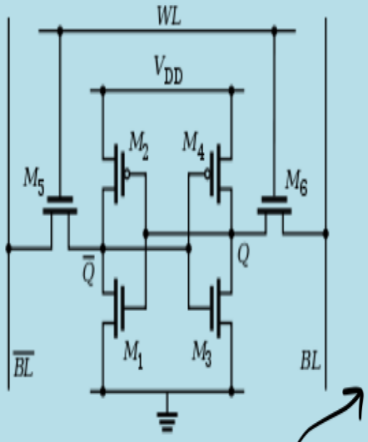
Types of RAMs

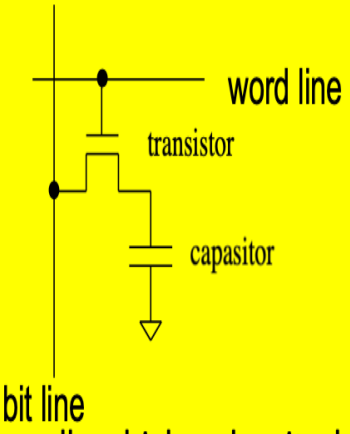
- **SRAM Cell** **static**



number of memory cells

- Larger cell ⇒ lower density, higher cost/bit
- No refresh required
- Simple read ⇒ faster access
- Standard IC process ⇒ natural for integration with logic

- **DRAM Cell** **dynamic**



- Smaller cell ⇒ higher density, lower cost/bit
- Needs periodic refresh, and refresh after read
- Complex read ⇒ longer access time
- Special IC process ⇒ difficult to integrate with logic circuits

SRAM vs DRAM Summary

	Tran. per bit	Access time	Cost	Applications
SRAM	6	1X	100x	cache memories
DRAM	1	10X	1X	Main memories,

CACHE MEMORY

The most widely used memory design for high-performance CPUs implements DRAMs for main memory along with a small amount (compared to the size of main memory) of SRAM for cache memory. This takes advantage of the speed of SRAM and the high density and cheapness of DRAM. As mentioned earlier, to implement the entire memory of the computer with SRAM is too expensive and to use all DRAM degrades performance. Cache memory is placed between the CPU and main memory. See Figure 22-4.

CACHE MEMORY

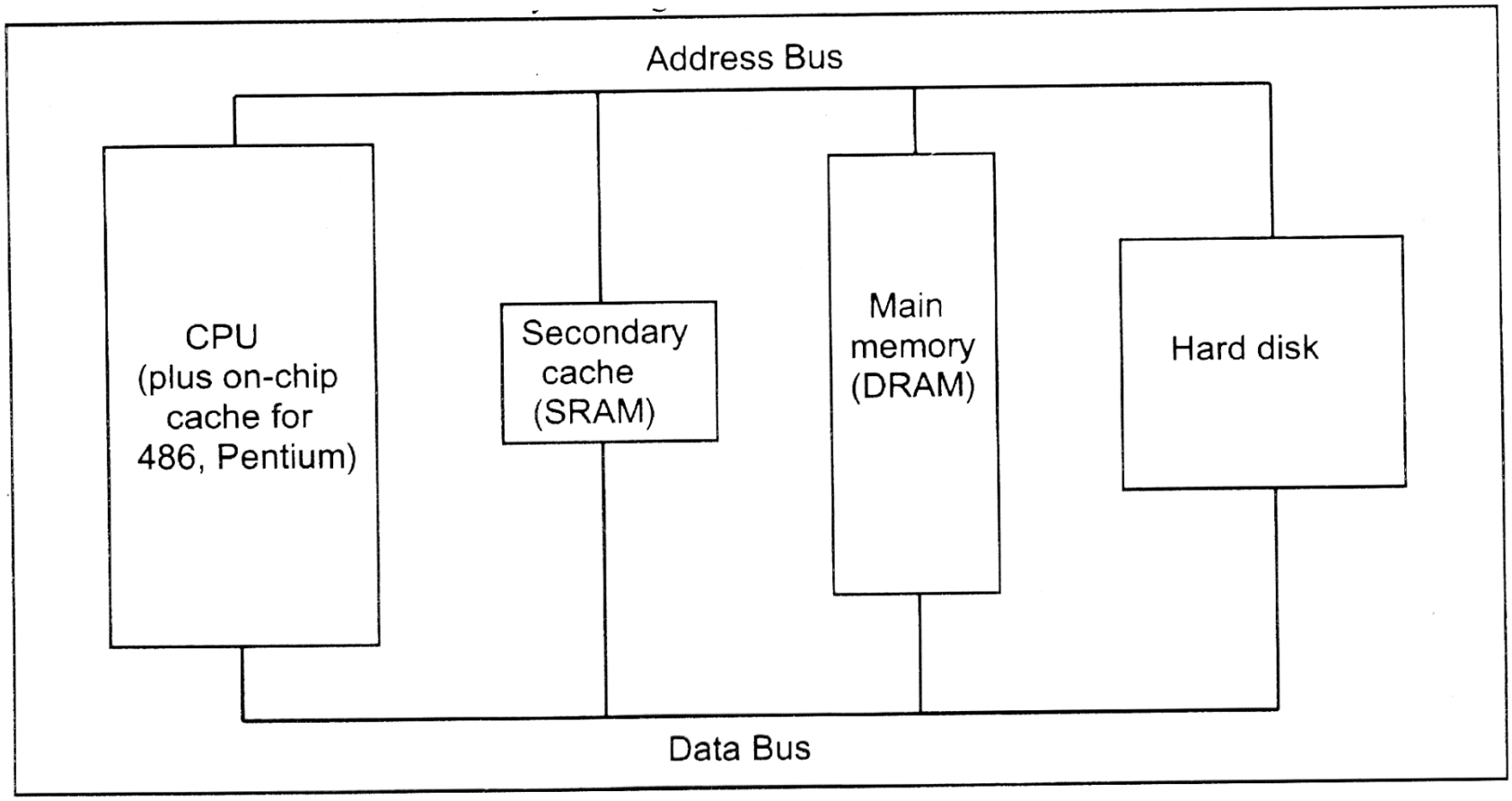


Figure 22-4. CPU and Its Relation to Various Memories

CACHE MEMORY

When the CPU initiates a memory access, it first asks cache for the information (data or code). If the requested data is there, it is provided to the CPU with zero wait states, but if the data is not in cache, the memory controller circuitry will transfer the data from main memory to the CPU while giving a copy of it to cache memory. In other words, at any given time the cache controller has knowledge of which information (code or data) is kept in cache; therefore, upon request for a given piece of code or data by the CPU the address issued by the CPU is compared with the addresses of data kept by the cache controller. If they match (hit) they are presented to the CPU with zero WS, but if the needed information is not in cache (miss) the cache controller along with the memory controller will fetch the data and present it to the CPU in addition to keeping a copy of it in cache for future reference. The reason a copy of data (or code) fetched from main memory is kept in the cache is to allow any subsequent request for the same information to result in a hit and provide it to the CPU with zero wait states. If the requested data is available in cache memory, it is called a *hit*; otherwise, if the data must be brought in from main memory, it is a *miss*.

CACHE MEMORY

- For example, a segment of program that implements a loop operation could be fetched, executed and placed in the cache.
- We find that the first execution of the loop references code held in the slow main program memory. During this access, the routine is copied into the cache part of memory. When the instructions of the loop are repeated, the up reaccesses the routine by using the instructions held in the cache instead of refetching them from main memory.

$$\text{Hit rate} = \frac{\text{no. of hits}}{\text{no. of bus cycles}} \times 100\%$$

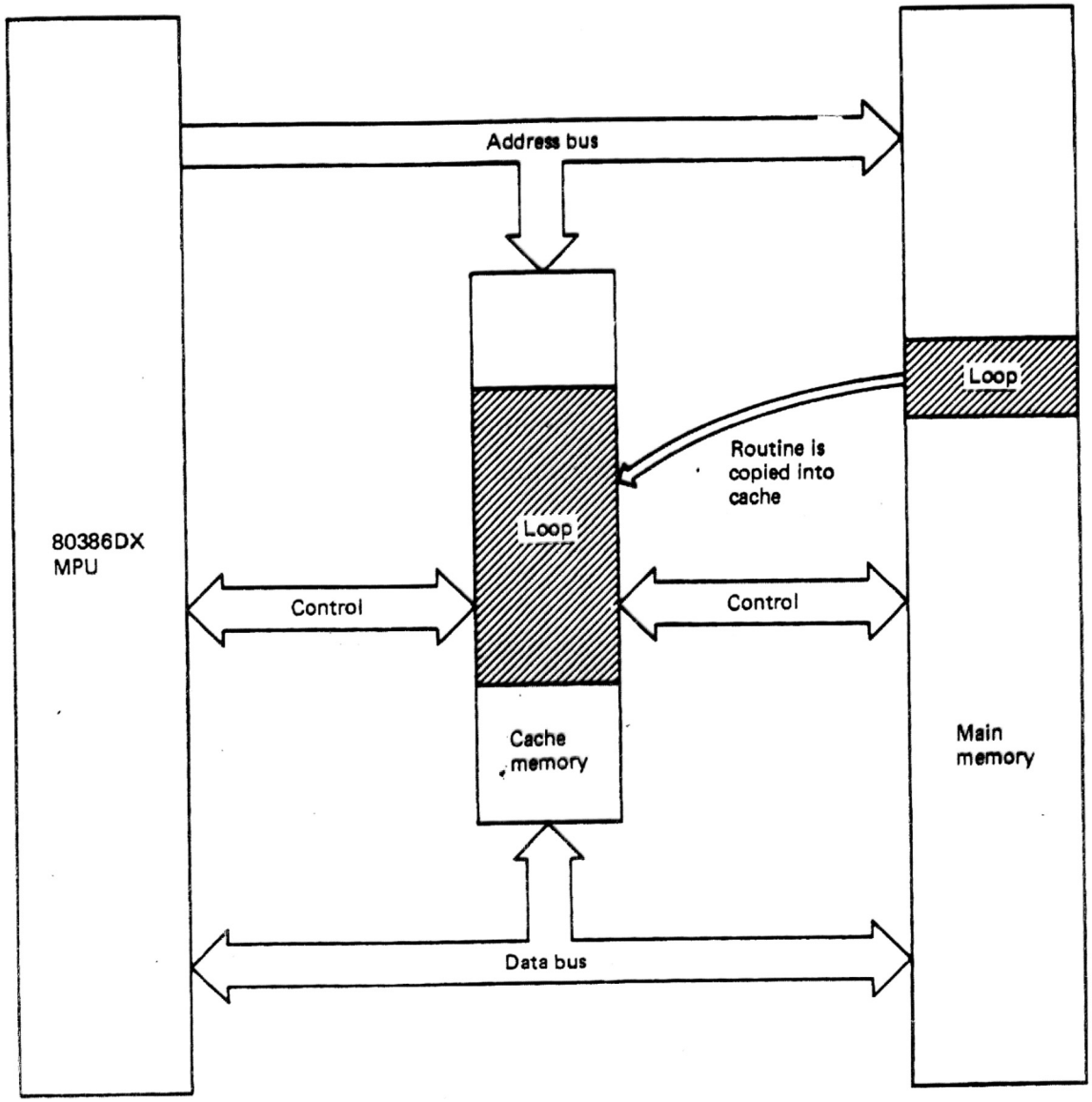


Figure 10.41 Caching a loop routine.

Cache Operation: Example 1

A processor has an on-chip cache that runs in one cycle of 5 ns (200 MHz). All accesses that are not found in the cache are found in main memory, with an access time of 60 ns. The *hit rate* of the cache [the probability that it contains the words we want] is 85%. The *miss rate* , therefore, is 15%.

Overall memory access time = 0. 85 * 5 *ns*

+ 0. 15 * 60 *ns*

= 13. 25 *ns*

This calculation may be extended to multi-level caches by generating the probabilities of each cache or memory component.

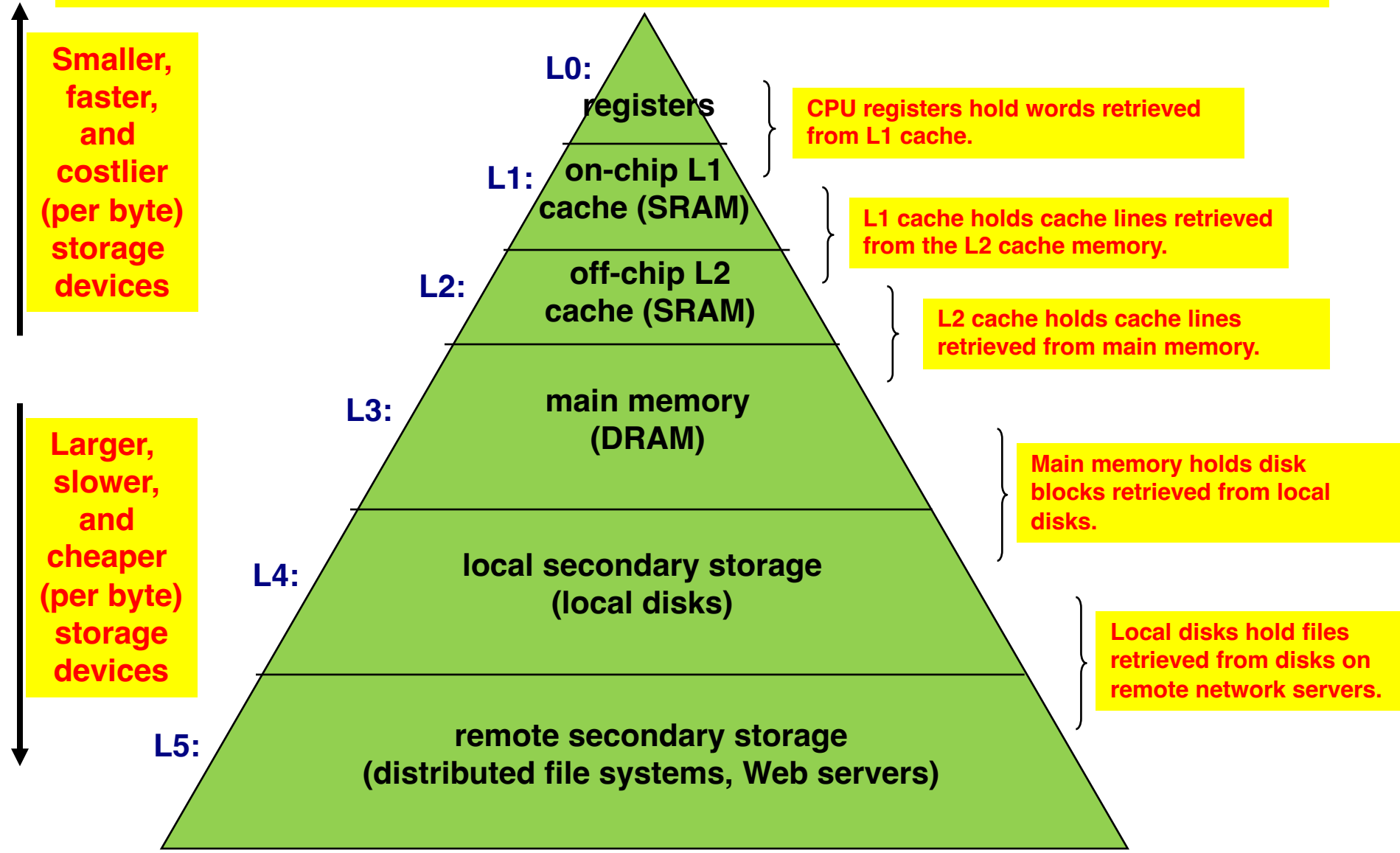
Cache Performance: Example 2

A processor has an on-chip cache that runs in one cycle of 5 ns (200 MHz). This cache has a hit rate of 90%. A second-level [motherboard] cache runs in three cycles and has a hit rate of 70% **of all accesses that are not found in the on-chip cache**. All accesses that are not found in either cache are found in main memory, with an access time of 60 ns.

Overall memory access time = 0. 90 * 5 *ns*
+ 0. 07 *15 *ns*
+ 0. 03 * 60 *ns*
= 7. 35 *ns*

3 cycles Motherboard x one cycle time (3x5ns)= 15ns x hit ratio

An Example Memory Hierarchy



Caches

- Cache: A smaller, faster storage device that acts as a staging area for a subset of the data in a larger, slower device.
- Fundamental idea of a memory hierarchy:
  - For each k, the faster, smaller device at level k serves as a cache for the larger, slower device at level k+1.
- Why do memory hierarchies work?
  - Programs tend to access the data at level k more often than they access the data at level k+1.
  - Thus, the storage at level k+1 can be slower, and thus larger and cheaper per bit.
  - Net effect: A large pool of memory that costs as much as the cheap storage near the bottom, but that serves data to programs at the rate of the fast storage near the top.

CACHE ORGANIZATION

There are three types of cache organization:

1- fully associative

2- direct mapped

3- set associative