4-Way Set Associative VIPT Instruction Cache

for RISC-V Processor Architecture

A comprehensive guide from first principles to full implementation — covering cache theory, address translation, hardware design, and RISC-V integration.

RISC-V ISA | Cache Architecture | VIPT Design | Set Associative | Computer Architecture

May 2026 | From Basics to Advanced

Table of Contents

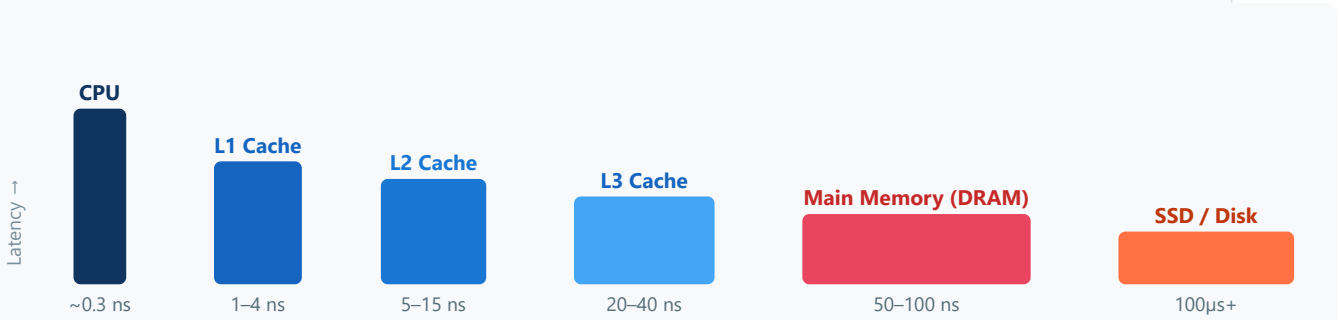
CHAPTER 1

Why Caches Exist — The Memory Hierarchy Problem

1.1 The CPU–Memory Speed Gap

Modern processors execute instructions at gigahertz speeds — one clock cycle can be as short as 0.3 nanoseconds. Yet main memory (DRAM) requires 50–100 nanoseconds to return data after a request. This means the CPU is **100–300× faster** than memory. Without a solution, the CPU would sit idle waiting for every instruction fetch and data access.

FIGURE 1.1 — CPU VS MEMORY LATENCY GAP



1.2 Locality of Reference

Caches work because programs exhibit **locality**. There are two types:

Temporal Locality

If you accessed memory address X recently, you are likely to access it again soon. Example: a loop counter is read and incremented thousands of times.

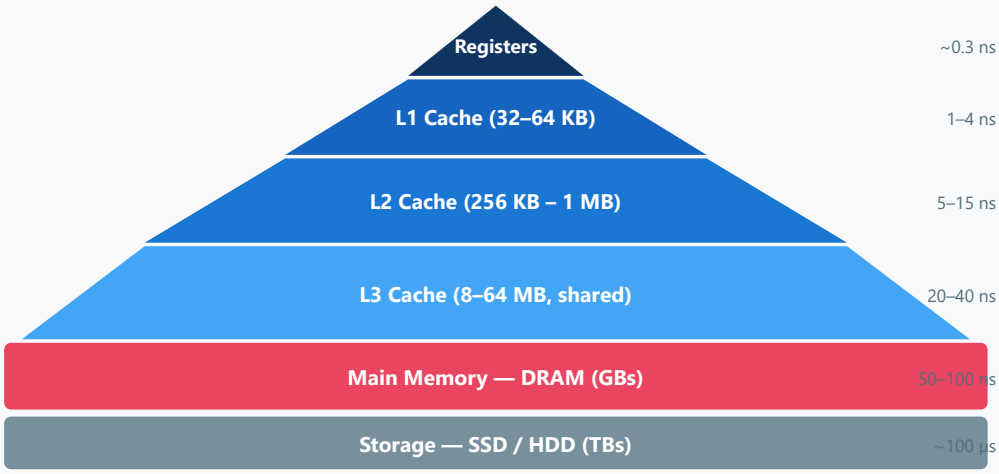
Spatial Locality

If you accessed address X, you are likely to access nearby addresses (X+4, X+8…) soon. Example: reading sequential instructions or array elements.

These two properties are the reason caches are effective. A cache exploits temporal locality by keeping recently-used data close to the CPU, and exploits spatial locality by loading an entire *cache line* (typically 64 bytes) instead of just the single byte or word requested.

1.3 The Memory Hierarchy Pyramid

FIGURE 1.2 — MEMORY HIERARCHY PYRAMID



The **L1 instruction cache (I-Cache)** is the very first level after the CPU's fetch unit. It stores recently-fetched instructions so the processor doesn't have to reach all the way to DRAM every cycle. This is exactly what we will design in this document.

CHAPTER 2

Cache Fundamentals — Terminology & Anatomy

2.1 Cache Lines, Sets, and Ways

A cache is organized into **cache lines** (also called blocks). Each line stores a contiguous chunk of memory — typically 64 bytes. We don't cache individual bytes because of spatial locality: if you need byte X, you'll probably need X+1 soon.

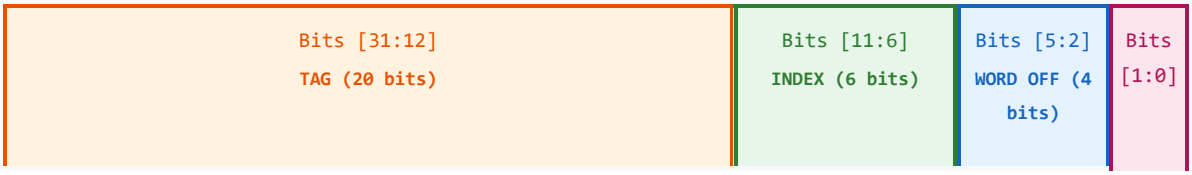
Term	Definition	Typical Value (L1 I-Cache)
Cache Line / Block	The smallest unit of transfer between cache and memory	64 bytes (16 × 32-bit words)
Set	A group of N lines (ways) that a given address can map to	32 or 64 sets
Way	One slot within a set; N-way = N choices per set	4 ways (4-way)
Total Capacity	Sets × Ways × Line Size	64×4×64 = 16 KB
Tag	Upper bits of address stored to verify identity	Remaining bits after index+offset
Index	Bits used to select which set to look in	$\log_2$(#sets) bits
Offset	Bits to select a byte within the cache line	$\log_2$(line size) bits
Valid Bit	1-bit flag: is this line's data meaningful?	1 bit per way

2.2 Address Decomposition: Tag, Index, Offset

Every memory access uses a **physical or virtual address** that is split into three fields. For a 32-bit address with a 16 KB, 4-way, 64-byte-line cache:

```
Cache Size = 16 KB = 16,384 bytes  Ways = 4  Line Size = 64 bytes → Offset bits = log₂(64) = 6 bits  #Sets = 16384 / (4 × 64) = 64 → Index bits = log₂(64) = 6 bits  Tag bits = 32 − 6 − 6 = 20 bits
```

FIGURE 2.1 — 32-BIT ADDRESS DECOMPOSITION



Note: for instruction cache, bits[1:0] are always 00 (word-aligned in RISC-V)

2.3 Valid Bits and Metadata Per Way

Each way in each set stores:

Per-Way Storage

`Valid(1) | Tag(20) | Data(64×8 = 512 bits)` = 533 bits per way. For all 4 ways and 64 sets: 64 × 4 × 533 ≈ 136 KB of raw SRAM. Plus ~14 bits of LRU state per set.

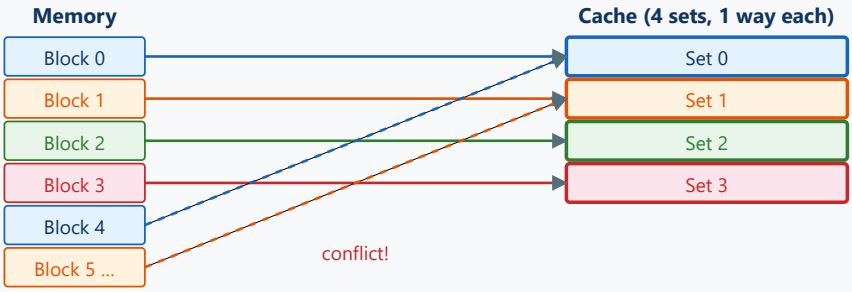
CHAPTER 3

Cache Organization Types

3.1 Direct-Mapped Cache

Each memory block maps to **exactly one** location in cache. Simple, fast — but suffers from *conflict misses* when two heavily-used addresses share the same slot.

FIGURE 3.1 — DIRECT-MAPPED (1-WAY) CACHE



3.2 Fully Associative Cache

A block can be placed in **any** cache line — no index field, just tag. No conflict misses, but requires comparing all tags simultaneously, which is expensive in hardware (many comparators).

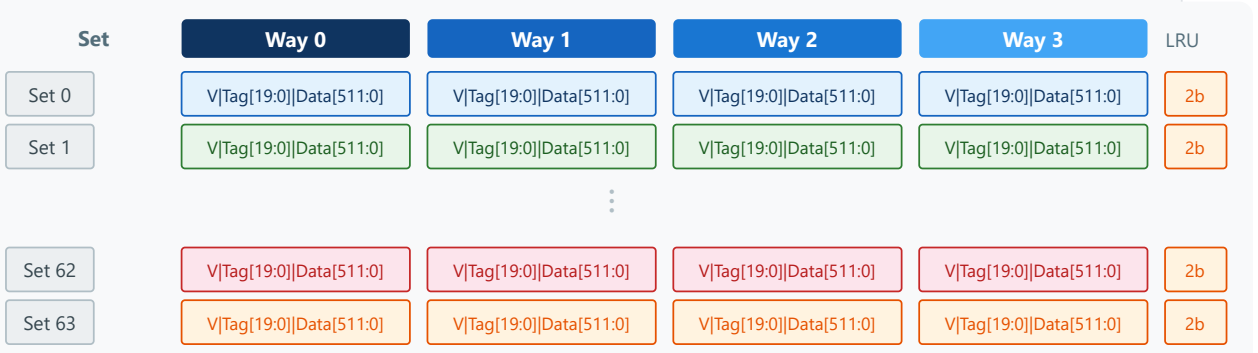
3.3 Set Associative Cache

The practical middle ground. The cache is divided into **S sets**, each holding **N ways**. A block maps to one set (chosen by the index bits) but can go into any of the N ways within that set. This eliminates most conflict misses while keeping hardware complexity manageable.

> **Key Trade-off**
>
> More ways → fewer conflict misses → better hit rate, but more tag comparators, more power, and potentially longer hit latency. 4-way is the most common sweet spot for L1 caches.

3.4 4-Way Set Associative — The Full Picture

FIGURE 3.2 — 4-WAY SET ASSOCIATIVE CACHE STRUCTURE (64 SETS)



Legend:

• V = Valid bit (1 bit) • Tag = Physical tag bits (20 bits) • Data = 64 bytes = 512 bits

• LRU = 2-bit PLRU state per set (4 ways needs $\log_2(4)=2$ bits minimum)

Total: 64 sets × 4 ways × (1+20+512) bits + 64×3 LRU bits ≈ 136 Kb SRAM

CHAPTER 4

Virtual Memory & Address Translation

4.1 Virtual Addresses vs Physical Addresses

Every process running on a CPU uses **virtual addresses** — a private address space that gives the illusion of having all of memory. The operating system and hardware MMU (Memory Management Unit) translate these into **physical addresses** — real locations in DRAM.

Virtual Address (VA)

What the CPU / program sees. Can be up to 48-bit on modern 64-bit systems. Two different processes can have the same VA pointing to different physical data.

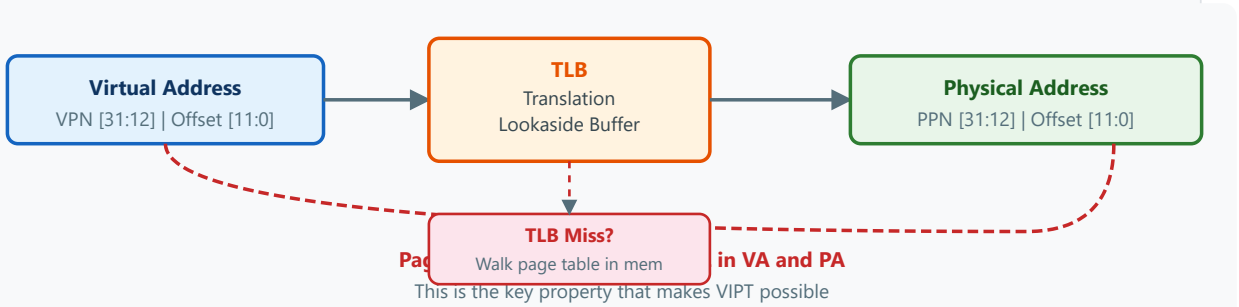
Physical Address (PA)

The actual DRAM row/column. Unique globally. The cache tag stores physical bits to prevent false hits across processes.

4.2 Page Tables and the TLB

Translation happens at **page granularity**. A page is typically 4 KB. The page table maps the upper bits of VA (the *Virtual Page Number, VPN*) to the upper bits of PA (the *Physical Page Number, PPN*). The lower bits (the *page offset*, 12 bits for 4 KB pages) are **identical in both VA and PA** — this is the crucial VIPT insight.

FIGURE 4.1 — VA TO PA TRANSLATION VIA TLB



The Golden Rule of VIPT

> Because the page offset bits are identical in both VA and PA, you can use them as the cache index without waiting for TLB translation. The TLB lookup happens *in parallel* with the SRAM lookup, not sequentially after it. This is why VIPT is fast.

CHAPTER 5

Cache Indexing Strategies

5.1 VIVT — Virtually Indexed, Virtually Tagged

Both the index and tag come from the virtual address. Fast (no TLB needed) but suffers from **aliasing** (two different virtual addresses in different processes map to the same PA — the cache will contain duplicate data) and **homonyms** (same VA, different PA in different processes — the cache may return wrong data). Requires full cache flush on context switch. Rare in modern CPUs.

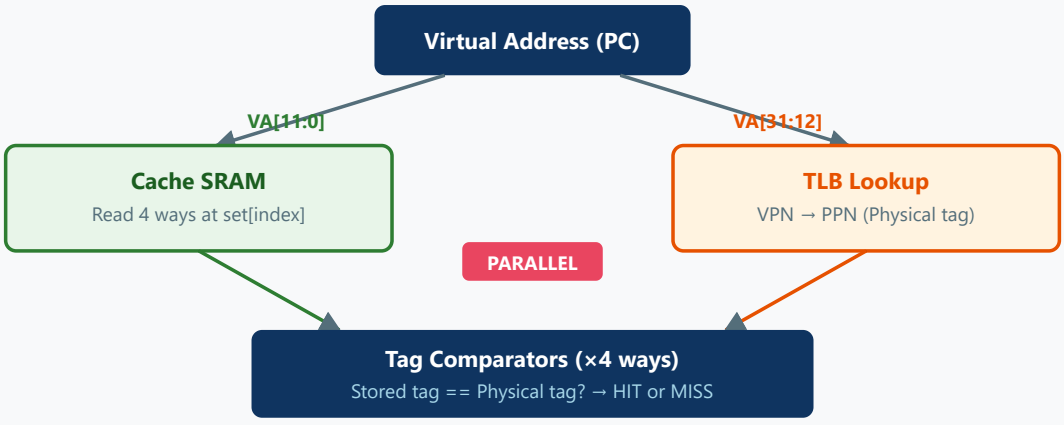
5.2 PIPT — Physically Indexed, Physically Tagged

Both index and tag come from the physical address. **Correct by design** — no aliasing or homonym problems. But the TLB must complete before the cache SRAM can even be addressed. This adds the full TLB latency to every cache hit — critical path penalty for L1.

5.3 VIPT — Virtually Indexed, Physically Tagged

Index comes from the virtual address; tag comes from the physical address. The key insight: **use the TLB and SRAM in parallel**. While the cache SRAM is being read (indexed by VA bits), the TLB translates the upper virtual bits to physical bits. Both complete at roughly the same time, then tag comparison uses the physical tag.

FIGURE 5.1 — VIPT PARALLEL LOOKUP PIPELINE



5.4 The VIPT Aliasing Condition and the Page Offset Rule

VIPT introduces a subtle problem called **virtual aliasing**: two virtual pages can map to the same physical page. If they both index into *different* cache sets, the same data will be stored twice — inconsistently.

> **VIPT Aliasing Condition**
>
> Aliasing is eliminated if and only if the cache index bits fall entirely within the page offset. For 4 KB pages, the page offset is 12 bits (bits [11:0]). The index bits must be a subset of [11:0].
>
> Our design: Index = bits[11:6], Offset = bits[5:0]. Index high bit = bit 11. Page offset top = bit 11. ✓ Safe — no aliasing.

```
VIPT is alias-free if: (index_bits + offset_bits) ≤ page_offset_bits Our example: index(6) + offset(6) = 12 ≤ 12 (for 4KB pages) ✓ Maximum alias-free cache size = Ways × Page_Size = 4 × 4KB = 16 KB (This is why L1 caches are typically ≤ 32 KB with 8-way or ≤ 16 KB with 4-way)
```

Strategy	Index Source	Tag Source	TLB on Critical Path?	Aliasing?	Used In
VIVT	Virtual	Virtual	No	Yes (severe)	Old ARM, rare
PIPT	Physical	Physical	Yes	No	D-Cache, L2/L3
VIPT	**Virtual**	**Physical**	**No (parallel)**	**Conditional**	**L1 I-Cache (most modern CPUs)**

CHAPTER 6

VIPT Cache Architecture — Full Design

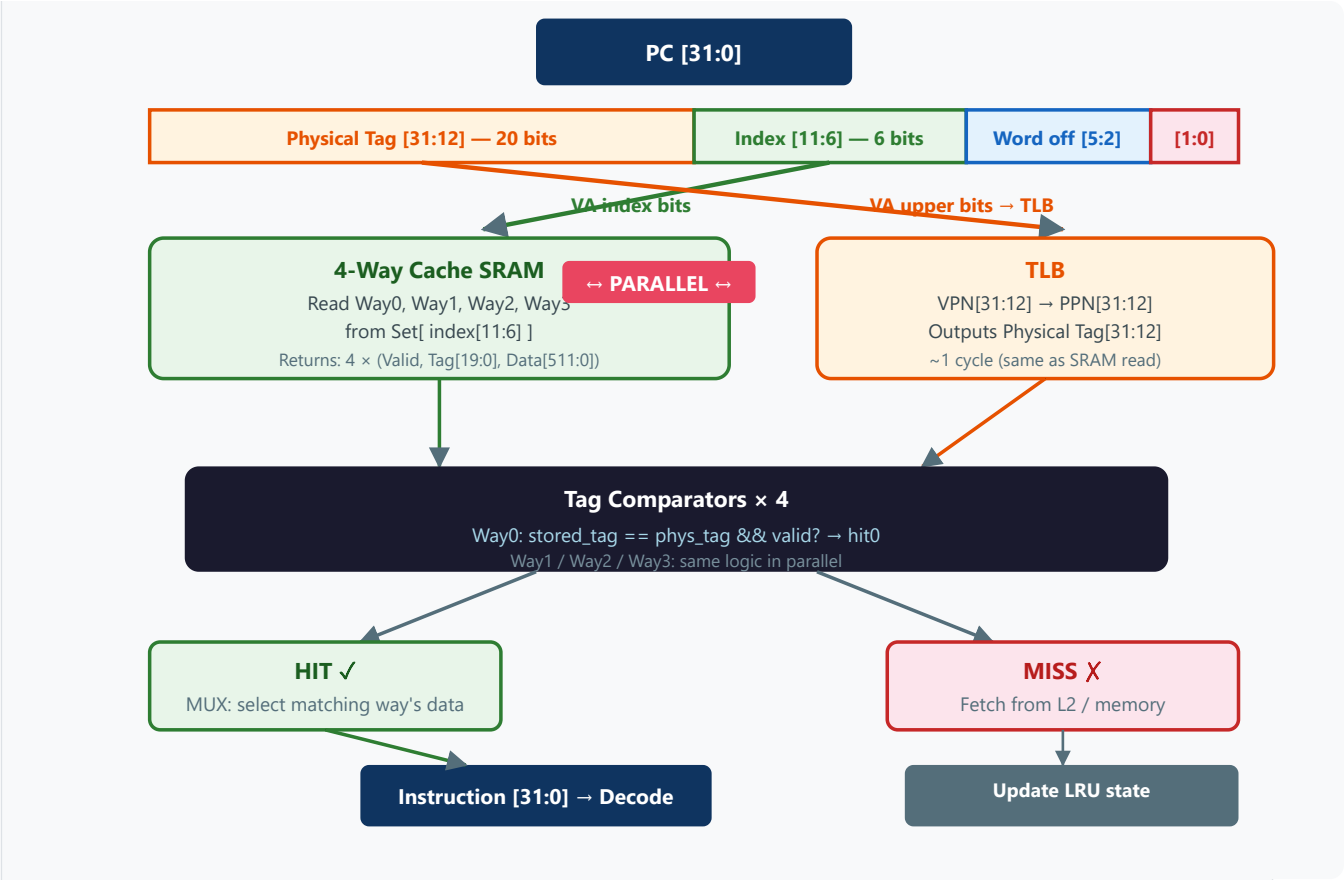
6.1 Address Map for 32-bit RISC-V

Our design targets a 32-bit RISC-V core with a **16 KB, 4-way set associative, 64-byte line** instruction cache. Here are all the parameters derived step by step:

Parameter	Value	Derivation
Address width	32 bits	RISC-V RV32I
Cache capacity	16 KB	Design choice
Associativity	4 ways	Design choice
Cache line size	64 bytes	Design choice (matches burst length)
Number of sets	64	16384 / (4 × 64) = 64
Block offset bits	6	$\log_2(64) = 6$, bits[5:0]
Index bits	6	$\log_2(64) = 6$, bits[11:6]
Tag bits (physical)	20	32 − 6 − 6 = 20, bits[31:12]
Page offset (4 KB page)	12 bits	$\log_2(4096) = 12$
Alias-free check	6+6=12 ≤ 12 ✓	VIPT safe
SRAM per way	(1+20+512) = 533 bits × 64	V + Tag + Data per line

6.2 Cache Lookup Pipeline — Cycle by Cycle

FIGURE 6.1 — FULL VIPT I-CACHE LOOKUP DATAPATH



6.3 Tag Comparison and Hit Detection Logic

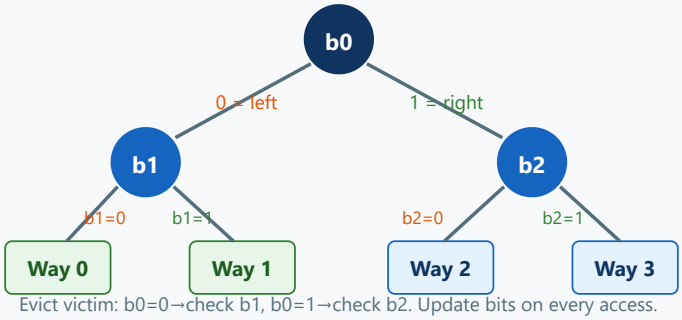
All four tag comparisons happen simultaneously in hardware — this is what *associativity* means in implementation. The logic for each way is:

```
hit_way_N = valid_N AND (stored_tag_N == physical_tag_from_TLB) overall_hit =
hit_way_0 OR hit_way_1 OR hit_way_2 OR hit_way_3 // Select the data from
whichever way hit: instr_out = MUX4(data_way_0, data_way_1, data_way_2,
data_way_3, {hit_way_3, hit_way_2, hit_way_1, hit_way_0})
```

6.4 Replacement Policy: LRU and PLRU

On a miss, we must evict one of the 4 ways. The most accurate policy is **True LRU** (Least Recently Used), but it requires $\log_2(4!) = 5$ bits per set. A practical approximation is **Pseudo-LRU (PLRU)** using a 3-bit binary tree per set.

FIGURE 6.2 — 4-WAY PLRU BINARY TREE (3 BITS PER SET)



Evict victim: b0=0→check b1, b0=1→check b2. Update bits on every access.

CHAPTER 7

Instruction Cache Specifics

7.1 Why I-Cache is Read-Only

The instruction cache is **read-only from the CPU's perspective**. The fetch unit never writes instructions — it only reads them. This has major design implications:

- **No dirty bits needed:** Data cache lines must track if data was modified (dirty) so it can be written back. I-cache lines never get modified — if evicted, they're just dropped.
- **No write-back logic:** Evictions are simple — just mark the way invalid and load the new line.
- **No write allocate:** All misses go directly to the refill path without allocating in the D-cache.
- **Simpler coherence:** The I-cache doesn't participate in store-to-load forwarding or store queue snooping.

7.2 Self-Modifying Code and FENCE.I

A problem arises with **JIT compilers or loaders** that write instructions to memory via the D-cache, then try to execute them. The I-cache and D-cache are separate; writing through the D-cache doesn't automatically update the I-cache. The program may fetch stale instructions.

> **RISC-V FENCE.I Instruction**
>
> RISC-V provides the `FENCE.I` instruction (Zifencei extension) to synchronize the instruction stream. When executed, it guarantees that any prior stores are visible to subsequent instruction fetches. Hardware implementations typically respond by flushing and invalidating the entire I-cache.

7.3 Cache Invalidation on Context Switch

With VIPT using physical tags, the cache is naturally process-safe — a tag miss from a different process's PA simply won't match. However, if the OS reuses a physical page for a different virtual mapping, stale I-cache data with the old physical tag could match. This is solved by:

ASID (Address Space ID)

Include the process ASID in the tag match. Lines from different processes never match even if PA is recycled. Avoids full flush on context switch.

Full Flush

Clear all valid bits on context switch. Simple but wastes the warm cache. Acceptable in embedded systems, costly in OS-heavy workloads.

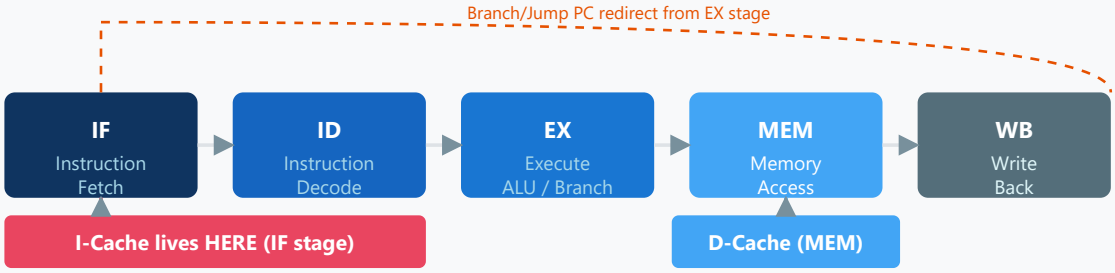
CHAPTER 8

RISC-V Architecture & I-Cache Integration

8.1 RISC-V Pipeline Overview

A standard 5-stage RISC-V pipeline (as in the reference Rocket or CVA6 cores) operates as:

FIGURE 8.1 — 5-STAGE RISC-V PIPELINE WITH I-CACHE



8.2 Where I-Cache Sits in the Pipeline

The **IF (Instruction Fetch)** stage drives the I-cache every cycle with the current PC. The cache must return the instruction within the same cycle (or signal a stall). For a typical 1 GHz embedded RISC-V core, the I-cache must deliver a result in ~1 ns — this means the SRAM access time and tag comparison must fit within one clock cycle.

8.3 PC Alignment in RISC-V

RISC-V RV32I instructions are always **4 bytes (32 bits) wide and word-aligned**. This means:

- Bits [1:0] of the PC are always `00` — they are never used for cache indexing.
- The byte offset within a cache line effectively uses bits [5:2] (word offset, 4 bits) since [1:0]=00.
- RISC-V C extension (compressed instructions) uses 16-bit alignment — bits[0] would be 0, but [1] could be non-zero. Our design assumes RV32I without C.

> **Practical Impact**
>
> Each cache line holds 64 bytes ÷ 4 bytes/instr =
> **16 instructions**

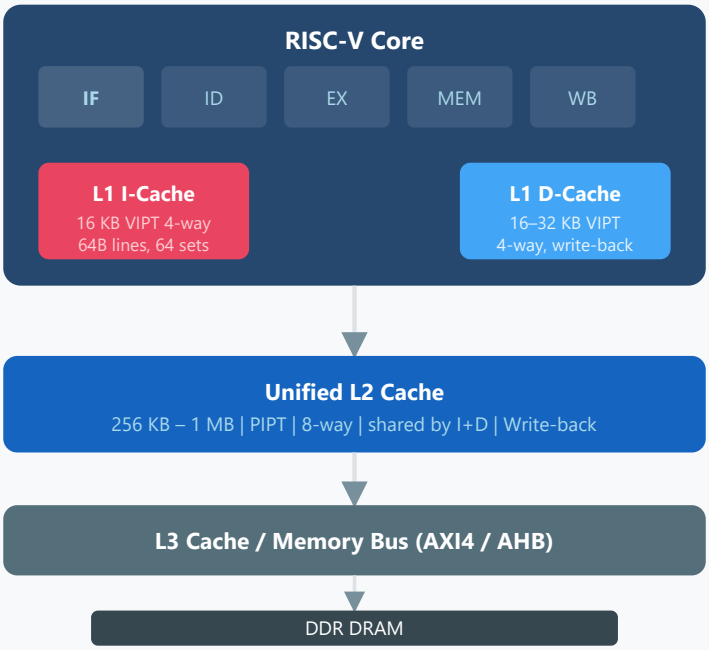
. On a cache miss, 16 instructions are fetched in one burst from L2/memory, amortizing the miss penalty. This is why spatial locality matters so much for instruction caches.

CHAPTER 9

Bigger Picture — Cache in a Real SoC

9.1 L1 / L2 / L3 Hierarchy in Context

FIGURE 9.1 — FULL SOC CACHE HIERARCHY WITH RISC-V CORE



9.2 Cache Coherence Basics

In a single-core RISC-V system, I-cache coherence is managed manually via `FENCE.I`. In multi-core systems, a hardware coherence protocol (MESI, MOESI) maintains consistency. The I-cache typically operates in a *read-only invalid/valid* state machine — it never produces modified data, so it participates in coherence as a snooper only.

9.3 Critical Path and Timing

The I-cache is on the **critical path** of the processor — it determines the minimum clock period. The critical path flows:

```
t_cycle ≥ t_SRAM_read + t_tag_compare + t_mux + t_setup Typical L1 SRAM read:
~200–400 ps (TSMC 28nm) Tag comparator (XOR + NOR): ~80 ps 4:1 MUX: ~60 ps Flip-
```

```
flop setup: ~50 ps ──────────────────────────── Total: ~400–600 ps → cycle time
≥ 600 ps → max freq ~1.6 GHz
```

This is why L1 caches are kept small (16–32 KB) — larger SRAMs have longer access times, violating timing. L2 caches run slower and can be larger.

CHAPTER 10

Worked Examples & Parameter Tables

Example 1: Cache Hit Trace

PC = `0x80001234`. Is there a hit?

```
Address: 0x80001234 = 1000_0000_0000_0000_0001_0010_0011_0100 (binary) Offset [5:0] = 11_0100 = 0x34 (but word aligned: [5:2]=1101=13, byte in word [1:0]=00) Index [11:6] = 00_0100 = Set 4 Tag [31:12]= 0x80001 → Physical tag after TLB = 0x80001 1. Read Set 4 from SRAM: get Way0..Way3 tags and valid bits 2. TLB maps 0x80001 (VPN) → 0x80001 (PPN) [identity mapped in this example] 3. Compare physical tag 0x80001 with stored tags in Set 4: - Way0: valid=1, tag=0x80001 → MATCH → HIT in Way 0! 4. Select Way 0 data, extract word at offset 13 → instr[31:0] 5. Update PLRU: Way 0 most recently used
```

Example 2: Cache Miss Trace

```
PC = 0xC0008000 Tag = 0xC0008, Index = Set 0, Offset = 0 1. Read Set 0: Way0(tag=0x00001,V=1), Way1(tag=0x00002,V=1), Way2(tag=0x80001,V=1), Way3(tag=0xA0001,V=1) 2. TLB returns physical tag = 0xC0008 3. No tag matches → MISS 4. Stall pipeline (insert bubbles) 5. PLRU selects victim way (say Way 1) 6. Issue L2 read: fetch 64 bytes at PA 0xC0008000 7. Fill Way 1 with new data, set valid=1, tag=0xC0008 8. Resume pipeline
```

Complete Parameter Table

Parameter	Our Design	Alternative (32KB, 8-way)
Total Size	16 KB	32 KB
Ways	4	8
Sets	64	64
Line size	64 bytes	64 bytes
Offset bits	6 (bits[5:0])	6 (bits[5:0])

Index bits	6 (bits[11:6])	6 (bits[11:6])
Tag bits	20 (bits[31:12])	20 (bits[31:12])
VIPT alias-free?	✓ 6+6=12≤12	✓ 6+6=12≤12
LRU bits/set	3 (PLRU tree)	7 (PLRU tree)
Tag comparators	4	8
SRAM area	~0.02 mm^2 (28nm)	~0.04 mm^2 (28nm)
Typical hit rate	~97–98%	~98–99%
Access time	~1 cycle	~1–2 cycles

CHAPTER 11

Glossary

Term	Definition
Associativity (N-way)	Number of cache lines within a set where a block can be placed
ASID	Address Space Identifier — process tag added to TLB/cache to avoid flushes on context switch
Cache Line / Block	The atomic unit of data moved between cache levels (typically 64 bytes)
Cold Miss	Miss because data has never been loaded (compulsory miss)
Conflict Miss	Miss because two lines compete for the same set (eliminated by more ways)
Capacity Miss	Miss because cache is too small to hold the working set
Critical Path	Longest combinational logic path that limits maximum clock frequency
D-Cache	Data cache — serves load/store instructions in the MEM pipeline stage
DRAM	Dynamic RAM — main memory, slow (50–100 ns) but large (GBs)
FENCE.I	RISC-V instruction to flush I-cache and synchronize instruction stream
Hit / Miss	Hit: requested data found in cache. Miss: not found, must fetch from lower level
I-Cache	Instruction cache — supplies instructions to the IF stage of the pipeline
Index	Address bits used to select which set to look in
LRU	Least Recently Used — eviction policy that removes the line accessed longest ago
MMU	Memory Management Unit — hardware that performs VA→PA translation
Offset	Address bits selecting a byte within a cache line
PA (Physical Address)	Real address in DRAM after MMU translation
Page	Fixed-size block of memory (typically 4 KB) — the unit of VA→PA mapping
PIPT	Physically Indexed, Physically Tagged — correct but TLB is on critical path
PLRU	Pseudo-LRU — approximation of LRU using a binary tree, fewer bits

PPN	Physical Page Number — upper bits of a physical address
RISC-V	Open-source reduced instruction set computer architecture
Set	A group of N ways in a cache; an address maps to exactly one set
Spatial Locality	Tendency to access addresses near recently-used addresses
SRAM	Static RAM — fast (sub-nanosecond), used for cache storage
Tag	Upper address bits stored alongside data to verify a cache line's identity
Temporal Locality	Tendency to re-access recently-used memory locations
TLB	Translation Lookaside Buffer — fast cache of recent VA→PA translations
VA (Virtual Address)	Address as seen by the program; private per-process view
Valid Bit	1-bit flag per cache way indicating whether stored data is meaningful
VIPT	Virtually Indexed, Physically Tagged — indexes via VA, tags via PA; fast and safe if alias condition met
VIVT	Virtually Indexed, Virtually Tagged — fast but prone to aliasing and homonym problems
VPN	Virtual Page Number — upper bits of a virtual address
Way	One slot within a set — N-way cache has N ways per set
Write-Back	D-cache policy: write to cache only, flush to memory on eviction
Write-Through	D-cache policy: every write goes to both cache and memory immediately

4-Way Set Associative VIPT Instruction Cache for RISC-V

From first principles to full SoC integration

To save as PDF: Open in browser → Print → Save as PDF → Set paper to A4/Letter, enable Background Graphics