of cache
   - A status bit called a dirty bit, is used to indicate whether the block was modified while in cache; if not the block is not written to main memory.
   - Uses less memory bandwidth than write through.
Cache Write Miss Policy

• Since data is usually not needed immediately on a write miss two options exist on a cache write miss:

Write Allocate:
The cache block is loaded on a write miss followed by write hit actions.

No-Write Allocate:
The block is modified in the lower level (lower cache level, or main memory) and not loaded into cache.

While any of the above two write miss policies can be used with either write back or write through:

• Write back caches use write allocate to capture subsequent writes to the block in cache.
• Write through caches usually use no-write allocate since subsequent writes still have to go to memory.
Memory Access Tree, Unified $L_1$
Write Through, No Write Allocate, No Write Buffer

CPU Memory Access

Read

$L_1$

L1 Read Hit: Access Time = 1
Stalls = 0

L1 Read Miss:
Access Time = $M + 1$
Stalls Per access
% reads $\times$ $(1 - H1) \times M$

Write

L1 Write Hit:
Access Time: $M + 1$
Stalls Per access:
% write $\times$ $(H1) \times M$

L1 Write Miss:
Access Time: $M + 1$
Stalls per access:
% write $\times$ $(1 - H1) \times M$

Stall Cycles Per Memory Access = % reads $\times$ $(1 - H1) \times M$ + % write $\times M$

$AMAT = 1 + \% \text{ reads} \times (1 - H1) \times M + \% \text{ write} \times M$
Memory Access Tree Unified $L_1$
Write Back, With Write Allocate

CPU Memory Access

Read

$L_1$

$L_1$ Hit:
% read x H1
Access Time = 1
Stalls = 0

$L_1$ Read Miss

Clean
Access Time = M +1
Stall cycles = M x (1-H1) x
% reads x % clean

Dirty
Access Time = 2M +1
Stall cycles = 2M x (1-H1) x
% read x % dirty

Write

$L_1$ Write Hit:
% write x H1
Access Time = 1
Stalls = 0

$L_1$ Write Miss

Clean
Access Time = M +1
Stall cycles = M x (1-H1) x
% write x % clean

Dirty
Access Time = 2M +1
Stall cycles = 2M x (1-H1) x
% read x % dirty

Stall Cycles Per Memory Access = (1-H1) x ( M x % clean + 2M x % dirty )

AMAT = 1 + Stall Cycles Per Memory Access
Write Through Cache Performance Example

• A CPU with $\text{CPI}_{\text{execution}} = 1.1$ uses a unified L1 Write Through, No Write Allocate and no write buffer.

• Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control

• Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem stalls per instruction}
\]

\[
\text{Mem Stalls per instruction} = \text{Mem accesses per instruction} \times \text{Stalls per access}
\]

\[
\text{Mem accesses per instruction} = 1 + .3 = 1.3
\]

\[
\text{Stalls per access} = \% \text{reads} \times \text{miss rate} \times \text{Miss penalty} + \% \text{write} \times \text{Miss penalty}
\]

\[
\% \text{reads} = 1.15/1.3 = 88.5\% \quad \% \text{writes} = .15/1.3 = 11.5\%
\]

\[
\text{Stalls per access} = 50 \times (88.5\% \times 1.5\% + 11.5\%) = 6.4 \text{ cycles}
\]

\[
\text{Mem Stalls per instruction} = 1.3 \times 6.4 = 8.33 \text{ cycles}
\]

\[
\text{AMAT} = 1 + 8.33 = 9.33 \text{ cycles}
\]

\[
\text{CPI} = 1.1 + 8.33 = 9.43
\]

The ideal memory CPU with no misses is $2.075/1.1 = 8.57$ times faster
Write Back Cache Performance Example

- A CPU with $\text{CPI}_{\text{execution}} = 1.1$ uses a unified L1 with write back, with write allocate, and the probability a cache block is dirty = 10%
- Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

$$\text{CPI} = \text{CPI}_{\text{execution}} + \text{mem stalls per instruction}$$

$$\text{Mem Stalls per instruction} = \text{Mem accesses per instruction} \times \text{Stalls per access}$$

$$\text{Mem accesses per instruction} = 1 + .3 = 1.3$$

$$\text{Stalls per access} = (1-H_1) \times (M \times \% \text{ clean} + 2M \times \% \text{ dirty})$$

$$\text{Stalls per access} = 1.5\% \times (50 \times 90\% + 100 \times 10\%) = .825 \text{ cycles}$$

$$\text{Mem Stalls per instruction} = 1.3 \times .825 = 1.07 \text{ cycles}$$

$$\text{AMAT} = 1 + 1.07 = 2.07 \text{ cycles}$$

$$\text{CPI} = 1.1 + 1.07 = 2.17$$

The ideal CPU with no misses is $2.17/1.1 = 1.97$ times faster
Impact of Cache Organization: An Example

Given:

- A perfect CPI with cache = 2.0    Clock cycle = 2 ns
- 1.3 memory references/instruction Cache size = 64 KB with
- Cache miss penalty = 70 ns, no stall on a cache hit
- One cache is direct mapped with miss rate = 1.4%
- The other cache is two-way set-associative, where:
  - CPU time increases 1.1 times to account for the cache selection multiplexor
  - Miss rate = 1.0%

Average memory access time = Hit time + Miss rate x Miss penalty

Average memory access time \(1\text{-way} = 2.0 + (0.014 \times 70) = 2.98\) ns
Average memory access time \(2\text{-way} = 2.0 \times 1.1 + (0.010 \times 70) = 2.90\) ns

CPU time = \(IC \times [CPI_{\text{execution}} + \text{Memory accesses/instruction} \times \text{Miss rate} \times \text{Miss penalty}] \times \text{Clock cycle time}\)

\(\text{CPUtime}_{1\text{-way}} = IC \times (2.0 \times 2 + (1.3 \times 0.014 \times 70)) = 5.27 \times IC\)
\(\text{CPUtime}_{2\text{-way}} = IC \times (2.0 \times 2 \times 1.10 + (1.3 \times 0.01 \times 70)) = 5.31 \times IC\)

→ In this example, 1-way cache offers slightly better performance with less complex hardware.
2 Levels of Cache: $L_1$, $L_2$

- **CPU**

  - **$L_1$ Cache**
    - Hit Rate = $H_1$, Hit time = 1 cycle
    - (No Stall)

  - **$L_2$ Cache**
    - Hit Rate = $H_2$, Hit time = $T_2$ cycles

- **Main Memory**

  Memory access penalty, $M$
2-Level Cache Performance

CPU\text{time} = IC \times (CPI_{\text{execution}} + \text{Mem Stall cycles per instruction}) \times C

Mem Stall cycles per instruction = Mem accesses per instruction \times Stall cycles per access

• For a system with 2 levels of cache, assuming no penalty when found in L_1 cache:

\text{Stall cycles per memory access} =

\left[ \text{miss rate } L_1 \right] \times \left[ \text{Hit rate } L_2 \times \text{Hit time } L_2 \\
+ \text{Miss rate } L_3 \times \text{Memory access penalty} \right] =

(1-H_1) \times H_2 \times T_2 + (1-H_1)(1-H_2) \times M

L_1 \text{ Miss, } L_2 \text{ Hit} \quad \text{L_1 Miss, L_2 Miss: Must Access Main Memory}
2-Level Cache Performance
Memory Access Tree

CPU Stall Cycles Per Memory Access

CPU Memory Access

L1
L1 Hit:
Stalls = H1 x 0 = 0
(No Stall)
L1 Miss:
\( \% = (1-H1) \)

L2
L2 Hit:
\((1-H1) x H2 x T2\)
L2 Miss:
Stalls = \((1-H1)(1-H2) x M\)

Stall cycles per memory access = \((1-H1) x H2 x T2 + (1-H1)(1-H2) x M\)
AMAT = \(1 + (1-H1) x H2 x T2 + (1-H1)(1-H2) x M\)
Two-Level Cache Example

- CPU with $\text{CPI}_{\text{execution}} = 1.1$ running at clock rate = 500 MHZ
- 1.3 memory accesses per instruction.
- $L_1$ cache operates at 500 MHZ with a miss rate of 5%
- $L_2$ cache operates at 250 MHZ with miss rate 3%, ($T_2 = 2$ cycles)
- Memory access penalty, $M = 100$ cycles. Find CPI.

\[
\text{CPI} = \text{CPI}_{\text{execution}} + \text{Mem Stall cycles per instruction}
\]

With No Cache, \( \text{CPI} = 1.1 + 1.3 \times 100 = 131.1 \)

With single $L_1$, \( \text{CPI} = 1.1 + 1.3 \times 0.05 \times 100 = 7.6 \)

Mem Stall cycles per instruction = Mem accesses per instruction $\times$ Stall cycles per access

Stall cycles per memory access = \( (1-H_1) \times H_2 \times T_2 + (1-H_1)(1-H_2) \times M \)
\[
= 0.05 \times 0.97 \times 2 + 0.05 \times 0.03 \times 100
\]
\[
= 0.097 + 0.15 = 0.247
\]

Mem Stall cycles per instruction = Mem accesses per instruction $\times$ Stall cycles per access
\[
= 0.247 \times 1.3 = 0.32
\]

\[
\text{CPI} = 1.1 + 0.32 = 1.42
\]

\[
\text{Speedup} = 7.6/1.42 = 5.35
\]
3 Levels of Cache

- **CPU**
- **L1 Cache**: Hit Rate = $H_1$, Hit time = 1 cycle
- **L2 Cache**: Hit Rate = $H_2$, Hit time = $T_2$ cycles
- **L3 Cache**: Hit Rate = $H_3$, Hit time = $T_3$ cycles
- **Main Memory**

Memory access penalty, $M$
3-Level Cache Performance

CPU time = IC \times (CPI_{\text{execution}} + \text{Mem Stall cycles per instruction}) \times C

Mem Stall cycles per instruction = \text{Mem accesses per instruction} \times \text{Stall cycles per access}

- For a system with 3 levels of cache, assuming no penalty when found in \( L_1 \) cache:

Stall cycles per memory access =

\[
[\text{miss rate } L_1] \times \left[ \text{Hit rate } L_2 \times \text{Hit time } L_2 \\
+ \text{Miss rate } L_2 \times (\text{Hit rate } L_3 \times \text{Hit time } L_3 \\
+ \text{Miss rate } L_3 \times \text{Memory access penalty}) \right] = \\
(1-H_1) \times H_2 \times T_2 \\
+ (1-H_1) \times (1-H_2) \times H_3 \times T_3 \\
+ (1-H_1)(1-H_2)(1-H_3) \times M
\]
3-Level Cache Performance
Memory Access Tree
CPU Stall Cycles Per Memory Access

CPU Memory Access

L1 Hit:
Stalls = H1 x 0 = 0
(No Stall)

L1 Miss:
% = (1-H1)

L2 Hit:
(1-H1) x H2 x T2

L2 Miss:
% = (1-H1)(1-H2)

L3 Hit:
(1-H1) x (1-H2) x H3 x T3

L3 Miss:
(1-H1)(1-H2)(1-H3) x M

Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1) x (1-H2) x H3 x T3 + (1-H1)(1-H2)(1-H3) x M
AMAT = 1 + Stall cycles per memory access
Three-Level Cache Example

- CPU with $CPI_{\text{execution}} = 1.1$ running at clock rate = 500 MHZ
- 1.3 memory accesses per instruction.
- $L_1$ cache operates at 500 MHZ with a miss rate of 5%
- $L_2$ cache operates at 250 MHZ with miss rate 3%, ($T_2 = 2$ cycles)
- $L_3$ cache operates at 100 MHZ with miss rate 1.5%, ($T_3 = 5$ cycles)
- Memory access penalty, $M = 100$ cycles. Find CPI.

With No Cache, $CPI = 1.1 + 1.3 \times 100 = 131.1$

With single $L_1$, $CPI = 1.1 + 1.3 \times 0.05 \times 100 = 7.6$

With $L_1$, $L_2$ $CPI = 1.1 + 1.3 \times (0.05 \times 0.97 \times 2 + 0.05 \times 0.03 \times 100) = 1.42$

CPI = $CPI_{\text{execution}} + \text{Mem Stall cycles per instruction}$

Mem Stall cycles per instruction = Mem accesses per instruction $\times$ Stall cycles per access

Stall cycles per memory access $= (1-H_1) \times H_2 \times T_2 + (1-H_1) \times (1-H_2) \times H_3 \times T_3 + (1-H_1)(1-H_2)(1-H_3) \times M$

$= 0.05 \times 0.97 \times 2 + 0.05 \times 0.03 \times 0.985 \times 5 + 0.05 \times 0.03 \times 0.015 \times 100$

$= 0.097 + 0.0075 + 0.00225 = 0.107$

CPI = $1.1 + 1.3 \times 0.107 = 1.24$

Speedup compared to L1 only $= 7.6/1.24 = 6.12$

Speedup compared to L1, L2 $= 1.42/1.24 = 1.15$