# Design 128kw, Fully Associative Cache That Has 4 32-bit Words Per Block. Assume A 32 Bit Address. Calculate

**Design 128kw, Fully Associative Cache That Has 4 32-bit Words Per Block. Assume A 32 Bit Address. Calculate**

Designing a cache system is crucial for enhancing the performance of computer architectures by reducing the average time to access data from the main memory. In this comprehensive article, we will explore the design of a 128kw fully associative cache with four 32-bit words per block, assuming a 32-bit address. We will delve into the key concepts, calculations, and design considerations involved in creating such a cache system, providing detailed explanations suitable for students, engineers, and enthusiasts interested in computer architecture.

---

## Understanding Cache Memory and Its Significance

Cache memory acts as a high-speed buffer between the processor and the main memory, storing frequently accessed data to speed up processing times. The primary goal of cache design is to maximize hit rates while maintaining manageable complexity and cost.

Key terms to understand:

- Cache Size: Total capacity of the cache memory.
- Block Size (Line Size): Number of bytes or words stored in a single cache line.
- Associativity: How cache lines are mapped to memory addresses (fully associative, set associative, direct mapped).
- Tag, Index, Offset: Components of the memory address used to locate data within the cache.

In this context, we're focusing on a fully associative cache, which means any block can be placed in any cache line, offering maximum flexibility and potentially higher hit rates.

---

## Defining the Cache Parameters

To design the cache, we need to specify several parameters:

- Cache Size (Total Capacity): 128 KB (kilobytes) or 128,000 bytes.
- Block Size: 4 words per block, with each word being 32 bits or 4 bytes.
- Address Size: 32 bits.

Let's interpret these parameters:

- Total Cache Size: 128 KB = 128 1024 bytes = 131,072 bytes.
- Block Size: 4 words per block 4 bytes per word = 16 bytes per block.
- Number of Blocks: Total cache size / block size = 131,072 bytes / 16 bytes = 8,192 blocks.

Since the cache is fully associative, all these blocks form a single pool where any block can be stored anywhere.

---

# Calculating Cache Components

To understand how data is found and stored in the cache, we break down the 32-bit address into three parts: Tag, Index, and Offset.

2.1 Offset Bits

The offset determines the specific byte within a block:

- Each block is 16 bytes.
- Number of offset bits: log2(16) = 4 bits.

2.2 Index Bits

Since the cache is fully associative:

- There is no index field because any block can go anywhere.
- Therefore, the index field size = 0 bits.

2.3 Tag Bits

Remaining bits in the address are used for the tag:

- Total address bits = 32.
- Offset bits = 4.
- Index bits = 0.
- Tag bits = 32 - 4 - 0 = 28 bits.

2.4 Summary of Address Breakdown

| Field | Bits | Description |
|---|---|---|
| Tag | 28 | Used to identify if data in cache matches request |
| Index | 0 | Not used in fully associative cache |

| Offset | 4 | Byte within the block |

---

# Calculating Cache Size and Data Structures

Understanding the cache's structure is critical for implementation and performance analysis.

2.1 Total Number of Cache Lines

- Since the cache is fully associative, it contains 8192 blocks (as calculated earlier).
- Each block contains 4 words or 16 bytes.

2.2 Storage Requirements

- Tag Storage: Each cache line stores a 28-bit tag.
- Data Storage: Each cache line holds 16 bytes.
- Valid Bit: Each line has a valid bit to indicate if the data in the line is valid.
- Possible Additional Bits: Dirty bit if write-back policy is used; for simplicity, we assume no.

2.3 Total Storage Calculation

- Total Tag Storage: 8,192 lines 28 bits = 229,376 bits ≈ 28,672 bytes.
- Total Data Storage: 8,192 lines 16 bytes = 131,072 bytes.
- Total Valid Bits: 8,192 bits ≈ 1,024 bytes.

---

# Accessing the Cache: Lookup and Hit/Miss Analysis

In a fully associative cache, the search involves comparing the tag of each cache line with the tag of the address being accessed.

2.1 Cache Lookup Process

1. Extract the tag from the address (28 bits).
2. Compare this tag with the tag stored in each cache line.
3. If a match is found and the valid bit is set, a cache hit occurs.
4. If no match or valid bits are unset, a cache miss occurs, leading to fetching data from main memory.

2.2 Impact of Fully Associative Design

- High Flexibility: Any block can be placed anywhere, reducing conflict misses.

- Higher Cost and Complexity: Because each lookup requires comparing the tag against all cache lines, hardware comparison circuits are needed.

---

# Design Considerations and Optimization Strategies

Designing an efficient fully associative cache involves balancing performance, cost, and complexity.

2.1 Replacement Policies

- Least Recently Used (LRU): Replaces the block that has been least recently accessed.
- Random: Replaces a random block.
- First-In-First-Out (FIFO): Replaces the oldest block.

2.2 Write Policies

- Write-Back: Data is written to the cache and only written back to main memory upon eviction.
- Write-Through: Data is written simultaneously to cache and main memory.

Choosing the appropriate policy impacts cache performance and complexity.

2.3 Associativity Considerations

While fully associative caches offer high hit rates, they are more expensive and complex. Set associative caches (e.g., 4-way, 8-way) can provide a good trade-off.

---

# Performance Metrics and Calculations

Understanding cache performance involves calculating metrics like hit rate, miss rate, and average access time.

2.1 Hit Rate and Miss Rate

- Hit Rate: Percentage of memory accesses found in cache.
- Miss Rate: Percentage of accesses not found, leading to main memory access.

2.2 Average Memory Access Time (AMAT)

$$ AMAT = (Hit~Time) + (Miss~Rate) \times (Miss~Penalty) $$

- Hit Time: Time to access data in cache.
- Miss Penalty: Time to fetch data from main memory.

Optimizing these metrics involves cache size, associativity, block size, and replacement policies.

---

# Summary of Calculations and Design Steps

| Step | Description | Calculations/Results |
|---|---|---|
| 1 | Cache size | 128 KB |
| 2 | Block size | 16 bytes (4 words) |
| 3 | Number of blocks | 8,192 |
| 4 | Address bits | 32 |
| 5 | Offset bits | 4 |
| 6 | Index bits | 0 (fully associative) |
| 7 | Tag bits | 28 |
| 8 | Total tag storage | ~28.7 KB |
| 9 | Data storage | 128 KB |

---

# Conclusion and Final Remarks

Designing a fully associative 128kw cache with 4 words per block involves careful consideration of address partitioning, hardware complexity, and performance trade-offs. The key takeaways include:

- The cache size and block size determine the number of blocks and address breakdown.
- Fully associative caches provide optimal flexibility but at higher hardware costs.
- Tag, data, and control bits must be properly allocated for efficient operation.
- Performance is influenced by cache hit/miss rates, which depend on workload and cache design.

This comprehensive approach provides a solid foundation for understanding how to design, analyze, and optimize a fully associative cache system with the specified parameters. Future enhancements could include set-associative designs, advanced replacement policies, and multi-level cache hierarchies to further improve system performance.

---

Keywords: Fully associative cache, cache design, cache size calculation, 32-bit address, cache block, cache performance, cache architecture, memory hierarchy, cache optimization

# Frequently Asked Questions

## What is the total size of the cache in bits for a 128kW fully associative cache with 4 words per block and 32-bit words?

First, convert 128kW to bytes: 128kW = 128,000 W. Since power (W) isn't typically used for cache size, assuming '128kW' was meant to be 128 KB (kilobytes), then total size = 128 KB 1024 bytes/KB 8 bits/byte = 1,048,576 bits.

## How many blocks are present in the cache given the cache size and block size?

Each block contains 4 words, and each word is 32 bits, so block size = 4 32 bits = 128 bits. Total cache size in bits (assuming 128 KB): 1,048,576 bits. Number of blocks = total size / block size = 1,048,576 / 128 = 8,192 blocks.

## What is the number of bits needed for the block offset in the address?

Since each block contains 4 words and each word is 32 bits, total bits per block = 128 bits. To index within a block, we need log2(4) = 2 bits for the block offset.

## How many bits are needed for the tag in the address?

Given a 32-bit address, and with 2 bits for block offset, and assuming fully associative cache with no index bits, the entire address minus offset bits is used for the tag. Therefore, tag bits = 32 - 2 = 30 bits.

## What is the total number of bits used for the tag, index, and offset combined?

Total address bits = 32; offset bits = 2; index bits = 0 (fully associative). Thus, total bits = 30 (tag) + 0 (index) + 2 (offset) = 32 bits.

## How would you calculate the total number of cache lines in this cache?

In a fully associative cache, each block is a line. Number of cache lines = total cache size in bits / size per block in bits = 1,048,576 / 128 = 8,192 lines.

## What is the significance of the cache being fully associative in this context?

A fully associative cache allows any block to be placed in any cache line, eliminating the

need for index bits and simplifying address translation to mainly use the tag and offset bits, providing maximum flexibility and potential hit rate at the cost of increased complexity.

## If the cache receives a 32-bit address, how is the address divided into tag, index, and offset bits for this fully associative cache?

In a fully associative cache, the address is divided into a tag (30 bits) and an offset (2 bits), with no index bits. The 30-bit tag identifies the block, and the 2-bit offset specifies the word within the block.

# Additional Resources

Design 128kw, Fully Associative Cache That Has 4 32-bit Words Per Block. Assume A 32 Bit Address. Calculate

In modern computing, cache memory plays a pivotal role in bridging the speed gap between the processor and main memory. As processors become faster and more complex, designing efficient cache architectures becomes essential to optimize performance and minimize latency. Today, we explore a detailed design scenario involving a fully associative cache with specific parameters: a total size of 128 kilowords (kW), each block comprising four 32-bit words, and a 32-bit address space. Our goal is to understand the underlying architecture, perform critical calculations, and shed light on how such a system functions in practice.

---

Understanding the Cache Architecture: Fundamental Concepts

Before diving into calculations, it’s essential to establish a clear understanding of the key components involved:

- Cache Size (Total Storage Capacity): The total amount of data the cache can hold.
- Block Size: The amount of data transferred in a single cache line or block.
- Associativity: How cache lines are mapped; in this case, fully associative, meaning any block can be placed anywhere.
- Address Structure: How the 32-bit address is partitioned into tag, index, and block offset bits.
- Words and Data Width: The size of each word (here, 32 bits) and how many words per block.

---

Step 1: Determining Cache Size in Bytes

The cache size is specified as 128kW (kilowords). Since each word is 32 bits (or 4 bytes), converting this to total bytes is straightforward:

- Number of words: 128 kW = 128,000 words
- Bytes per word: 4 bytes
- Total bytes: 128,000 words × 4 bytes/word = 512,000 bytes

Thus, the cache can hold 512 KB of data.

---

Step 2: Calculating Block Size in Bytes

Each block contains 4 words, and each word is 4 bytes:

- Words per block: 4
- Bytes per word: 4
- Bytes per block: 4 × 4 = 16 bytes

This block size influences how the address is partitioned and affects the number of blocks in the cache.

---

Step 3: Computing the Number of Cache Blocks

Total number of blocks in the cache:

- Total cache size in bytes: 512,000 bytes
- Bytes per block: 16 bytes

Number of blocks:

$$\text{Number of blocks} = \frac{\text{Total bytes}}{\text{Bytes per block}} = \frac{512,000}{16} = 32,000 \text{ blocks}$$

In other words, the cache contains 32,000 blocks.

---

Step 4: Address Breakdown — Tag, Index, and Block Offset

Given a 32-bit address, we need to determine how to split it into:

- Tag bits: Identify if the data in a cache line matches the requested address.
- Index bits: Select a specific cache block (not used in fully associative caches).
- Block Offset bits: Specify the exact word inside a block.

Important note: In a fully associative cache, there is no index field because any block can be placed anywhere; the entire cache is searched in parallel.

4.1 Block Offset Bits

Since each block is 16 bytes, and data is word-aligned:

- Number of bytes per block: 16 bytes
- Number of words per block: 4

To identify a specific word within a block:

$$ \text{Block offset bits} = \log_2(\text{Bytes per block}) = \log_2(16) = 4 \text{ bits} $$

These 4 bits specify which byte within the block is being accessed. However, since the cache is word-oriented, and each word is 4 bytes, we typically address words rather than individual bytes, so:

$$ \text{Word offset bits} = \log_2(\text{Words per block}) = \log_2(4) = 2 \text{ bits} $$

Thus, for word addressing within a block, 2 bits are needed.

4.2 Tag Bits

The remaining bits in the address are used for the tag:

$$ \text{Tag bits} = 32 - \text{Block offset bits} $$

Assuming word-level addressing (i.e., selecting words, not bytes), and considering the 2 bits for word offset:

- Remaining bits for tag:

$$ 32 - 2 = 30 \text{ bits} $$

In full detail:

- Block offset bits: 2 (for selecting the specific word within the block)
- Tag bits: 30

Since no index bits are used in fully associative caches, the entire address bits, apart from the block offset, serve as the tag.

---

Step 5: Calculating the Tag and Data Storage

- Total number of blocks: 32,000
- Number of bits needed to uniquely identify each block (if using direct mapping): $\log_2(32,000) \approx 15$ bits

However, in a fully associative cache, the data structure is different:

- All blocks are stored in a common pool.
- The cache controller must compare the tag bits of the address with all stored tags during a lookup to find a match.

This means:

- Number of tag bits per block: 30 bits (as calculated)
- Total number of tags stored: 32,000

---

Step 6: Addressing and Lookup Process

In a fully associative cache, the lookup process involves:

- Comparing the incoming address’s tag bits (30 bits) with the tags stored in all 32,000 cache lines.
- Using a content-addressable memory (CAM) or parallel comparison circuitry for quick matching.
- If a match is found, it indicates a cache hit; otherwise, a miss.

Because the cache is fully associative, there is no need for index bits to select a cache line, simplifying the address structure but increasing hardware complexity for comparison.

---

Step 7: Summary of Key Parameters

| Parameter | Value | Explanation |
|---|---|---|
| Total cache size | 512 KB | 128,000 words × 4 bytes/word |
| Block size | 16 bytes | 4 words per block |
| Number of blocks | 32,000 | Total size divided by block size |
| Address bits | 32 bits | Given |
| Block offset bits | 2 bits | To select the word within a block |
| Tag bits | 30 bits | Remaining bits in the address |
| Associativity | Fully associative | Any block can be stored anywhere |

---

Practical Implications and Design Considerations

Designing such a cache involves balancing hardware complexity and performance:

- Hardware complexity: Fully associative caches require comparison circuitry that scales with the number of blocks. For 32,000 blocks, this can be significant, leading to increased cost and power consumption.
- Performance benefits: Fully associative caches eliminate conflict misses caused by limited associativity, leading to higher hit rates for workloads with high access diversity.
- Latency considerations: Parallel comparison circuitry helps in achieving fast lookups, but the hardware design becomes more intricate.

---

Final Thoughts: The Significance of Cache Design

Understanding the calculations behind cache architecture is fundamental for computer

engineers aiming to optimize system performance. The specific parameters—cache size, block size, associativity, and address structure—directly influence the speed, cost, and efficiency of a computing system.

In this example, a 128kW fully associative cache with 4-word blocks and a 32-bit address exemplifies a high-capacity cache designed to minimize conflict misses and maximize data access speed. While hardware complexity increases with the size and associativity, innovations in parallel comparison techniques and high-speed circuitry continue to push the boundaries of what is feasible.

As technology advances, the insights gleaned from these calculations inform the design of next-generation caches, ensuring that processors can handle ever-increasing data demands with minimal latency, ultimately leading to faster, more efficient computing systems.

---

In conclusion, designing a 128kw fully associative cache with 4 words per block involves meticulous calculations of size, address partitioning, and hardware implications. By understanding these core principles, designers can craft caches that strike a balance between performance and complexity, keeping pace with the evolving landscape of computing technology.

## Design 128kw Fully Associative Cache That Has 4 32 Bit Words Per Block Assume A 32 Bit Address Calculate

Design 128kw Fully Associative Cache That Has 4 32 Bit Words Per Block Assume A 32 Bit Address Calculate

## Related Articles


- Can Yall Pls Help I Need Number 2 Quick Plsss Help Me Fast I Need It Will Mark Brainliest. This Phsychology
- Electrical Resistance Occurs Because (choose ALL The Correct Ones)and. Electrons Collide With Imperfections
- Current Attempt In Progress The Vice President For Sales And Marketing At Waterways Corporation Is Planning

Back to Home