Chapter 2
Memory Hierarchy Design

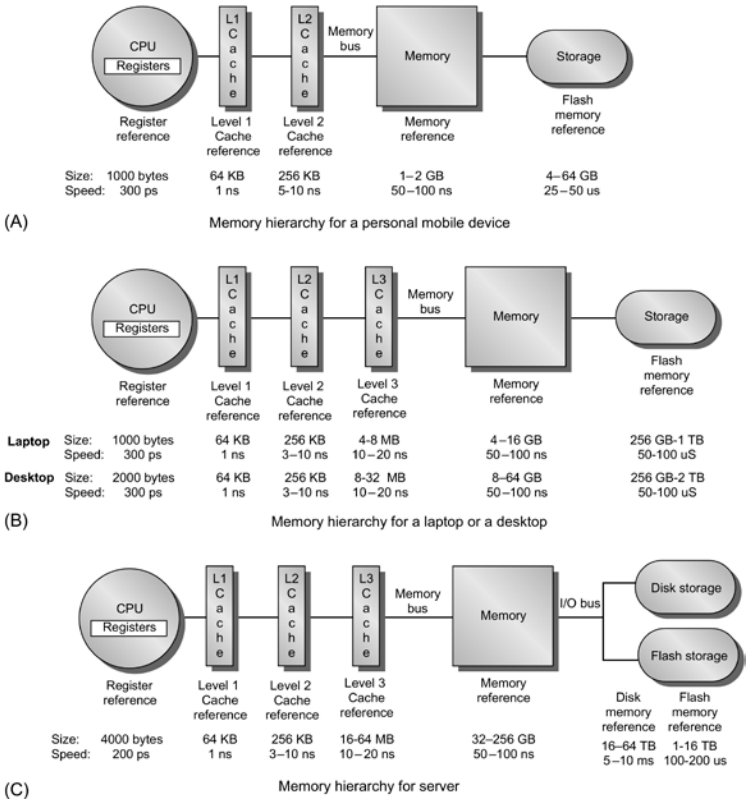

Figure 2.1 The levels in a typical memory hierarchy in a personal mobile device (PMD), such as a cell phone or tablet (A), in a laptop or desktop computer (B), and in a server (C). As we move farther away from the processor, the memory in the level below becomes slower and larger. Note that the time units change by a factor of 10^9 from picoseconds to milliseconds in the case of magnetic disks and that the size units change by a factor of 10^{10} from thousands of bytes to tens of terabytes. If we were to add warehouse-sized computers, as opposed to just servers, the capacity scale would increase by three to six orders of magnitude. Solid-state drives (SSDs) composed of Flash are used exclusively in PMDs, and heavily in both laptops and desktops. In many desktops, the primary storage system is SSD, and expansion disks are primarily hard disk drives (HDDs). Likewise, many servers mix SSDs and HDDs.

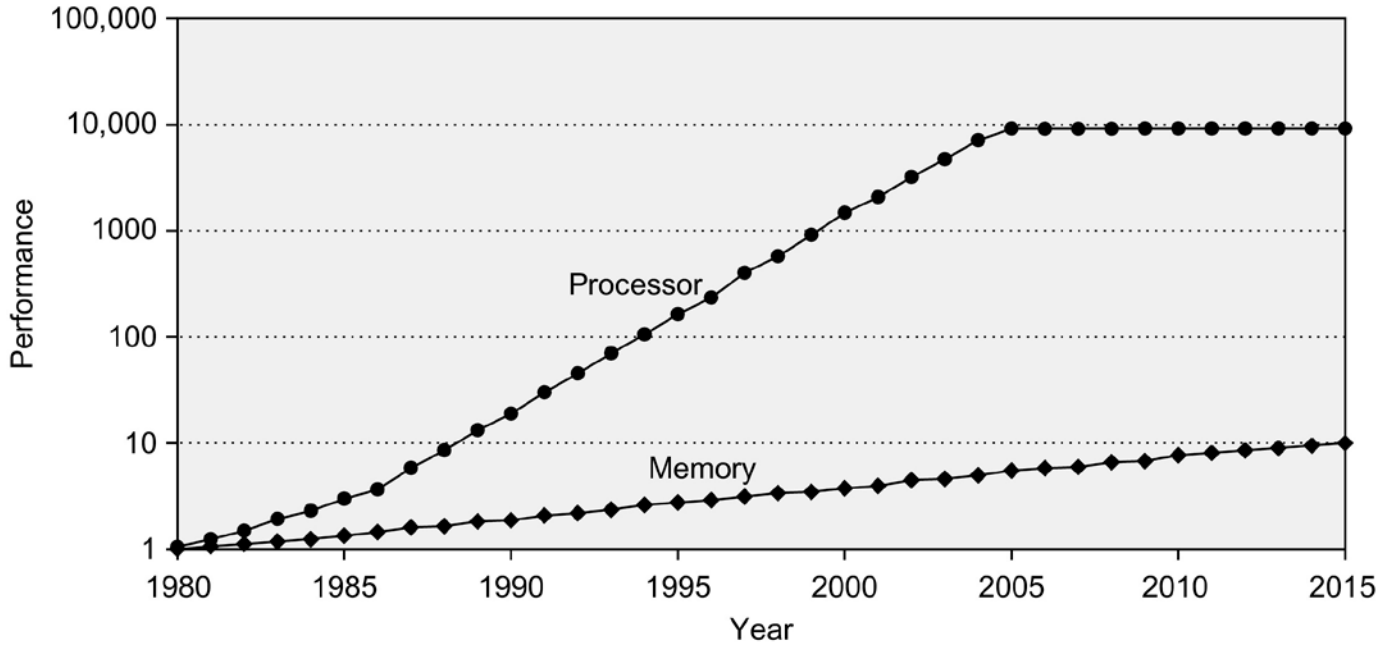

Figure 2.2 Starting with 1980 performance as a baseline, the gap in performance, measured as the difference in the time between processor memory requests (for a single processor or core) and the latency of a DRAM access, is plotted over time. In mid-2017, AMD, Intel and Nvidia all announced chip sets using versions of HBM technology. Note that the vertical axis must be on a logarithmic scale to record the size of the processor-DRAM performance gap. The memory baseline is 64 KiB DRAM in 1980, with a 1.07 per year performance improvement in latency (see Figure 2.4 on page 88). The processor line assumes a 1.25 improvement per year until 1986, a 1.52 improvement until 2000, a 1.20 improvement between 2000 and 2005, and only small improvements in processor performance (on a per-core basis) between 2005 and 2015. As you can see, until 2010 memory access times in DRAM improved slowly but consistently; since 2010 the improvement in access time has reduced, as compared with the earlier periods, although there have been continued improvements in bandwidth. See Figure 1.1 in Chapter 1 for more information.

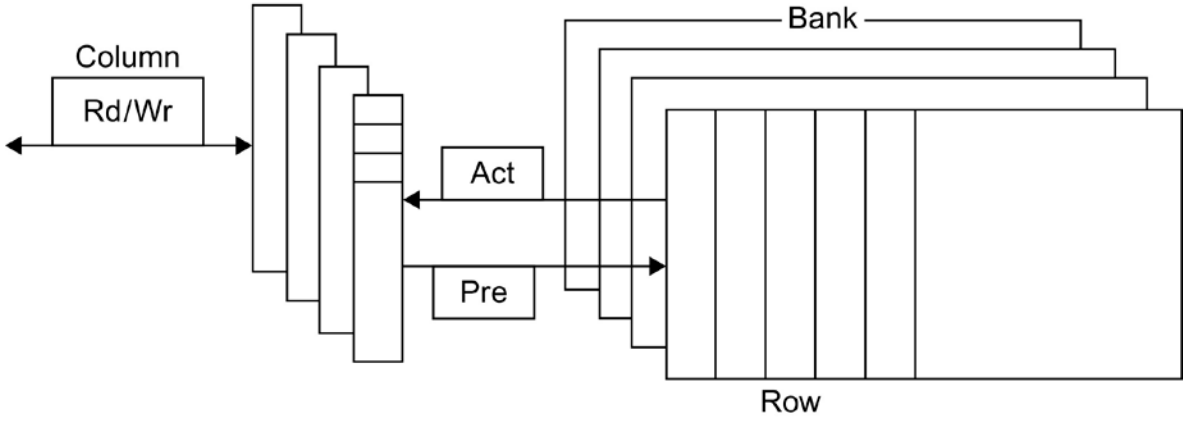

Figure 2.3 I**nternal organization of a DRAM.** Modern DRAMs are organized in banks, up to 16 for DDR4. Each bank consists of a series of rows. Sending an ACT (Activate) command opens a bank and a row and loads the row into a row buffer. When the row is in the buffer, it can be transferred by successive column addresses at whatever the width of the DRAM is (typically 4, 8, or 16 bits in DDR4) or by specifying a block transfer and the starting address. The Precharge commend (PRE) closes the bank and row and readies it for a new access. Each command, as well as block transfers, are synchronized with a clock. See the next section discussing SDRAM. The row and column signals are sometimes called RAS and CAS, based on the original names of the signals.

			Best case access time (no precharge)			Precharge needed
Production year	Chip size	DRAM type	RAS time (ns)	CAS time (ns)	Total (ns)	Total (ns)
2000	256M bit	DDR1	21	21	42	63
2002	512M bit	DDR1	15	15	30	45
2004	1G bit	DDR2	15	15	30	45
2006	2G bit	DDR2	10	10	20	30
2010	4G bit	DDR3	13	13	26	39
2016	8G bit	DDR4	13	13	26	39

Figure 2.4 Capacity and access times for DDR SDRAMs by year of production. Access time is for a random memory word and assumes a new row must be opened. If the row is in a different bank, we assume the bank is precharged; if the row is not open, then a precharge is required, and the access time is longer. As the number of banks has increased, the ability to hide the precharge time has also increased. DDR4 SDRAMs were initially expected in 2014, but did not begin production until early 2016.

Standard	I/O clock rate	M transfers/s	DRAM name	MiB/s/DIMM	DIMM name
DDR1	133	266	DDR266	2128	PC2100
DDR1	150	300	DDR300	2400	PC2400
DDR1	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1333	2666	DDR4-2666	21,300	PC21300

Figure 2.5 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2016. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. DDR4 saw significant first use in 2016.

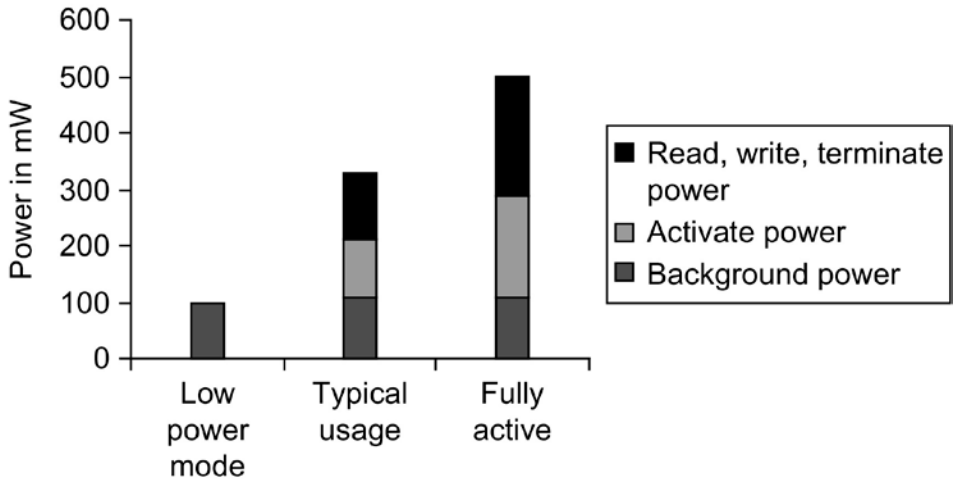

Figure 2.6 Power consumption for a DDR3 SDRAM operating under three conditions: low-power (shutdown) mode, typical system mode (DRAM is active 30% of the time for reads and 15% for writes), and fully active mode, where the DRAM is continuously reading or writing. Reads and writes assume bursts of eight transfers. These data are based on a Micron 1.5V 2GB DDR3-1066, although similar savings occur in DDR4 SDRAMs.

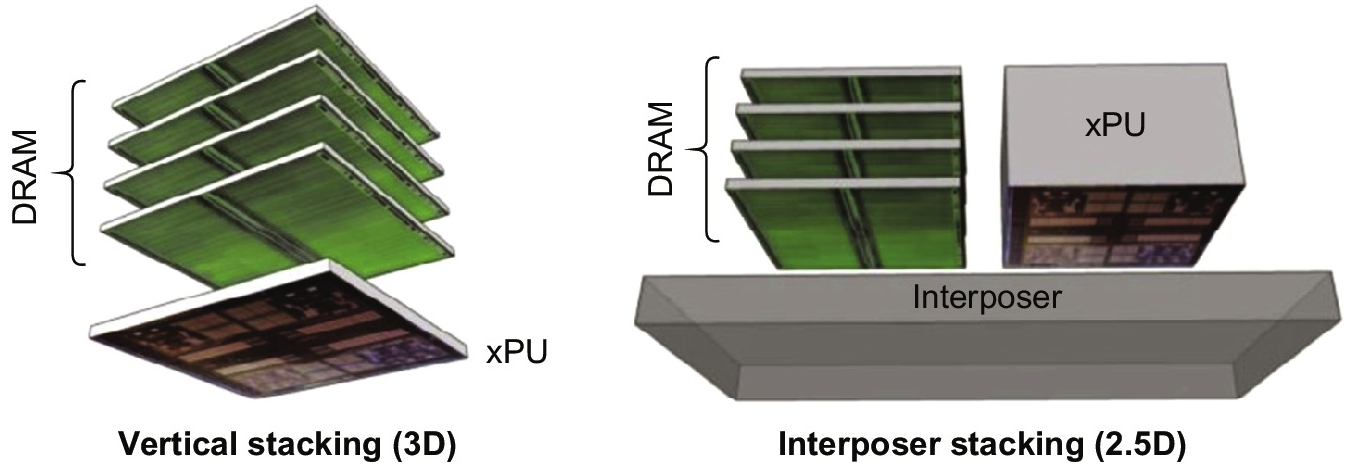

Figure 2.7 Two forms of die stacking. The 2.5D form is available now. 3D stacking is under development and faces heat management challenges due to the CPU.

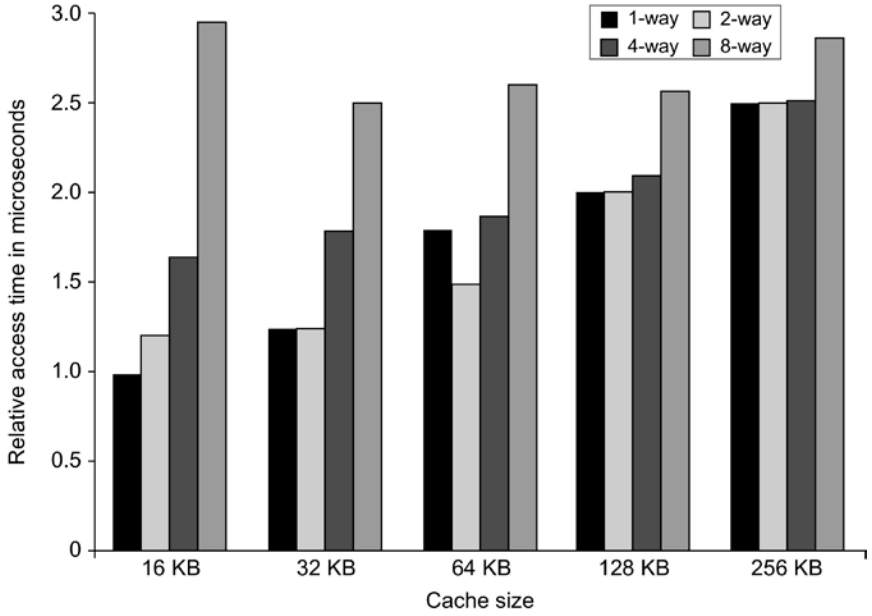

Figure 2.8 Relative access times generally increase as cache size and associativity are increased. These data come from the CACTI model 6.5 by Tarjan et al. (2005). The data assume typical embedded SRAM technology, a single bank, and 64-byte blocks. The assumptions about cache layout and the complex trade-offs between interconnect delays (that depend on the size of a cache block being accessed) and the cost of tag checks and multiplexing lead to results that are occasionally surprising, such as the lower access time for a 64 KiB with two-way set associativity versus direct mapping. Similarly, the results with eight-way set associativity generate unusual behavior as cache size is increased. Because such observations are highly dependent on technology and detailed design assumptions, tools such as CACTI serve to reduce the search space. These results are relative; nonetheless, they are likely to shift as we move to more recent and denser semiconductor technologies.

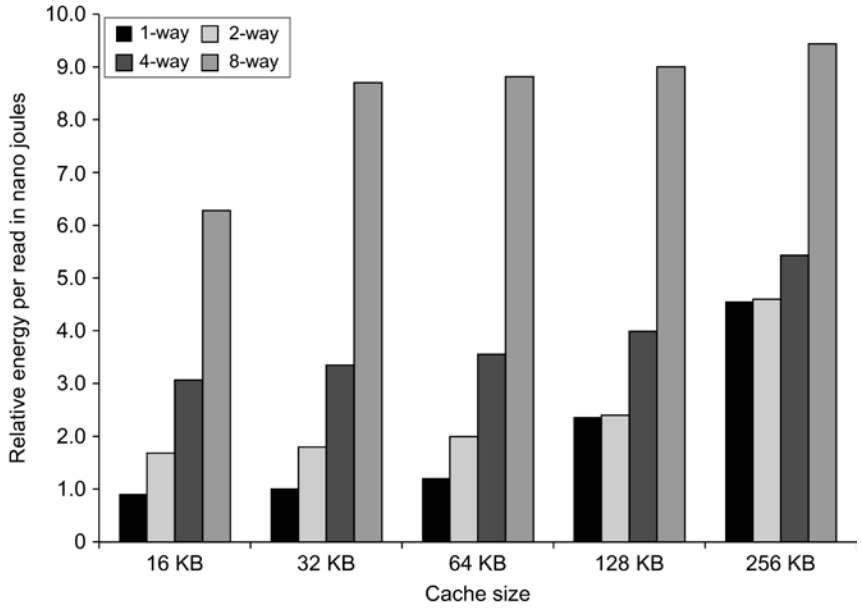

Figure 2.9 Energy consumption per read increases as cache size and associativity are increased. As in the previous figure, CACTI is used for the modeling with the same technology parameters. The large penalty for eight-way set associative caches is due to the cost of reading out eight tags and the corresponding data in parallel.

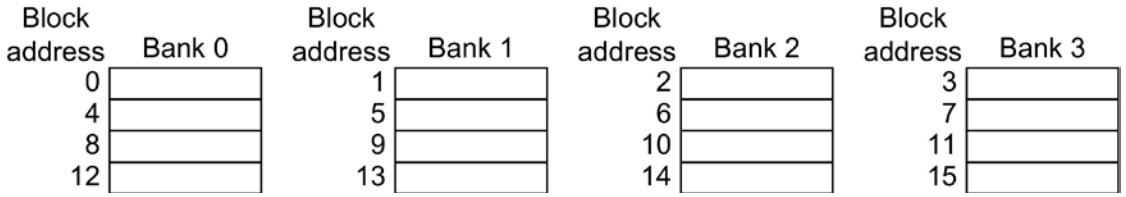

Figure 2.10 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per block, each of these addresses would be multiplied by 64 to get byte addressing.

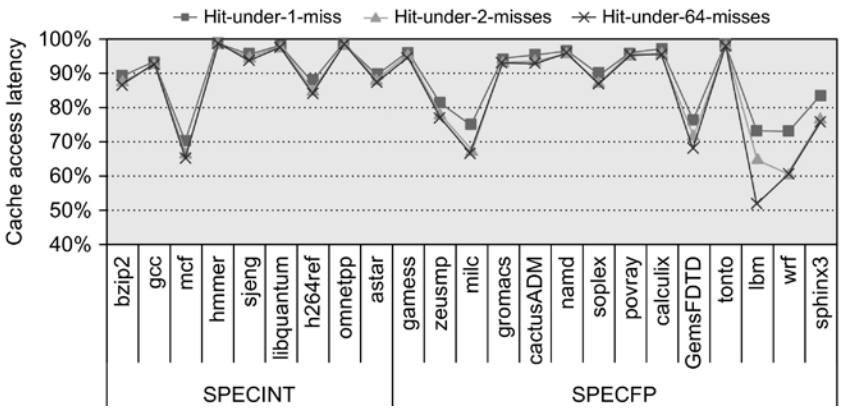

Figure 2.11 The effectiveness of a nonblocking cache is evaluated by allowing 1, 2, or 64 hits under a cache miss with 9 SPECINT (on the left) and 9 SPECFP (on the right) benchmarks. The data memory system modeled after the Intel i7 consists of a 32 KiB L1 cache with a four-cycle access latency. The L2 cache (shared with instructions) is 256 KiB with a 10-clock cycle access latency. The L3 is 2 MiB and a 36-cycle access latency. All the caches are eight-way set associative and have a 64-byte block size. Allowing one hit under miss reduces the miss penalty by 9% for the integer benchmarks and 12.5% for the floating point. Allowing a second hit improves these results to 10% and 16%, and allowing 64 results in little additional improvement.

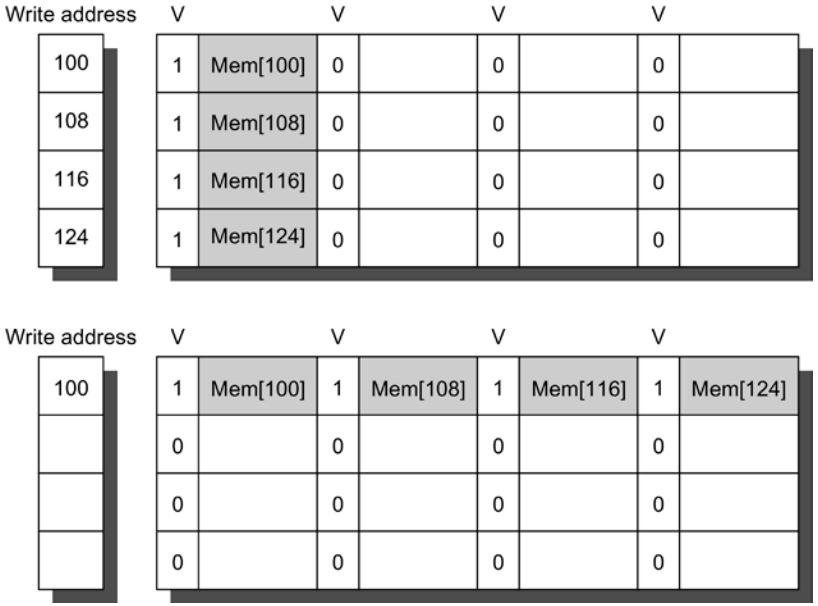

Figure 2.12 In this illustration of write merging, the write buffer on top does not use write merging while the write buffer on the bottom does. The four writes are merged into a single buffer entry with write merging; without it, the buffer is full even though three-fourths of each entry is wasted. The buffer has four entries, and each entry holds four 64-bit words. The address for each entry is on the left, with a valid bit (V) indicating whether the next sequential 8 bytes in this entry are occupied. (Without write merging, the words to the right in the upper part of the figure would be used only for instructions that wrote multiple words at the same time.)

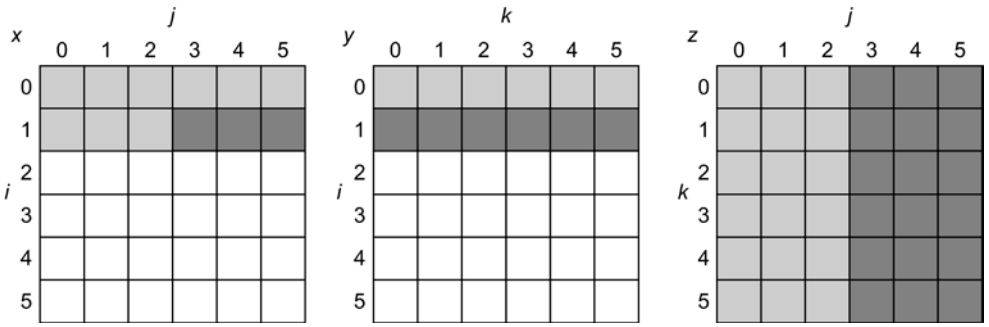

Figure 2.13 A snapshot of the three arrays x, y, and z when *N* = 6 and i = 1. The age of accesses to the array elements is indicated by shade: white means not yet touched, light means older accesses, and dark means newer accesses. The elements of y and z are read repeatedly to calculate new elements of x. The variables i, j, and k are shown along the rows or columns used to access the arrays.

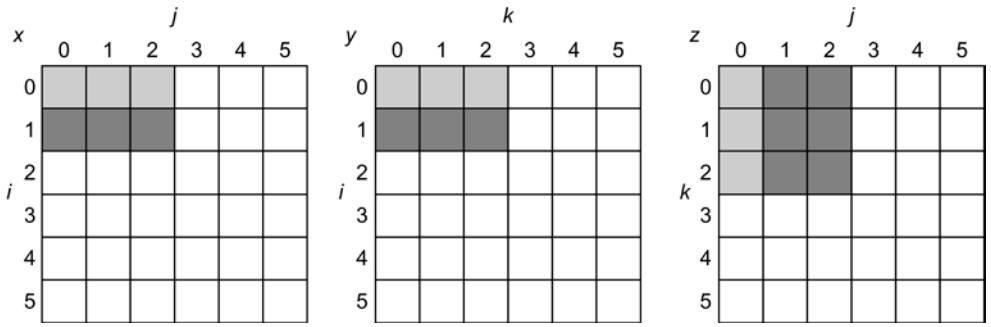

Figure 2.14 The age of accesses to the arrays x, y, and z when *B* = 3. Note that, in contrast to Figure 2.13, a smaller number of elements is accessed.

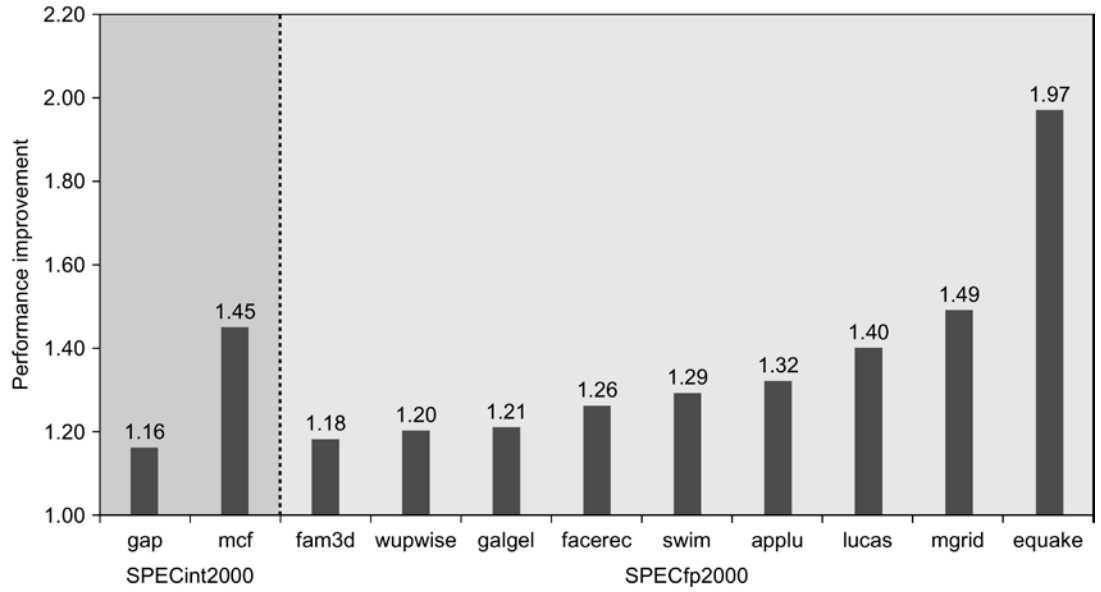

Figure 2.15 Speedup because of hardware prefetching on Intel Pentium 4 with hardware prefetching turned on for 2 of 12 SPECint2000 benchmarks and 9 of 14 SPECfp2000 benchmarks. Only the programs that benefit the most from prefetching are shown; prefetching speeds up the missing 15 SPECCPU benchmarks by less than 15% (Boggs et al., 2004).

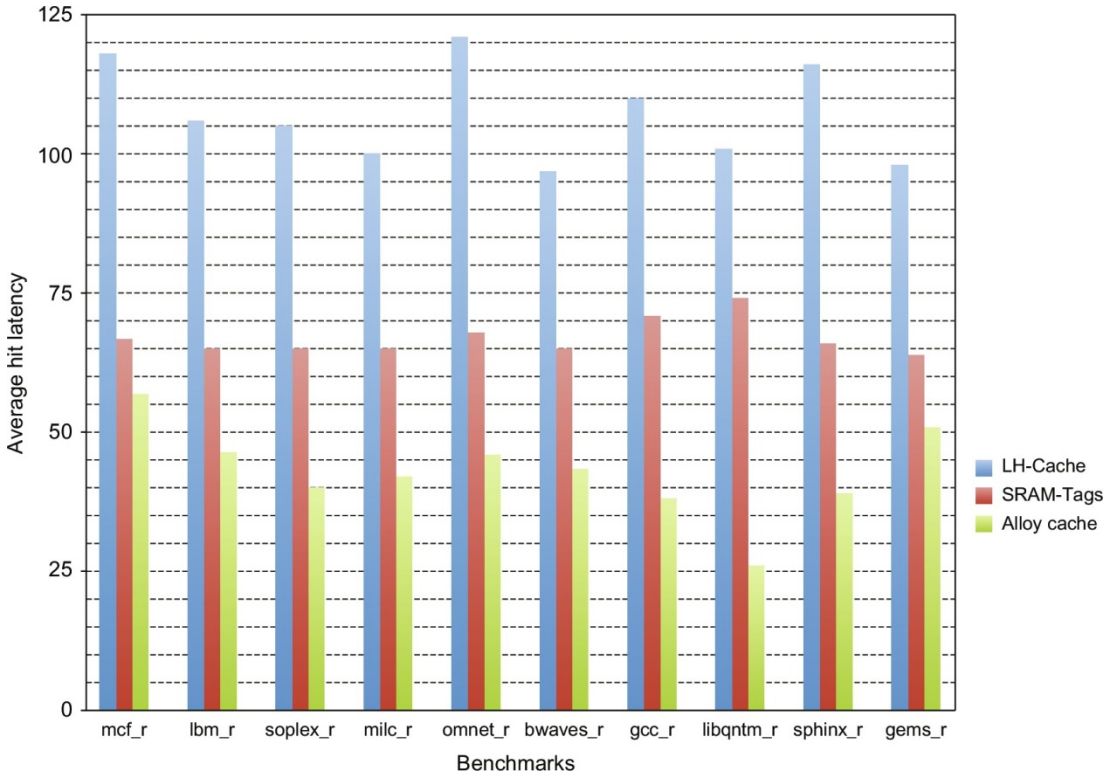

Figure 2.16 Average hit time latency in clock cycles for the L-H scheme, a currently-impractical scheme using SRAM for the tags, and the alloy cache organization. In the SRAM case, we assume the SRAM is accessible in the same time as L3 and that it is checked before L4 is accessed. The average hit latencies are 43 (alloy cache), 67 (SRAM tags), and 107 (L-H). The 10 SPECCPU2006 benchmarks used here are the most memory-intensive ones; each of them would run twice as fast if L3 were perfect.

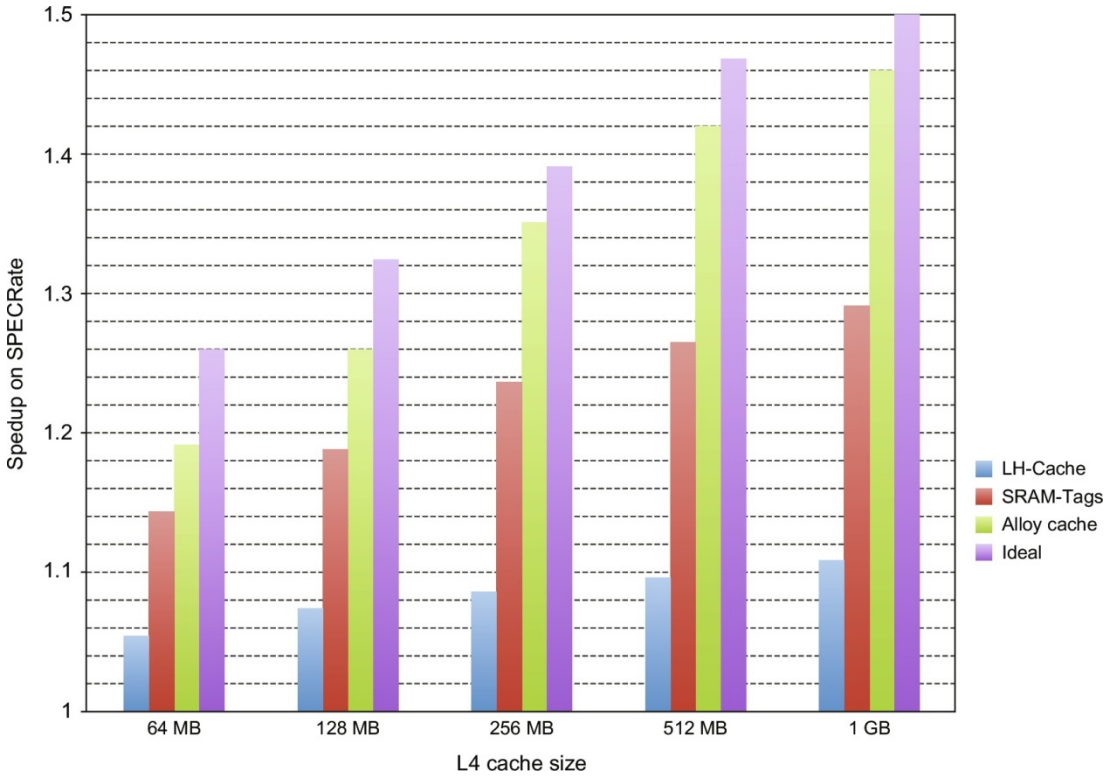

Figure 2.17 Performance speedup running the SPECrate benchmark for the LH scheme, an SRAM tag scheme, and an ideal L4 (Ideal); a speedup of 1 indicates no improvement with the L4 cache, and a speedup of 2 would be achievable if L4 were perfect and took no access time. The 10 memory-intensive benchmarks are used with each benchmark run eight times. The accompanying miss prediction scheme is used. The Ideal case assumes that only the 64-byte block requested in L4 needs to be accessed and transferred and that prediction accuracy for L4 is perfect (i.e., all misses are known at zero cost).

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined & banked caches	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs
HBM as additional level of cache		+/–	–	+	+	3	Depends on new packaging technology. Effects depend heavily on hit rate improvements

Figure 2.18 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, − means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Structure	Size	Organization	Typical miss penalty (clock cycles)
Instruction MicroTLB	10 entries	Fully associative	2
Data MicroTLB	10 entries	Fully associative	2
L2 Unified TLB	512 entries	4-way set associative	20
L1 Instruction cache	8–64 KiB	2-way set associative; 64-byte block	13
L1 Data cache	8–64 KiB	2-way set associative; 64-byte block	13
L2 Unified cache	128 KiB to 2 MiB	16-way set associative; LRU	124

Figure 2.19 The memory hierarchy of the Cortex A53 includes multilevel TLBs and caches. A page map cache keeps track of the location of a physical page for a set of virtual pages; it reduces the L2 TLB miss penalty. The L1 caches are virtually indexed and physically tagged; both the L1 D cache and L2 use a write-back policy defaulting to allocate on write. Replacement policy is LRU approximation in all the caches. Miss penalties to L2 are higher if both a MicroTLB and L1 miss occur. The L2 to main memory bus is 64–128 bits wide, and the miss penalty is larger for the narrow bus.

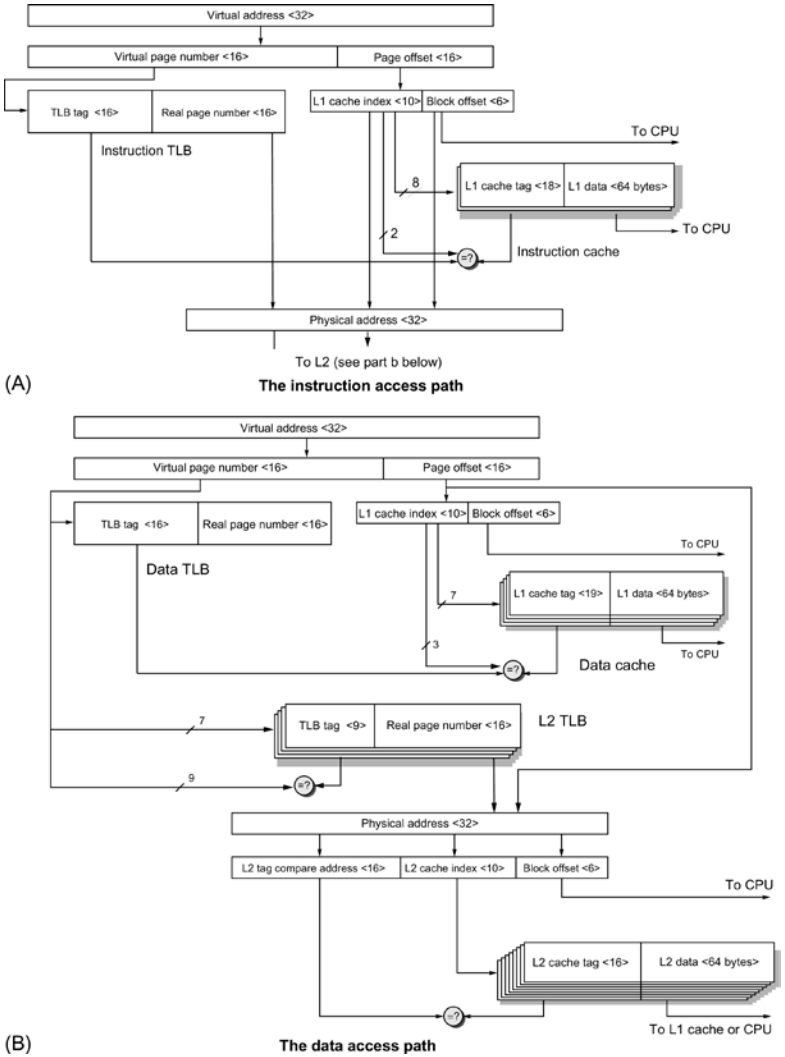

Figure 2.20 The virtual address, physical and data blocks for the ARM Cortex-A53 caches and TLBs, assuming 32-bit addresses. The top half (A) shows the instruction access; the bottom half (B) shows the data access, including L2. The TLB (instruction or data) is fully associative each with 10 entries, using a 64 KiB page in this example. The L1 I-cache is two-way set associative, with 64-byte blocks and 32 KiB capacity; the L1 D-cache is 32 KiB, four-way set associative, and 64-byte blocks. The L2 TLB is 512 entries and four-way set associative. The L2 cache is 16-way set associative with 64-byte blocks and 128 cKiB to 2 MiB capacity; a 1 MiB L2 is shown. This figure doesn't show the valid bits and protection bits for the caches and TLB.

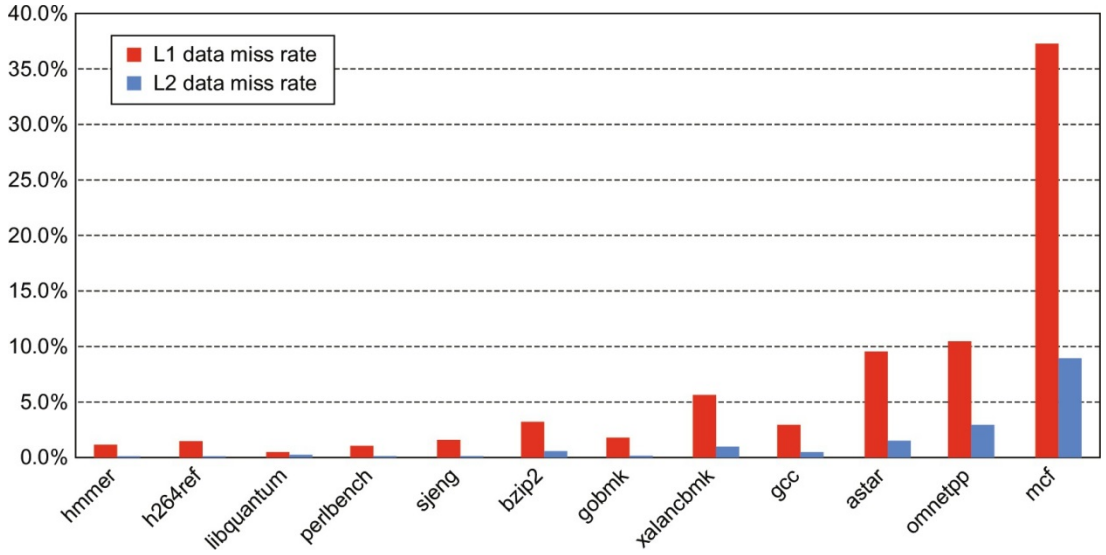

Figure 2.21 The data miss rate for ARM with a 32 KiB L1 and the global data miss rate for a 1 MiB L2 using the SPECInt2006 benchmarks are significantly affected by the applications. Applications with larger memory footprints tend to have higher miss rates in both L1 and L2. Note that the L2 rate is the global miss rate that is counting all references, including those that hit in L1. MCF is known as a cache buster.

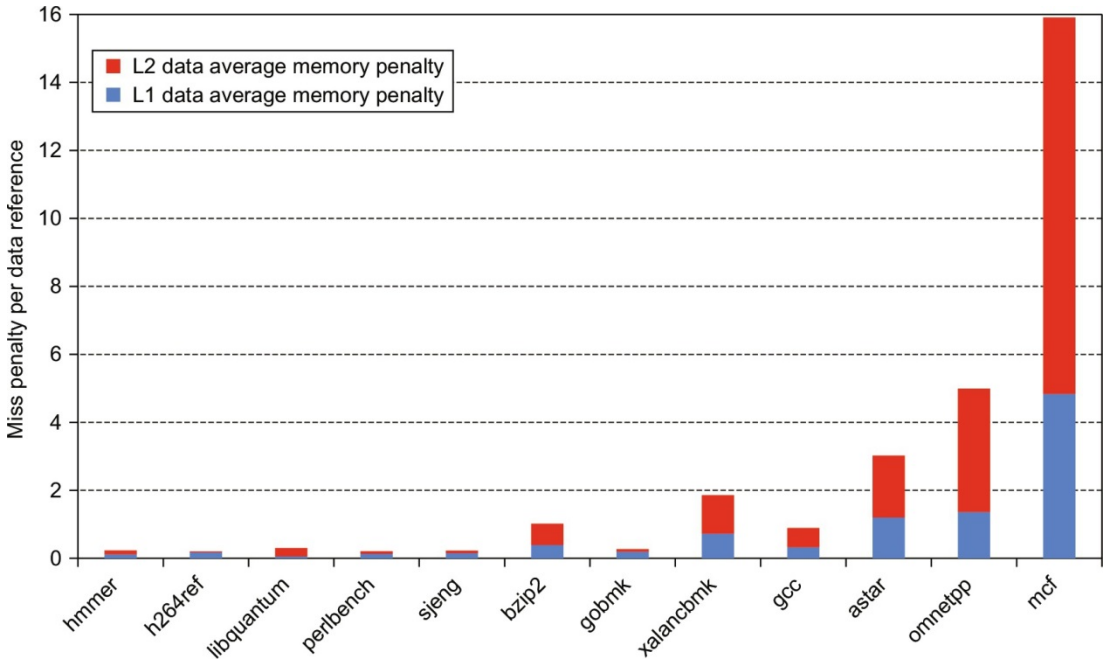

Figure 2.22 The average memory access penalty per data memory reference coming from L1 and L2 is shown for the A53 processor when running SPECInt2006. Although the miss rates for L1 are significantly higher, the L2 miss penalty, which is more than five times higher, means that the L2 misses can contribute significantly.

Characteristic	Instruction TLB	Data DLB	Second-level TLB
Entries	128	64	1536
Associativity	8-way	4-way	12-way
Replacement	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU
Access latency	1 cycle	1 cycle	8 cycles
Miss	9 cycles	9 cycles	Hundreds of cycles to access page table

Figure 2.23 Characteristics of the i7's TLB structure, which has separate first-level instruction and data TLBs, both backed by a joint second-level TLB. The first-level TLBs support the standard 4 KiB page size, as well as having a limited number of entries of large 2–4 MiB pages; only 4 KiB pages are supported in the second-level TLB. The i7 has the ability to handle two L2 TLB misses in parallel. See Section L.3 of online Appendix L for more discussion of multilevel TLBs and support for multiple page sizes.

Characteristic	L1	L2	L3
Size	32 KiB I/32 KiB D	256 KiB	2 MiB per core
Associativity	both 8-way	4-way	16-way
Access latency	4 cycles, pipelined	12 cycles	44 cycles
Replacement scheme	Pseudo-LRU	Pseudo-LRU	Pseudo-LRU but with an ordered selection algorithm

Figure 2.24 Characteristics of the three-level cache hierarchy in the i7. All three caches use write back and a block size of 64 bytes. The L1 and L2 caches are separate for each core, whereas the L3 cache is shared among the cores on a chip and is a total of 2 MiB per core. All three caches are nonblocking and allow multiple outstanding writes. A merging write buffer is used for the L1 cache, which holds data in the event that the line is not present in L1 when it is written. (That is, an L1 write miss does not cause the line to be allocated.) L3 is inclusive of L1 and L2; we explore this property in further detail when we explain multiprocessor caches. Replacement is by a variant on pseudo-LRU; in the case of L3, the block replaced is always the lowest numbered way whose access bit is off. This is not quite random but is easy to compute.

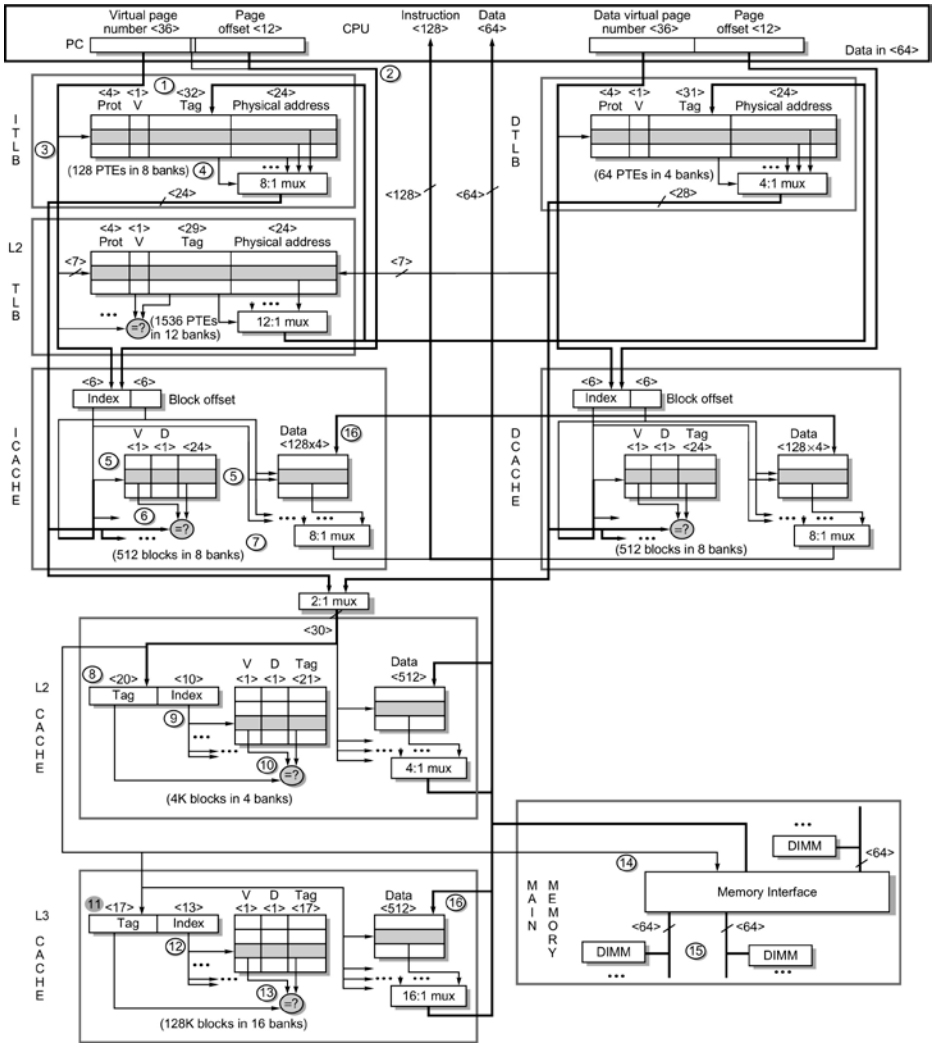

Figure 2.25 The Intel i7 memory hierarchy and the steps in both instruction and data access. We show only reads. Writes are similar, except that misses are handled by simply placing the data in a write buffer, because the L1 cache is not write-allocated.

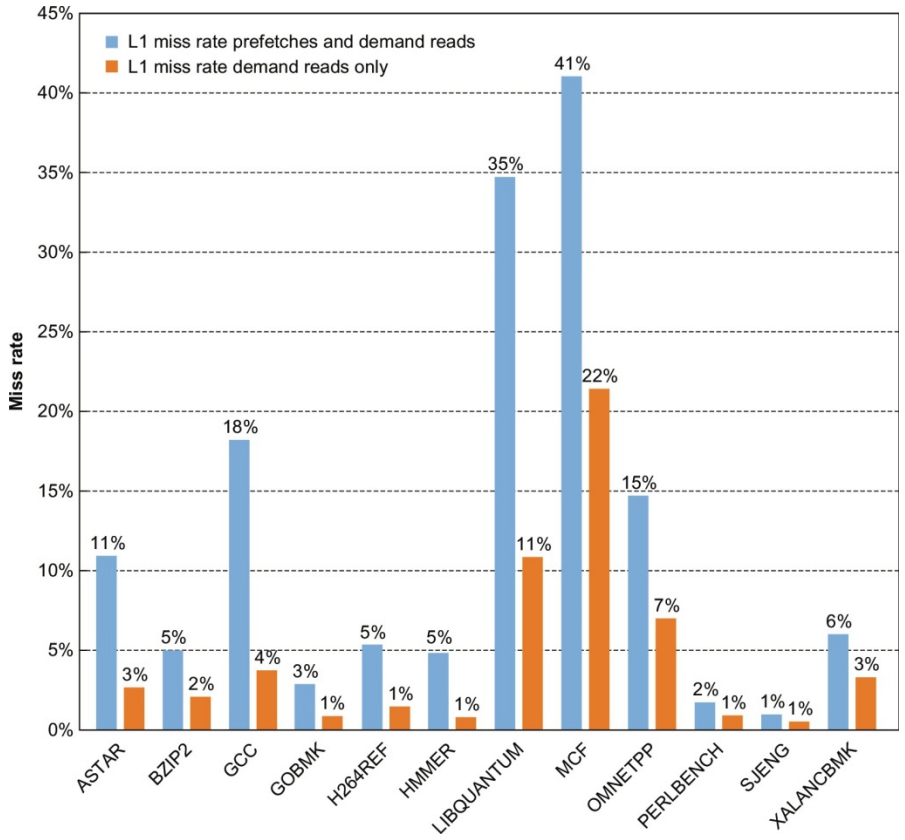

Figure 2.26 The L1 data cache miss rate for the SPECint2006 benchmarks is shown in two ways relative to the demand L1 reads: one including both demand and prefetch accesses and one including only demand accesses. The i7 separates out L1 misses for a block not present in the cache and L1 misses for a block already outstanding that is being prefetched from L2; we treat the latter group as hits because they would hit in a blocking cache. These data, like the rest in this section, were collected by Professor Lu Peng and PhD student Qun Liu, both of Louisiana State University, based on earlier studies of the Intel Core Duo and other processors (see Peng et al., 2008).

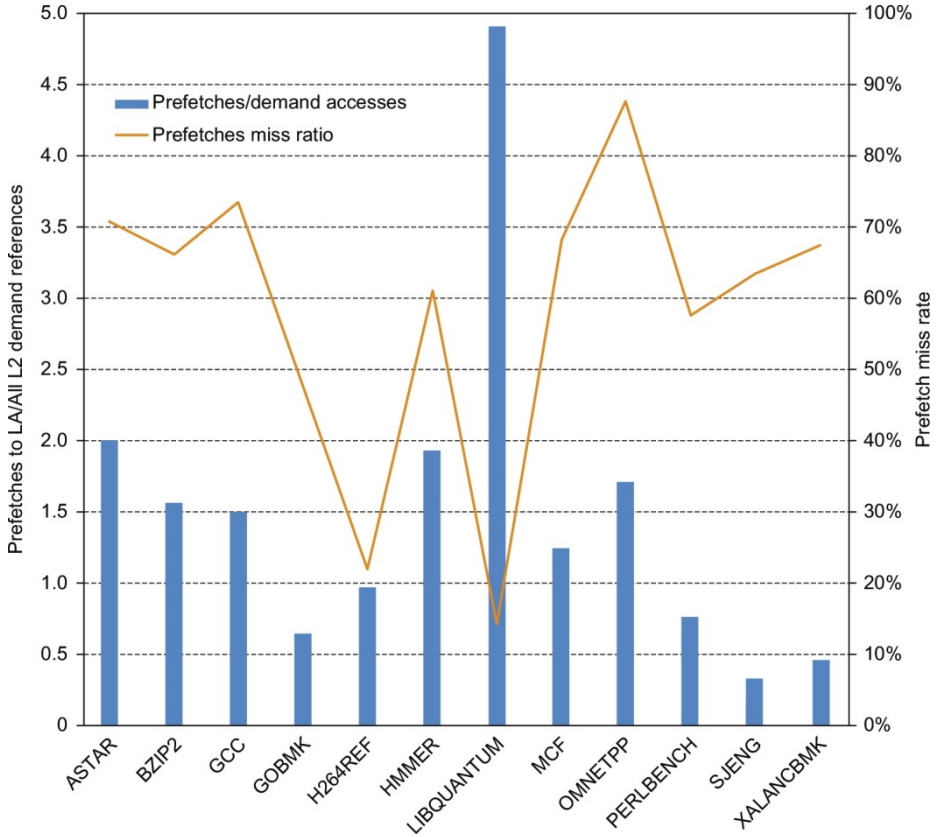

Figure 2.27 The fraction of L2 requests that are prefetches is shown via the columns and the left axis. The right axis and the line shows the prefetch hit rate. These data, like the rest in this section, were collected by Professor Lu Peng and PhD student Qun Liu, both of Louisiana State University, based on earlier studies of the Intel Core Duo and other processors (see Peng et al., 2008).

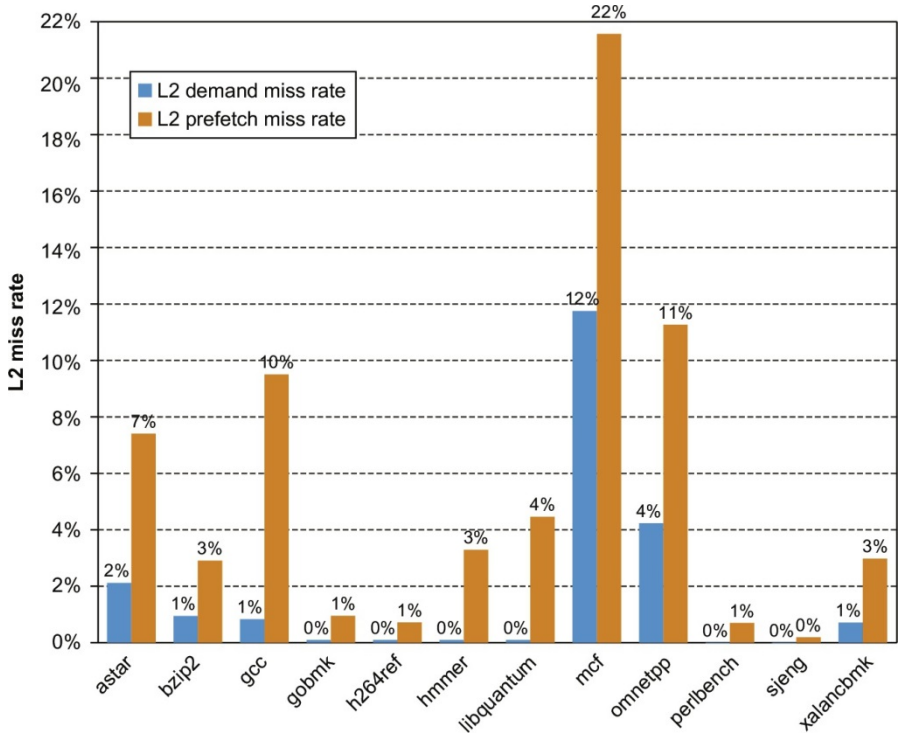

Figure 2.28 The L2 demand miss rate and prefetch miss rate, both shown relative to all the references to L1, which also includes prefetches, speculative loads that do not complete, and program-generated loads and stores (demand references). These data, like the rest in this section, were collected by Professor Lu Peng and PhD student Qun Liu, both of Louisiana State University.

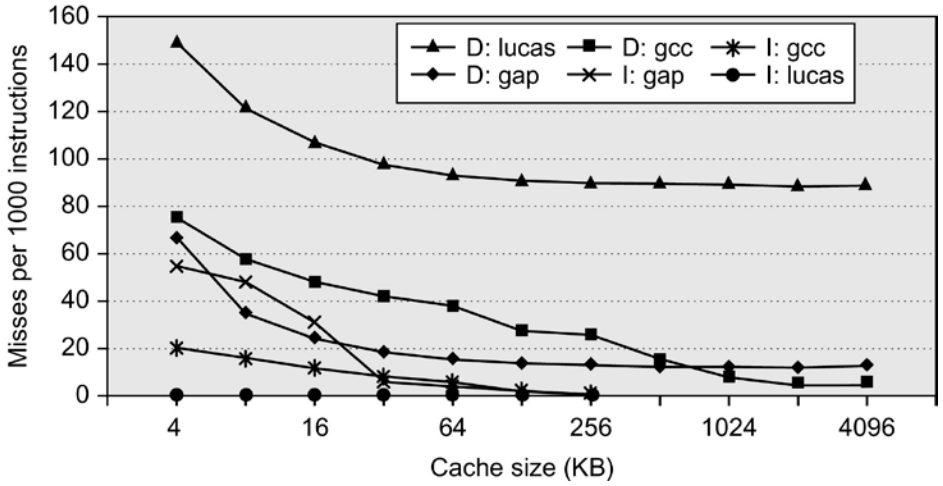

Figure 2.29 Instruction and data misses per 1000 instructions as cache size varies from 4 KiB to 4096 KiB. Instruction misses for gcc are 30,000–40,000 times larger than for lucas, and, conversely, data misses for lucas are 2–60 times larger than for gcc. The programs gap, gcc, and lucas are from the SPEC2000 benchmark suite.

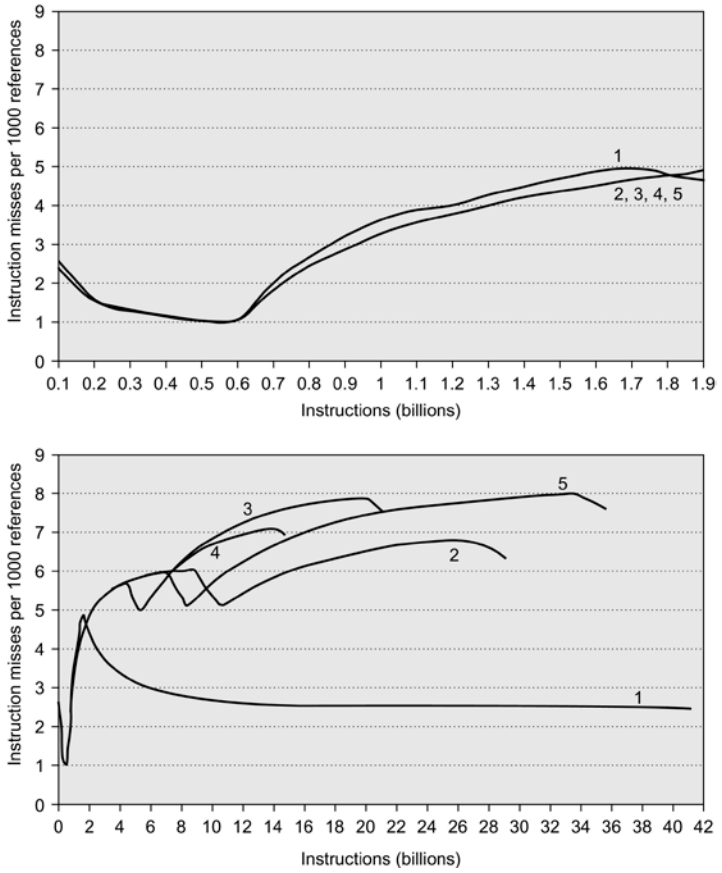

Figure 2.30 Instruction misses per 1000 references for five inputs to the perl benchmark in SPEC2000. There is little variation in misses and little difference between the five inputs for the first 1.9 billion instructions. Running to completion shows how misses vary over the life of the program and how they depend on the input. The top graph shows the running average misses for the first 1.9 billion instructions, which starts at about 2.5 and ends at about 4.7 misses per 1000 references for all five inputs. The bottom graph shows the running average misses to run to completion, which takes 16–41 billion instructions depending on the input. After the first 1.9 billion instructions, the misses per 1000 references vary from 2.4 to 7.9 depending on the input. The simulations were for the Alpha processor using separate L1 caches for instructions and data, each being two-way 64 KiB with LRU, and a unified 1 MiB direct-mapped L2 cache.

Problem category	Problem 80x86 instructions
Access sensitive registers without trapping when running in user mode	Store global descriptor table register (SGDT) Store local descriptor table register (SLDT) Store interrupt descriptor table register (SIDT) Store machine status word (SMSW) Push flags (PUSHF, PUSHFD) Pop flags (POPF, POPFD)
When accessing virtual memory mechanisms in user mode, instructions fail the 80x86 protection checks	Load access rights from segment descriptor (LAR) Load segment limit from segment descriptor (LSL) Verify if segment descriptor is readable (VERR) Verify if segment descriptor is writable (VERW) Pop to segment register (POP CS, POP SS, ...) Push segment register (PUSH CS, PUSH SS, ...) Far call to different privilege level (CALL) Far return to different privilege level (RET) Far jump to different privilege level (JMP) Software interrupt (INT) Store segment selector register (STR) Move to/from segment registers (MOVE)

Figure 2.31 Summary of 18 80x86 instructions that cause problems for virtualization (Robin and Irvine, 2000). The first five instructions of the top group allow a program in user mode to read a control register, such as a descriptor table register without causing a trap. The pop flags instruction modifies a control register with sensitive information but fails silently when in user mode. The protection checking of the segmented architecture of the 80x86 is the downfall of the bottom group because each of these instructions checks the privilege level implicitly as part of instruction execution when reading a control register. The checking assumes that the OS must be at the highest privilege level, which is not the case for guest VMs. Only the MOVE to segment register tries to modify control state, and protection checking foils it as well.

```
#include "stdafx.h"
#include <stdio.h>
#include <time.h>
#define ARRAY_MIN (1024) /* 1/4 smallest cache */
#define ARRAY_MAX (4096*4096) /* 1/4 largest cache */
int x[ARRAY_MAX]; /* array going to stride through */

double get_seconds() { /* routine to read time in seconds */
   __time64_t ltime;
   _time64( &ltime );
   return (double) ltime;
}
int label(int i) {/* generate text labels */
   if (i<1e3) printf("%1dB,",i);
   else if (i<1e6) printf("%1dK,",i/1024);
   else if (i<1e9) printf("%1dM,",i/1048576);
   else printf("%1dG,",i/1073741824);
   return 0;
}
int _tmain(int argc, _TCHAR* argv[]) {
int register nextstep, i, index, stride;
int csize;
double steps, tsteps;
double loadtime, lastsec, sec0, sec1, sec; /* timing variables */

/* Initialize output */
printf(" ,");
for (stride=1; stride <= ARRAY_MAX/2; stride=stride*2)
   label(stride*sizeof(int));
printf("\n");

/* Main loop for each configuration */
for (csize=ARRAY_MIN; csize <= ARRAY_MAX; csize=csize*2) {
   label(csize*sizeof(int)); /* print cache size this loop */
   for (stride=1; stride <= csize/2; stride=stride*2) {

      /* Lay out path of memory references in array */
      for (index=0; index < csize; index=index+stride)
         x[index] = index + stride; /* pointer to next */
      x[index-stride] = 0; /* loop back to beginning */

      /* Wait for timer to roll over */
      lastsec = get_seconds();
       sec0 = get_seconds(); while (sec0 == lastsec);

      /* Walk through path in array for twenty seconds */
      /* This gives 5% accuracy with second resolution */
      steps = 0.0; /* number of steps taken */
      nextstep = 0; /* start at beginning of path */
      sec0 = get_seconds(); /* start timer */
         { /* repeat until collect 20 seconds */
            (i=stride;i!=0;i=i-1) { /* keep samples same */
               nextstep = 0;
               do nextstep = x[nextstep]; /* dependency */
               while (nextstep != 0);
            }
         steps = steps + 1.0; /* count loop iterations */
         sec1 = get_seconds(); /* end timer */
      } while ((sec1 - sec0) < 20.0); /* collect 20 seconds */
      sec = sec1 - sec0;

      /* Repeat empty loop to loop subtract overhead */
      tsteps = 0.0; /* used to match no. while iterations */
      sec0 = get_seconds(); /* start timer */
         { /* repeat until same no. iterations as above */
            (i=stride;i!=0;i=i-1) { /* keep samples same */
               index = 0;
               do index = index + stride;
               while (index < csize);
            }
         tsteps = tsteps + 1.0;
         sec1 = get_seconds(); /* - overhead */
      } while (tsteps<steps); /* until = no. iterations */
      sec = sec - (sec1 - sec0);
      loadtime = (sec*1e9)/(steps*csize);
      /* write out results in .csv format for Excel */
      printf("%4.1f,", (loadtime<0.1) ? 0.1 : loadtime);
     }; /* end of inner for loop */
     printf("\n");
   }; /* end of outer for loop */
   return 0;
}
```

Figure 2.32 C program for evaluating memory system.

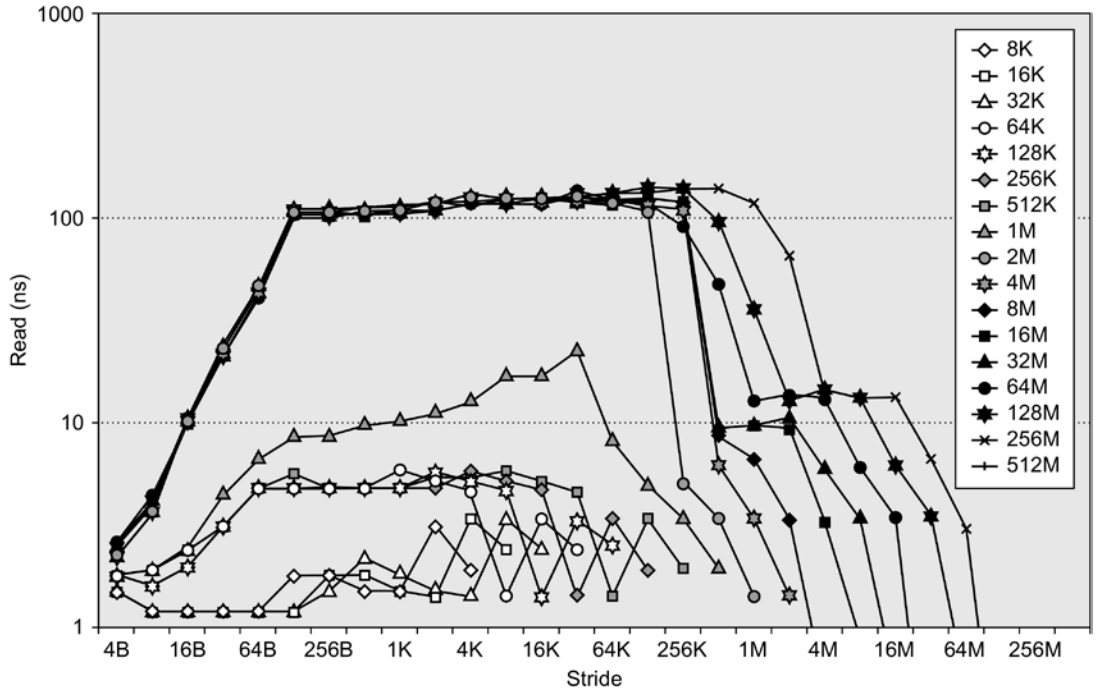

Figure 2.33 Sample results from program in Figure 2.32.

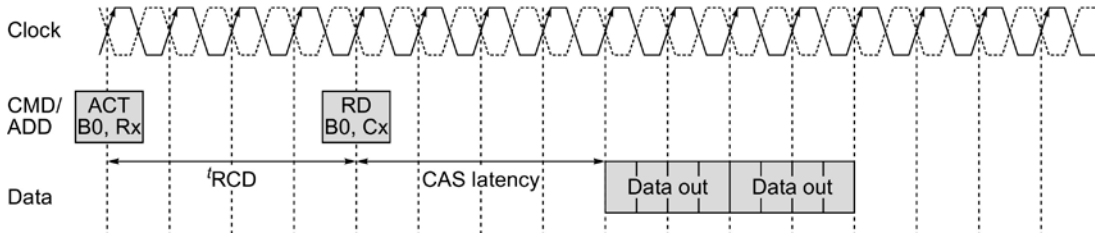

Figure 2.34 DDR2 SDRAM timing diagram.

Benchmark	Native	Pure	Para
Null call	0.04	0.96	0.50
Null I/O	0.27	6.32	2.91
Stat	1.10	10.69	4.14
Open/close	1.99	20.43	7.71
Install signal handler	0.33	7.34	2.89
Handle signal	1.69	19.26	2.36
Fork	56.00	513.00	164.00
Exec	316.00	2084.00	578.00
Fork + exec sh	1451.00	7790.00	2360.00

Figure 2.35 Early performance of various system calls under native execution, pure virtualization, and paravirtualization.

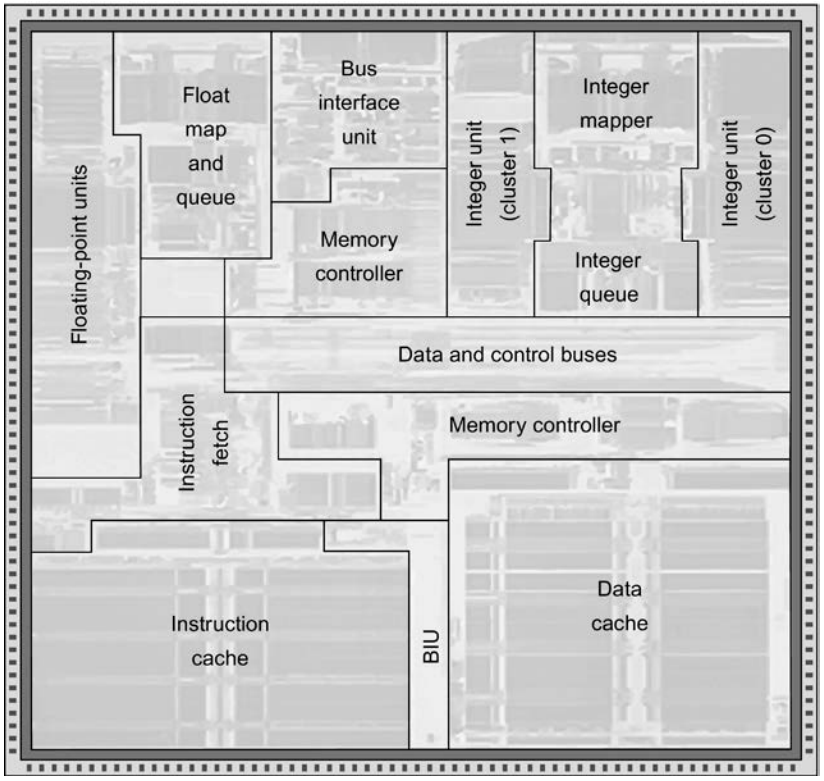

Figure 2.36 Floorplan of the Alpha 21264 [Kessler 1999].