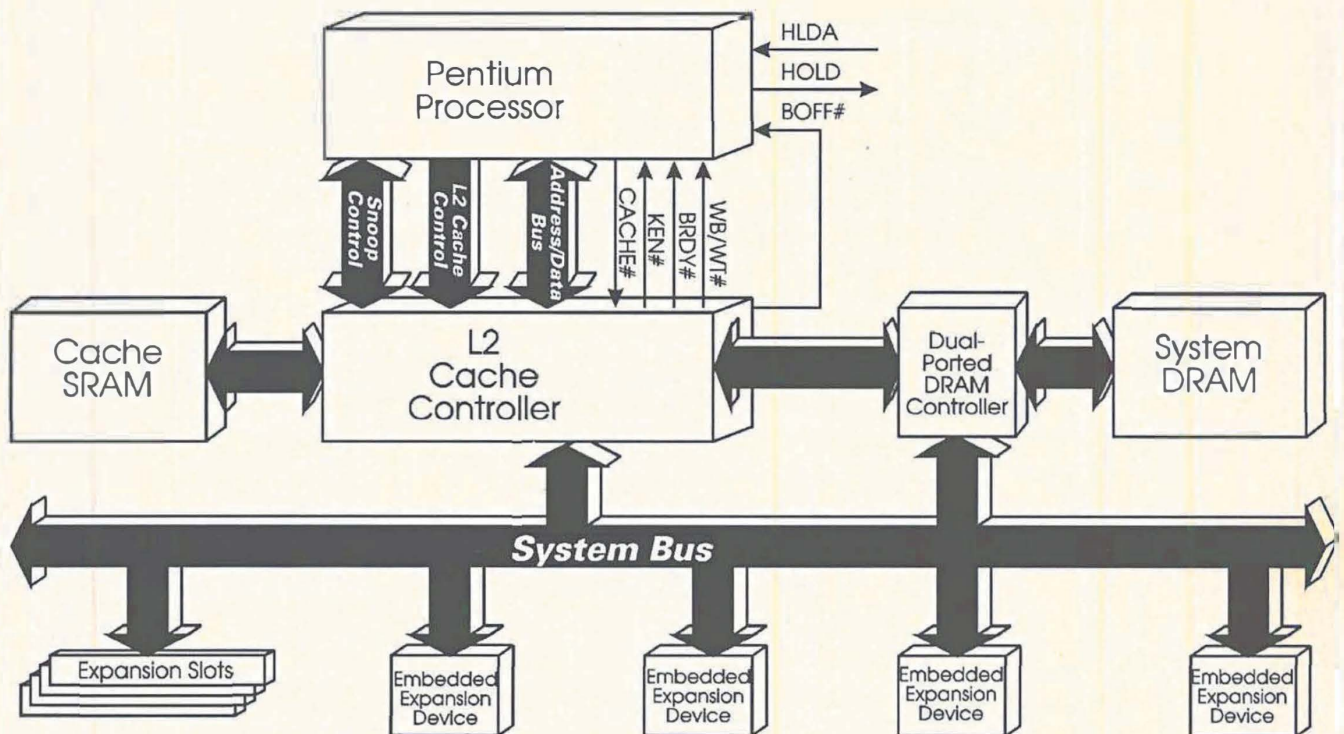


PentiumTM Processor System Architecture

by Don Anderson
and Tom Shanley



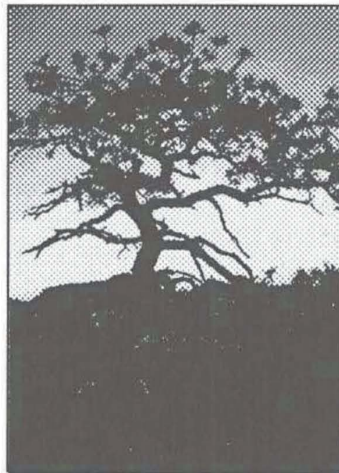
- The ONLY book that explains Intel's PentiumTM processor step by step from a *system* point of view
- Used by hardware and software engineers at IBM, Intel, Compaq and Dell
- Used by Intel Technical Support

**PC SYSTEM
ARCHITECTURE
SERIES • VOLUME**

5

PentiumTM Processor System Architecture

by
DON ANDERSON
AND
TOM SHANLEY



MINDSHARE PRESS

This publication contains information protected by copyright. No part of this book may be copied or reproduced in any form without prior written consent from MindShare, Inc.

**©1993 MindShare, Inc.
2202 Buttercup Dr.
Richardson, Tx 75082
(214)231-2216
fax (214)783-4715**

**All rights reserved. Printed in U.S.A.
First Edition
ISBN 1-881609-07-3**

Pentium is a trademark of Intel Corporation. Other product names mentioned in this material may be trademarks and/or registered trademarks of their respective companies.

Every effort has been made to supply complete and accurate information. However, MindShare, Inc. shall not be liable for any technical or editorial errors or omissions; nor for incidental or consequential damages resulting from the use of this publication.

Distributed by
Computer Literacy Bookshops Inc.
P.O. Box 641897
San Jose, CA 95164 USA

Phone (408) 435-0744
Fax (408) 435-1823
Email info@clbooks.com

Overview of Cache Operation and Cache Types

The solution chosen by most manufacturers of fast 80386-based PCs (20Mhz and faster) was the inclusion of an external cache. Implementation of a cache memory subsystem is an attempt to achieve acceptable system cost and a high percentage of zero wait state bus cycles when accessing memory.

A cache memory subsystem is implemented by populating system board memory with relatively slow access, inexpensive DRAM, while also incorporating a relatively small amount of high cost, fast access SRAM into the system. The SRAM memory is referred to as cache memory and a cache memory controller maintains copies of frequently accessed information (previously read from DRAM memory) within the cache.

Refer to Figure 3 - 1. The cache controller keeps track of the information it has copied into the cache memory. When the processor initiates a memory read bus cycle, the cache controller determines if it has a copy of the requested information in cache memory. If a copy is present, it immediately reads the information from the cache, sends it back to the processor over the processor's data bus, and asserts the processor's ready signal. This is referred to as a read hit. The access can be completed in zero wait states because the information is fetched from the fast access cache SRAM.

Pentium Processor System Architecture

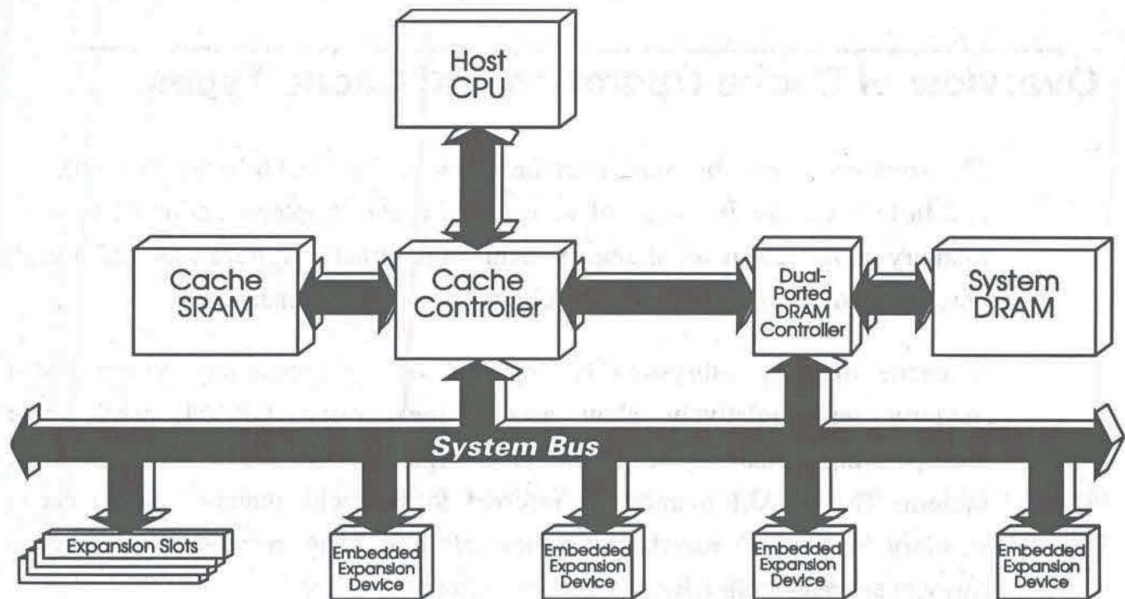


Figure 3-1. The Cache Memory Concept

If, on the other hand, the cache controller determines that it does not have a copy of the requested information in its cache, the information must be read from DRAM memory. This is known as a read miss and results in wait states (due to the slow access time of the DRAM memory). The requested information is sent from DRAM back to the processor to fulfill the request. In addition, the information is also copied into the cache memory by the cache controller. The cache controller updates its directory to track the information stored in cache memory.

On the surface, it may not be evident how this can speed up the processor's memory accesses. However, most programs contain code loops that are executed many times and data structures that are accessed repetitively. A loop is a series of instructions that must be executed a number of times in succession to produce the desired results.

As an example, spreadsheet applications typically use a program loop to recalculate multiple spreadsheet cells. Depending on the number of calculations to be made during a recalc, this program loop may have to be executed hundreds or even thousands of times.

An Historic Retrospective

Assume that the cache memory is empty: it contains no copies of information from DRAM memory. The first time the program loop is executed, the following series of events takes place:

1. The processor initiates a memory read bus cycle to fetch the first instruction from memory.
2. The cache memory controller uses the memory address to determine if a copy of the requested information is already in the cache memory. Since the cache is initially empty, a copy doesn't exist and a read miss occurs.
3. The cache memory controller initiates a memory read bus cycle to fetch the requested information (the instruction) from DRAM memory. This will incur one or more wait states, depending on the access time of the DRAM.
4. The information from DRAM memory (the instruction) is sent back to the processor. The cache controller also copies it into the cache memory and updates its directory to reflect the presence of the new information. No performance advantage was realized during this memory read because the information had to be read from slow DRAM memory.
5. Upon completion of the first instruction, the processor's prefetcher initiates a series of memory read bus cycles to fetch the remaining instructions in the program loop. If the cache is sufficiently large, all instructions in the program loop can become resident in cache memory.
6. The last instruction in the program loop is an instruction to jump to the beginning of the loop and start over again. The processor starts fetching and executing instructions at the memory address of the first instruction in the program loop again.
7. When the processor initiates the memory read bus cycle to fetch the first instruction again, the cache controller checks its cache directory and detects the presence of the requested information in

Pentium Processor System Architecture

cache memory. It reads the information from SRAM and sends it back to the processor with zero wait states incurred.

The processor is thus able to fetch and begin executing the instruction faster than it was able to the first time through the loop. This proves true for the remaining instructions in the program loop as well. The second and subsequent times through the program loop, the processor incurs no wait states in fetching the instructions, so the program runs substantially faster.

Principles of Locality

The term used to explain the characteristics of programs that run in relatively small loops in consecutive memory locations is locality of reference. As described in the previous example, cache subsystems work because most programs that run on PCs require the same information from consecutive memory locations over and over again. The locality of reference principle is comprised of two components: temporal locality and spatial locality.

Temporal Locality

Since programs run in loops, the same instructions must be fetched from memory on a frequent and continuing basis, meaning that programs tend to use the most recently used information again. The longer its been since information in the cache has been used, the less likely it is to be used again. This is known as the principle of temporal locality. How well a cache subsystem performs is tied directly to how often the software requests the same information from memory. If a program were intentionally designed to never access information more than one time, the cache subsystem would provide absolutely no performance increase.

Pentium Processor System Architecture

Since all information destined for the internal L1 cache must pass through the external L2 cache, the advantage of the L2 cache may not be immediately apparent. If the L2 cache were the same size as the L1 cache, there would be no advantage. If, however, the L2 cache is substantially larger than the L1 cache, the advantage becomes clear. L2 caches are usually much larger (64KB-512 KB) than L1 caches.

L2 caches improve overall performance because the L1 cache can get information from the L2 cache quickly on most internal read misses. Furthermore, to take full advantage of the i486 burst cycles, the typical L2 cache can accommodate the fastest possible burst transfer.

Cache Coherency with an L1 and L2 Cache

Refer to figure 3 - 5. When an L2 cache is used in conjunction with the i486, care must be taken to ensure that cache consistency is maintained between the L1 and L2 caches and memory. To help maintain cache consistency, the i486 L1 cache implements a write-through policy to ensure that the external cache is notified of all write operations to the i486 L1 cache.

The L2 cache can be designed to utilize either a write-through or a write-back policy. These cache types maintain cache coherency with memory as described earlier. However, if both the L2 cache and L1 cache have copies of the line that is being written to by another bus master, additional steps must be taken by both the L2 and L1 caches.

1. When another bus master performs a write operation to memory, the L2 cache snoops the line address that is being written to in memory. If the L2 cache detects a snoop hit, then the information in the L2 cache will be stale when the write operation completes. To ensure that the processor does not subsequently receive stale data from the L2 cache, it will invalidate the directory entry. The next access to the line will then result in a cache miss and an access to memory.

An Historic Retrospective

2. Since the data that is being updated was found in the L2 cache during the snoop, it is also possible that the L1 cache has a copy of the same line. If no further action is taken, the L1 cache will contain stale data. To prevent this type of cache inconsistency, the L2 cache must pass the snoop address back to the i486. It can then determine if a copy of the target line is also contained in its L1 cache. If a cache hit occurs, the i486 will invalidate the directory entry to ensure that stale data isn't contained in the L1 cache.

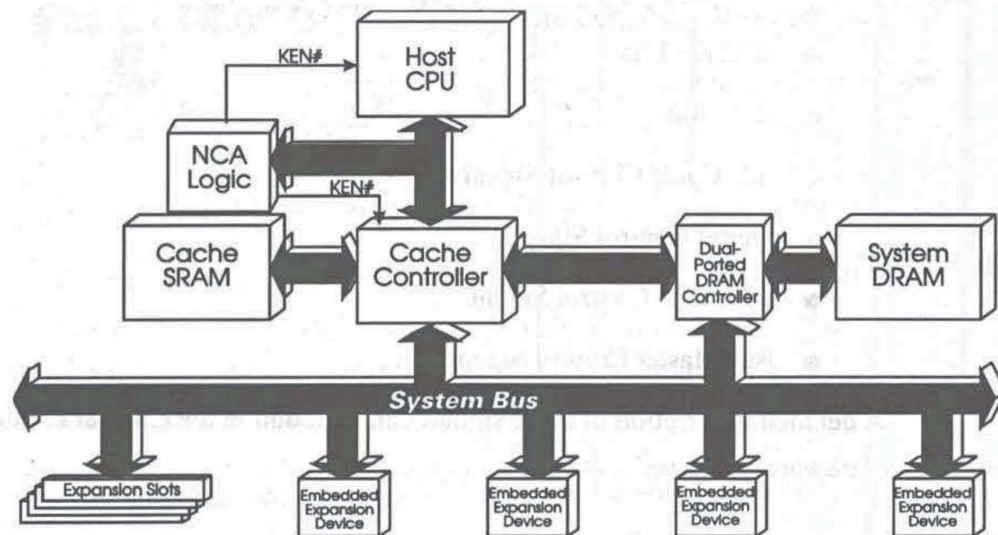


Figure 3-5. An i486 System Incorporating an L2 Cache

Pentium Processor Cache Overview

The Pentium processor improves performance over the i486 in a variety of ways, including the implementation of separate data and code caches. The line size for each of these caches is 32 bytes and the data bus is eight bytes, or 64 bits in width. A burst of four consecutive transfers is therefore required to fill the cache line. As with the i486, implementing the Pentium processor with an L2 cache provides the highest performance.

Pentium Processor System Architecture

Due to its write-back policy, the Pentium processor's L1 data cache introduces additional complexity into the cache consistency logic. The data cache can also use a write-through or a write-back policy on a line-by-line basis. The instruction, or code, cache is read-only, so it doesn't require a write policy.

Figure 3 - 6 illustrates the relationship between the Pentium processor, the L2 cache, the Non-Cacheable Access (NCA) logic, system memory and devices connected to an expansion bus (ISA, EISA, or Micro Channel). Six major signal groups comprise the processor's interface to the external environment. These signal groups are:

- Address Bus
- Data Bus
- Bus Cycle Control Signals
- Snoop Control Signals
- L2 Cache Control Signals
- Bus Master Control Signals

A detailed description of these signals can be found in the Chapter entitled, "The Hardware Interface".

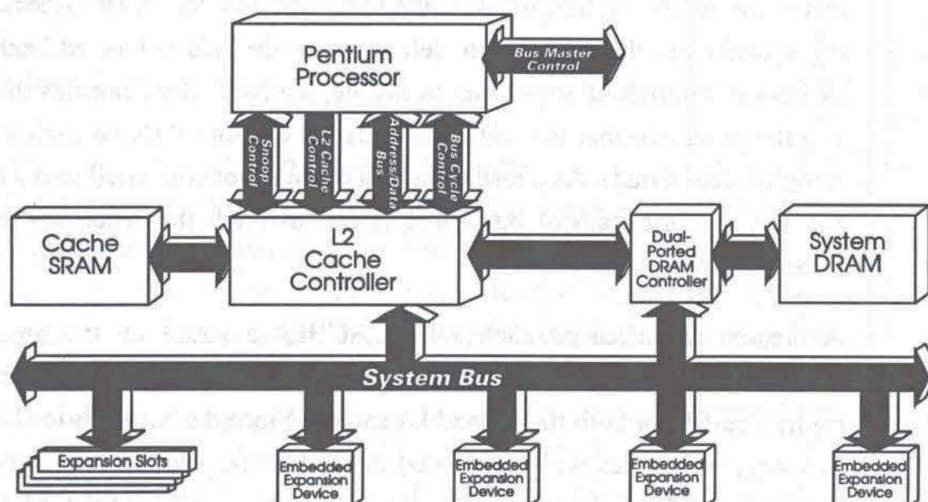


Figure 3-6. The Pentium Processor's Relationship To Its External Environment

When the Pentium processor's data cache utilizes a write-through policy, its operation with an L2 cache is conceptually identical to that of an i486. The following descriptions assume that both the Pentium processor and its L2 cache are using write-back policies.

Handling of Memory Reads Initiated by the Pentium Processor

When either the prefetcher or one of the execution units requests information from an internal cache, the request is immediately fulfilled if a cache hit occurs. If the information is not in the internal cache, the Pentium processor must perform a cache line-fill to read the target line from external memory.

The L2 cache detects the memory read bus cycle and interrogates its directory to determine if it has a copy of the requested information. If it's a hit on the L2 cache, the L2 cache notifies the Pentium processor that the address is cacheable (KEN# is asserted). Note that if the target line is found in the L2 cache, the line must be cacheable. The L2 cache then accesses its cache SRAM memory, satisfying the cache line-fill in a burst of four consecutive 64-bit transfers.