address that is concatenated with the ten LSBs of the displacement to form the real memory address.

This system required two memory accesses to perform a name translation, a rather slow process as the processor did not have a cache. As will be described in Section 3.4, the GE 645 also used a translation lookaside buffer to cache the most recently used segment descriptors.

Due to the poor performance of a segmented system that is virtualized in real memory, two forms of hardware supported segment tables have evolved. One system supports segment length checking, while the other does not. Examples of these two types are given in this section.

Hardware Support with Segment Length Checking

An example of hardware support for the segment table of paged segmentation is found in the Intel i386, i486, and Pentium microprocessors shown in Figure 3.31, which is an extension of Figure 3.6. The i386 member of the x86 family had significant changes to the memory management system that have carried over into all subsequent members of the family. While there are differences between these later processors, they are sufficiently similar that only one description is given. The terminology used is that of the i486 and Pentium, which is different from the terminology of the i386. The segmentation system evolved from steps taken to extend the address space of the ix86 family of processors. Evolution of the mapping structure design was constrained by the requirement for upward compatibility for MS-DOS and applications software. A break in this capability occurred with the i386 with its effective address extended to 32 bits compared to the 16 bits of previous members of the family [MORS80].

There are two paths for obtaining a segment base address value; one path uses active descriptors stored in segment registers, the other path uses descriptors found in descriptor tables in memory. The six active descriptors, found in the *invisible part* of the segment registers, are indexed by implication with each memory reference in parallel with the selectors in the *visible part* of the segment registers; this is essentially a zero time access. The descriptors (both in the segment registers and descriptor tables) have a 32-bit segment base address field, a 20-bit segment limit field, and 12 control bits. One of the control bits is a valid bit, indicating that the descriptor information contained is valid. If the entry is valid, segment base address value selection is performed via the descriptor table and the memory resident translation tables are bypassed. The linear address is formed by adding the segment base address to the displacement field from the instruction.

In addition to various access checks based on the control bits, the

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000199

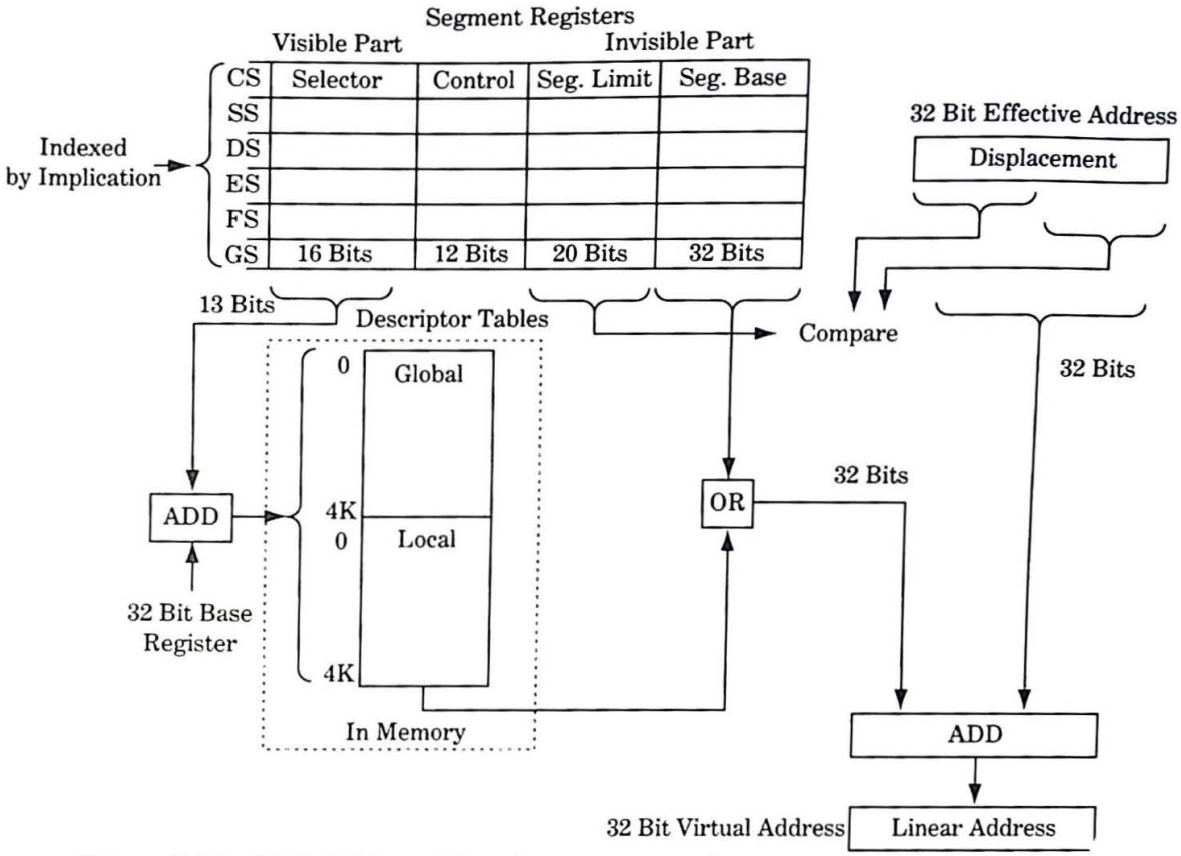


FIGURE 3.31 i386, i486, and Pentium segmentation support.

segment length is also checked. A bit in the control field determines if the comparison is between the 20 MSBs or the 20 LSBs of the displacement: if the MSBs, the segment length is measured in pages as a page is 4 Kbytes; if the LSBs, the segment length is measured in bytes. Thus the segment size is either $2^{20} - 1$ pages or bytes.

If the segmentation information in the segment registers is not valid, segment base address values are obtained via memory-based descriptor tables indexed by one of six, 16-bit selector registers, called *visible selectors*. These selectors are indexed by implication in parallel with the hidden descriptors. The 13 MSBs of the indexed selector register are used, after being added to a 32-bit base register value, as an address into memory to access a descriptor table entry; each descriptor is eight bytes. These tables are in two halves: the *global* table and the *local* table. The field allocation to the entries of the descriptor tables is identical to the hidden descriptors. The same access checks and segment limit checks are made before a memory access can proceed. The segment base and

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000200

displacement are added to form the linear address that will be the address for the paged system, as shown in Figure 3.12.

With two instances of the same segmentation information in the segment registers and descriptor tables, the normal provisions must be made to ensure that coherency is maintained. This problem is made difficult because the segment registers are updated by hardware while the descriptor tables are updated by software. Thus, software can change a descriptor in memory and the descriptor in the segment registers must be invalidated.

Hardware Support Without Segment Length Checking

A number of systems provide hardware segment tables without segment length checking. Examples are RS/6000, PowerPC 601, and the HP Precision Architecture. For example, the RS/6000 map of segment registers, as shown in Figure 3.5, is loaded with values that are created by the operating system. The function of assuring that valid displacements do not exceed the segment length and generating the page table addresses is an operating system task.

The PowerPC 601, one of the examples of paged segmentation without hardware for segment length checking, is shown in Figure 3.25. The page index is 16 bits, which constrains the segment size to be no more than 2^{16}-1 pages. With a page size of 4 Kbytes, a segment is 26 Mbytes.

3.4 Translation Lookaside Buffers

The translation of a virtual page name into the page frame address via memory resident tables can take a significant number of memory cycles. To hide the translation time, a page table system (direct or inverted) is usually augmented with a cache of active page frame addresses. This type of cache is usually known as a *translation lookaside buffer* (TLB), a name used by IBM. Motorola uses the term *address translation cache* (ATC); DEC uses the term *translation buffer* (TB); and Intel uses the term *page translation cache* (PTC). The Multics system employed the first known instance of a TLB [SCHR71].

Translating page names and block names, as shown in Figure 3.32, is usually performed in a hierarchy of translation systems, much as a memory reference is processed in a hierarchical memory. This figure illustrates the number of translation maps possible, not the tables for any particular processor. Usually, all of the TLBs are accessed in parallel and the first valid translation is used to form the real memory address. If translation fails in the TLBs, the page tables, either direct or inverted, are then accessed before a page fault is declared. As will be discussed when the MC88200 TLBs are described, one level of this hierarchy may

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000201

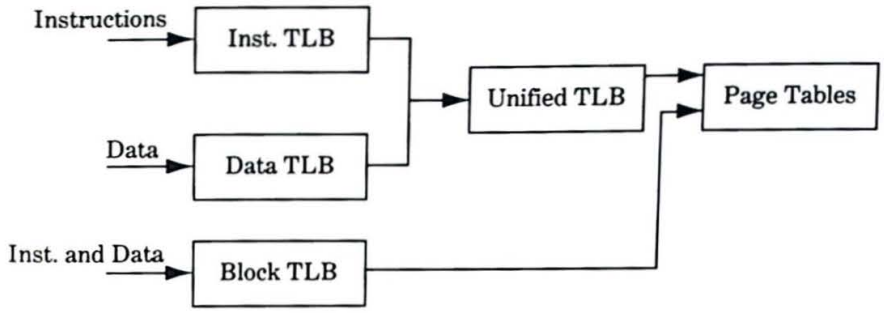


FIGURE 3.32 Canonical name translation hierarchy.

be implemented for both page and block name translation. Also, portions of the page tables may be resident in the processor's cache, further improving the name translation time.

Multilevel TLBs have many of the design problems of multilevel caches and hierarchical memories. There are design issues of information allocation, coherency, particularly when the processor is used in a multiprocessor configuration, and multilevel inclusion. There is no need for read and write policies as with a cache, but there are issues dealing with invalidating the contents of TLBs.

The reduction in name translation latency with a TLB is similar to that of a data or instruction cache. There is some probability that the page frame address will not be found in the TLB; as a result, the name translation time is the weighted average of the time to translate in the TLBs or in the page tables. On a TLB miss, the delay in access time is the two or more memory cycles for accessing the page tables plus the table walking time (discussed in Section 3.5.5). Note that there can be two reasons for a TLB miss. First, if the referenced page is not in real memory, there will be a miss in the translation tables. Second, a miss occurs when the page is in real memory and its translation tables are correct but the translation information is not in the TLB. It is this second form of TLB miss that is most often noted in the literature.

A TLB holds the translated addresses of pages that are also resident in the page tables. Most access checks have been made and do not require verification when translating a virtual page or segment name in a TLB. Because the name translation process can be hierarchical, the treatment of multilevel inclusion is a design issue.

The PowerPC 601 has three levels of translation: the Instruction TLB (ITLB), Unified TLB (UTLB), and the hash-accessed inverted page tables. For this system, multilevel inclusion is maintained with the following policy. Allocation of translation information to the page tables is a software process that follows a page fault. Allocation to the ITLB and UTLB is performed by hardware following a page fault that brings

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000202

Operation	ITLB	UTLB	Page Table
Add entry	Add on most recent translation (MRT), set valid bit.	Add on most recent translation (MRT), set valid bit.	The page table is locked, the entry added with a move instruction, then the table is unlocked, all under program control.
Replace entry	Replace if a new entry is translated and there is no vacant slot. MRT bits select entry to be replaced.	Replace if a new entry is translated and there is no vacant slot. MRT bits select the slot within the sector if there is a set conflict.	An entry is: locked, invalidated, flushed, updated, marked valid and unlocked, all under program control.
Delete entry	Valid bit is reset explicitly by the execution of tibie instruction or as a side effect of some other instructions.	Valid bit is reset explicitly by the execution of the tibie instruction or as a side effect of some other instructions.	All entries have their valid bits reset when there is a context switch.

TABLE 3.5 PowerPC 601 maintaining translation multilevel inclusion.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000203

a new page into the real memory. At any time, entries that are in the ITLB are in the UTLB and the page tables and entries in the UTLB are in the page tables but not necessarily in the ITLB.

The management policy of the three tables, ITLB, UTLB, and the page tables, is shown in Table 3.5. Three cases must be comprehended to assure multilevel inclusion: adding a new entry, modifying an old entry, and deleting an old entry. Note that the two TLBs are managed by hardware, while the page table is managed by software. Also note the synchronization operations that are required to lock and unlock the page tables.

3.4.1 *TLB Organization*

The organization of a canonical TLB is shown in Figure 3.33. Note that this figure is similar to Figure 2.4 and differs only in that page table entries (PTEs) with page frame addresses are stored rather than data and only a valid bit is required. If a page is evicted to the disk, the entry in the TLB can become invalid and is modified by the operating system or microcode. A TLB, being a cache of translation information, can be organized for access as: (1) associative search, (2) n-way set associative search, and (3) hash accessed.

Note that the size of a TLB is sometimes described by its *length* o. *size*, not the number of sets. With associative TLBs, the size is described in *entries*, not sectors as with caches. A TLB operates on the principle of temporal page locality; when a page is loaded into real memory, the page tables and TLB are updated. Because of page locality, caching addresses can be quite effective in reducing the latency of name translation and the access of real memory. Simulations and measurements show that TLBs have a hit ratio in the range of 90% [SATY81]. The hit ratio

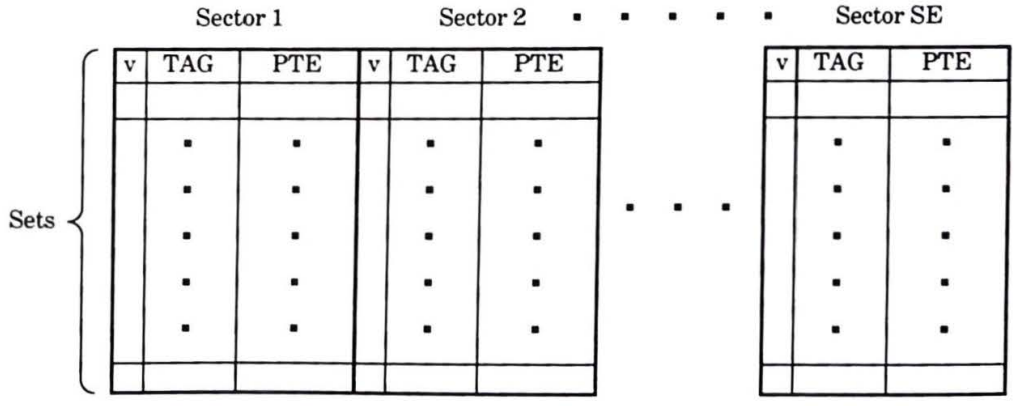


FIGURE 3.33 Canonical TLB.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000204

of a TLB is, as with other caches, a function of its size. D. W. Clark et al. [CLAR85] and M. D. Hill [HILL87] provide data on TLB hit ratios.

In general, TLBs are invisible to the program as they are loaded under hardware control and serve to speedup the name translation process. However, TLBs must usually be invalidated when there are changes to the page tables such as will occur when a page table change occurs that will violate coherency. There must be a replacement policy for the n-way set associative TLB, and the same policies described in the section on caches apply here. For example, the policies may be MRU, LRU, Random, Clock, and others [BAER80].

Associative Searched TLBs

Associative searched TLBs have a number of positive features. One feature is that, for small TLBs, the latency of translating a name is quite small. Another feature is the ease of expandability. As there is only one set, the number of sectors in the TLB can be increased without architectural change. And, the number of sectors need not be an even power of two. This feature eases implementation of TLBs on a chip. For these reasons, most contemporary microprocessors use associative searched TLBs. Examples are discussed in the following paragraphs.

MC68451. An example of an associatively searched TLB, called an *address register translation table*, is the Motorola MC68451, shown in part in Figure 3.34 [MOTO83]. This rather strange device was designed for use with the MC68000 to support paged segmentation. Providing virtual page and segmentation support for functions such as segment length checking and table management is a pure software function.

The virtual 23-bit word address is divided into a segment name and displacement fields. The segment name is the key for an associative access of the page directory, reading out a page frame address, mask, and control bits. The mask selects the MSBs of the page address and the LSBs of the segment name concatenating them with the 7-bit displacement field to form the 23-bit real address. By adjusting the mask, the size of the segment varies for 128 words (mask all 1s) to 65K words (mask all 0s).

Wakerly [WAKE89] states that the MC68451 was not successful because of its limitations, which are believed to be the limited virtual address space and too much flexibility. For this and other reasons, the MC68451 was not used by early customers for the MC68000. Sun and Apollo crafted their own MMU, which are modeled after the memory mapping system of the Atlas Computer (described in Figure 3.15).

IBM RS/6000. This processor architecture specification calls for a unified TLB, but specific implementations may have separate ITLBs and DTLBs [IBM90]. One such implementation has a split TLB. The DTLB

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000205

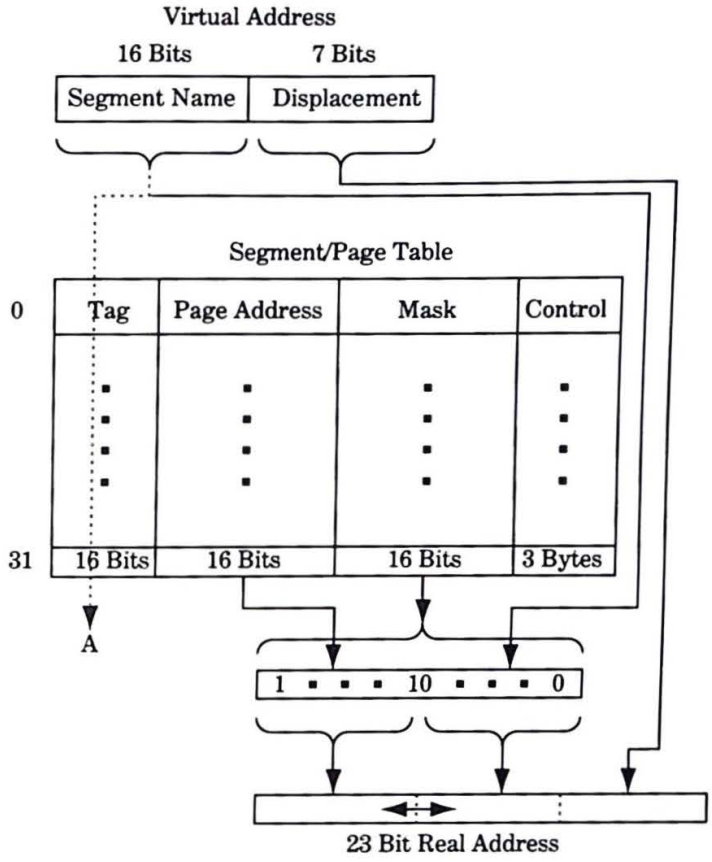


FIGURE 3.34 Associative register name translation.

is organized as (64, 2, 1, PTE) and is late select. However, the ITLB of this processor is believed to be associatively searched as shown in Figure 3.35. The 40-bit page name is used as the key to search the ITLB associatively; the number of sectors is not revealed in the literature. A hit on a translation provides a 20-bit page frame address that is concatenated with the 12-bit displacement of the virtual address. If the translation is successful, the reference to real memory is initiated. If not successful, the reference address is translated in the translation tables resident in either the cache or real memory.

Note the similarity between this TLB design and the Atlas name translation system depicted in Figure 3.15. The major differences between the two are the design parameters, not the organization.

MC88200. The MC88200 that supports several Motorola processors (88100, 68030) has two UTLBs, one for user programs with 4-Kbyte pages and one for system programs with 512-Kbyte blocks [MOTO90a]. Note that Motorola calls TLBs *address translation caches*. Both of the TLBs are unified, translating the page and block names of both instruc-

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000206

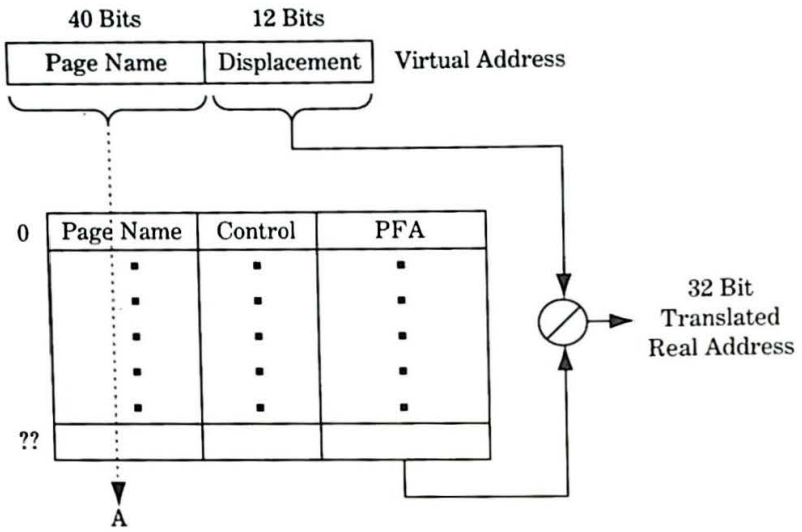


FIGURE 3.35 RS/6000 ITLB.

tions and data. Block diagrams for these TLBs are shown in Figure 3.31, and note should be taken again of the similarity between these TLBs and the Atlas name translation system.

The block TLB contains high-use translation information that is cached at the block level, thereby reducing TLB misses. Operating systems occupy the same address space, for instructions and data, over long periods of time. Thus, having a large page, called a *block*, can significantly reduce the number of block name translation misses and page misses. Note that these blocks are not the same as cache blocks.

The block TLB has 10 sectors that are associatively searched on the 13-bit block name and is organized as (1, 10, 1, PFA). On a hit, the 13-bit page frame address is concatenated with the 17-bit block displacement to give a 30-bit real memory address. Two of the 10 sectors are hardwired to provide an identity mapping in the upper 1M bytes of real memory for supervisor space.

User programs have higher paging activity, and the page TLB reduces external and internal fragmentation that would occur if the page is the same size as the block. The page TLB, also shown in Figure 3.36, has 56 sectors that are associatively searched on the 20-bit page name from the virtual address. The organization is (1, 56, 1, PFA). The 20-bit page name is associatively compared to the 56 page name tags of the resident pages. Upon a hit, the 20-bit page frame address is concatenated with the 10-bit page displacement to give a 30-bit real memory address. The page TLB is managed by the operating system with entries being evicted on a FIFO basis; a new entry pushes out the oldest entry.

A page name translation is conditionally and concurrently attempted

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000207

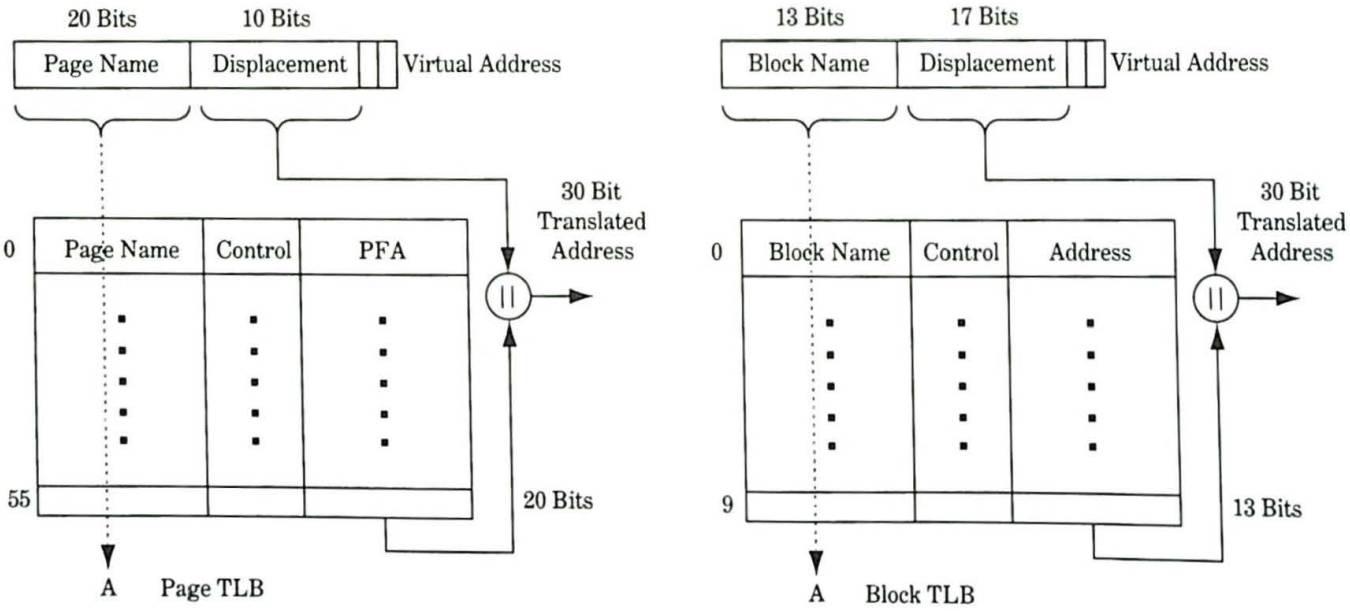


FIGURE 3.36 MC88200 TLBs.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000208

PTLB	BTLB	Action
Hit	Hit	Use BTLB
Hit	Miss	Use PTLB
Miss	Hit	Use BTLB
Miss	Miss	Table search, update PTLB and retry translation

Note. The BTLB is loaded by the operating system.

TABLE 3.6 MC88200 TLB management.

in both TLBs. If an access misses on *both* of the TLBs, the name translation is made via the page tables previously described. Table 3.6 shows the action taken for the four possible events when accessing the two TLBs.

MIPS R2000. This processor has a TLB integrated onto the processor chip [MIPS87]. The TLB has 64 entries and is accessed by an associative search on the 20-bit virtual page name. The page size is fixed at 4 Kbytes. Each TLB entry has the fields:

20 bits: virtual page name;
20 bits: page frame address;
6 bits: PID number that must match the PID value in a processor register;
4 bits: Control bits: Read Only, Non-Cacheable, Valid and Global;
14 bits: unused.

The page name from the effective address plus the PID value gives a virtual name space of 26 bits. The addressable unit is a 4-byte (32-bit) word. The 20-bit page frame number is concatenated with the 10-bit displacement, giving a 30-bit real memory address or 4 Gbytes.

DEC Alpha. The DEC Alpha processor has split, associative TLBs: one for the instruction page names and one for the data page names [DIGI92]. The virtual memory of the Alpha system is paged with variable page sizes. The I-Stream TLB has 12 sectors, 8 for 8-Kbyte pages and 4 for 4-Mbyte pages. The D-Stream TLB has 32 sectors, each of which can be used to translate to 8-Kbyte, 64-Kbyte, 512-Kbyte, or 4-Mbyte pages.

PowerPC 601. The PowerPC 601 has a small first-level instruction TLB (ITLB) for translating instruction page names and a unified TLB (UTLB) for data page names and instruction page names that miss in the ITLB [MOTO93]. The ITLB is searched associatively and has four sectors. The 20-bit page name of the 32-bit effective address is the search key. A hit provides a 20-bit page frame address that is concatenated with the 12-bit displacement to give the real memory address. Note that the extension to the virtual address is not part of the search key.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000209

If the instruction page name translation fails in the ITLB, the translation is attempted in the UTLB organized as (256, 2, 1, PFA). If this translation fails, page name translation is then performed via the tables described in Section 3.3.1.

In addition to the ITLB and UTLB, the processor can also translate block names in a small TLB, called *block-address translation registers* (BATR). These registers are organized as (1, 4, 1, PFA). The BATR holds four block name translated base addresses. The size of the blocks is a programmable parameter and can be in the range of 18 Kbytes doubling to 8 Mbytes.

VAX 11/780. The VAX 11/780 has a small first-level instruction TLB, called an *instruction translation buffer*. This ITLB has only one sector organized as (1, 1, 1, PFA). In other words, the last translated page name is held in the ITLB and is concatenated to the displacement for every instruction read. Thus instruction page name translation is a zero time process. If an instruction fetch crosses the boundary of the page (the page name changes), the new page name is translated in a second-level unified page table.

n-Way Set Associative TLBs

A number of TLBs are implemented with *n*-way set associative organizations in which the page name is congruence mapped for translation. To accomplish this, the page name is allocated to a set index field and an unnamed field that is compared to the tags of an accessed set. Some of the TLBs that use this organization are discussed below.

i386, i486, and Pentium. These processors support paged segmentation, and virtual page names are translated via a multilevel page tables system as shown in Figure 3.12. A TLB organized as (8, 4, 1, PFA) is implemented as a unified TLB on the i386, as shown in Figure 3.37. This same design is also used as a unified TLB on the i486. With the Pentium, two copies of the same TLB are used as split TLB without a unified TLB.

The page name is allocated into two fields: an 8-bit index into the TLB and a 12-bit field that is compared to the TLB tags. If there is a true comparison on one of the four sectors, the 20-bit page frame address is concatenated with the 12-bit page displacement to form the real memory address. If there is a miss in the TLB, the address is translated via the multilevel page tables.

Note that the TLB provides name translation of the 20-bit page name and not the extended virtual address; the 20-bit extension is not translated. The consequence of this restriction is that only translated page names for one root directory at a time are in the TLB. As a result,

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000210

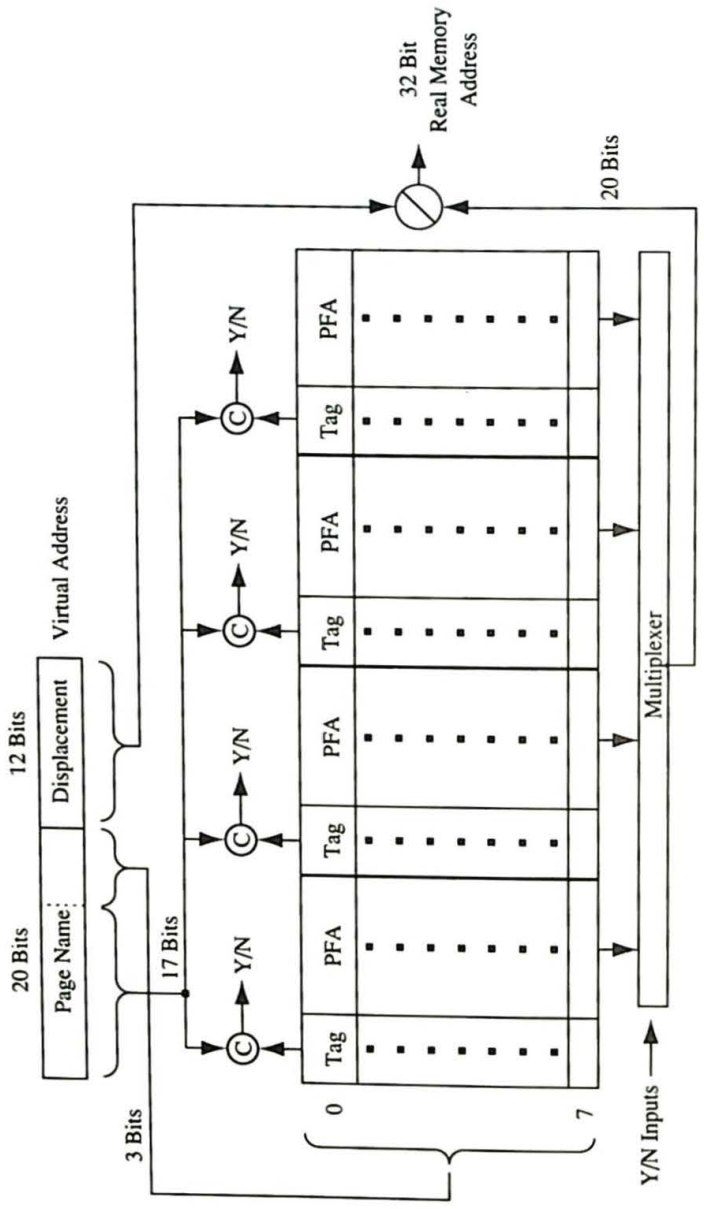


FIGURE 3.37 i386, i486, and Pentium TLBs.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000211

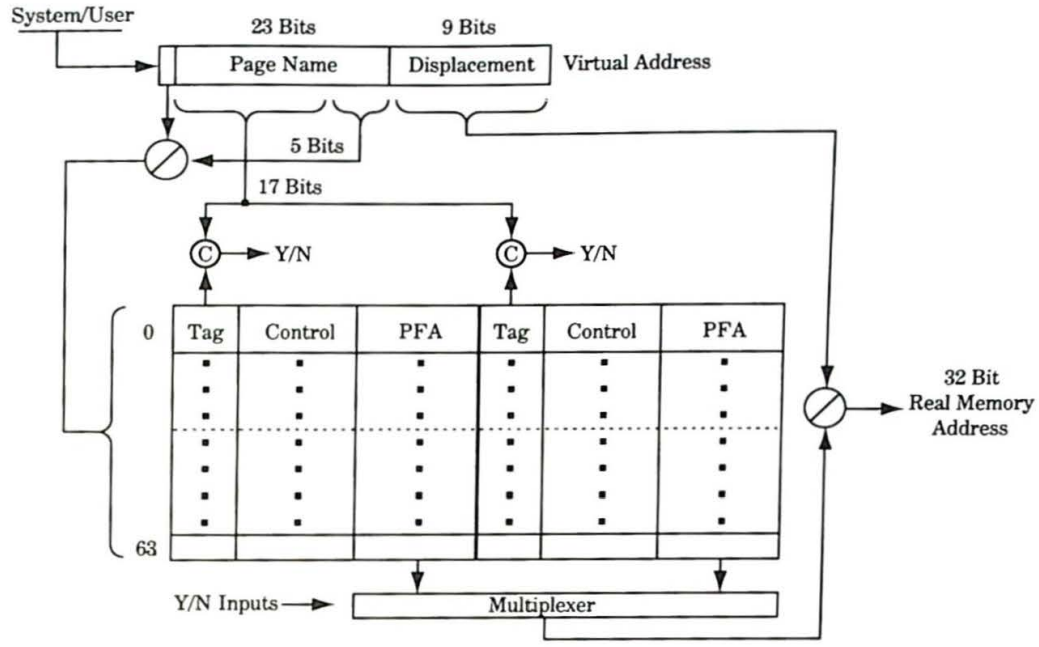


FIGURE 3.38 VAX 11/780 unified TLB.

aliases are possible. To prevent aliases, any change of the root directory by the executing program requires a reload of the TLB.

VAX 11/780. The VAX 11/780 has a unified TLB that translates the page names of data references and misses in the ITLB [SATY81] and [HAMA90]. A block diagram of the unified TLB is shown in Figure 3.38. The 32-bit virtual address is divided into a 9-bit page displacement field and a 23-bit page name field. The five LSBs of the page name are concatenated with the MSB to form a 6-bit index into the 2-way set associative TLB organized as (64, 2, 1, PFA). An interesting feature of this TLB is that the MSB of the page name (system/user bit) is used as the MSB of the index to select between two halves of the TLB. This guarantees that the system and user both have full access to 1/2 of the TLB and 1/2 of virtual memory space.

PowerPC 601. The PowerPC 601 has a first-level TLB and a unified (UTLB) that translates instruction page names that failed to translate in the ITLB and data page names. This TLB is organized as (256, 2, 1, PFA).

Hash-Accessed TLBs

IBM S/370/168. This system has a unified TLB translating both instruction and data references. The 14-bit page name is hashed to create a 6-bit index into the TLB that is organized as (64, 2, 1, PFA), an unusual organization [MATI77]. By making the TLB 2-way set associative, two

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000212

TLB Access Method	Page Table Access Method	
	Direct	Inverted
Associative	MC88200, MC88110	IBM RS/6000 PowerPC 601 (Inst.)
n-Way set associative	i386, i486, Pentium	PowerPC 601 (Unified)
Hashed	IBM S/370/168	HP Precision (only one in the system)

TABLE 3.7 Name translation methods.

virtual page names can hash into the same page table entry and yet produce a valid translation. Remember that approximately two probes are required for a successful translation. This level of redundancy eliminates the need to link entries (common with full name translation in hash accessed systems), and a large fraction of collisions can be accommodated with this design.

HP Precision Architecture. This processor does not have a TLB to speedup the name translation process. The hashed access page translation (Figure 3.24) is as fast as a TLB, thus one is not used.

Summary of TLB Organizations

The selection of the TLB organization by a designer is sometimes constrained by the design of the translation tables. Table 3.7 shows the TLB design and the table translation method for a number of processors.

3.4.2 *TLB Miss Ratio Data*

Satyanarayanan and Bhandarker [SATY81] performed a miss ratio evaluation of TLB size, degree of associativity, sensitivity to context switching, and replacement algorithms. The tests are conducted by means of a simulator and FORTRAN programs for the VAX product family. The evaluation was performed to set the design parameters prior to the construction of a processor. A clear interval is used as a surrogate for context switching. By periodically clearing the TLB, the effect of clearing based on a context switch can be evaluated. The authors of this evaluation characterize TLB performance as

$$\text{TLB miss ratio} = \frac{\text{no. of main memory references to page table}}{\text{explicit virtual memory reference}}$$

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000213

No. of Sectors per Set	No. of Sets			
	32	64	128	256
1	97	47	22	8
2	25	9	3	2
4	7	2	1.5	1

Number of page table references $\times 10^3$ for 5.5×10^6 explicit references.

TABLE 3.8 VAX TLB miss ratios.

Explicit virtual memory references are those references generated by the program. In addition, there are *implicit* references to memory that are generated by the translation process itself. The number of implicit references is influenced by the organization of the translation tables themselves, such as the number of levels. One of the performance results of the VAX study is shown in Table 3.8. The data indicates the reduction in page table references as the size and organization of the TLB are changed.

As with other figures that show P_{miss} data for cache, the diagonals of this figure are for TLBs of equal size. The total number of explicit references to the memory system is 5.5×10^6, and the number of references that miss the TLB are shown. Thus, for this table, small numbers represent an improvement in the number of page names translated in the fast TLB. This data indicates a clear preference for a large TLB and for higher degrees of associativity and a smaller number of sets. Higher degrees of associativity reduce the occurrence of set conflicts. Thus the preference is for associative TLBs in many processors that are implemented in VLSI technology where the increase in tag bits is not a significant cost problem.

Another source of TLB miss data is [WOOD86]. Five benchmarks are executed on six processors. Four of the processors are of the VAX family (512-byte page) and two are IBM S/370 or Amdahl compatibles (4-Kbyte page); excerpted results of the tests for TLB miss ratios, expressed in percentages, are shown in Table 3.9.

This information is quite interesting in that while the size and organization of the TLB are important, the size of the page is even more important. A larger page will result in a lower miss ratio for TLBs of equal size. This result confirms the use of a page TLB, as with the MC88200 and the PowerPC 601, for the operating system. This data also confirms the advantage of associative TLBs.

Saavedra and Smith [SAAV93] measured TLB miss ratios on a number of commercial processors for six SPEC benchmarks. Table 3.10

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000214

No. of Sectors per Set	No. of Sets		
	128	256	512
1 (VAX, 512 Kbytes)	3.68		0.639
2 (VAX, 512 Kbytes)	1.78		0.324
2 (IBM S/370, 4 Kbyte)	0.097	0.023	0.014

LISZT Benchmark.

TABLE 3.9 TLB miss ratios (%).

Processor	No. of Sets	Sectors per Set	Page Size	Miss Ratio (%)
DECstation 3100	64	64	4096 Bytes	2.42
DECstation 5400	64	64	4096 Bytes	2.42
DECstation 5500	64	64	4096 Bytes	2.42
MIPS R/2000	64	64	4096 Bytes	2.42
VAX9000	1024	2	8192 Bytes	0
RS/6000 530	128	2	4096 Bytes	1.31
HP 9000/720	64	64	8192 Bytes	1.30

TABLE 3.10 TLB percentage miss ratios for SPEC Benchmarks.

shows an excerpt of their results; the average TLB miss ratio is given for the six benchmarks.

This data also indicates the primacy of page size in determining TLB miss ratios. A large page, or the granularity of the TLB, significantly reduces the miss ratio.

We can speculate that the arguments advanced by Hill and others on the advantage of direct caches will apply to TLBs as well. The arguments (discussed in Chapter 2) suggest that a direct cache, with its shorter logic path but with a higher miss ratio, may be the most effective design for reducing TLB translation time. [HILL86] also provides TLB miss ratio data for a number of processors and three benchmarks.

As discussed in Chapter 5, TLB translation time is of significant importance for caches that are addressed with translated real addresses. However, for some cache organizations, the TLB must be only fast enough to match the access time of the cache. Thus, higher degrees of set associativity, with lower miss ratios, may be quite satisfactory.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000215

TLB Size, Sectors	Normalized No. of Instructions Executed	TLB Miss Ratio
0	1.0	1.0
4	2.3	0.106
8	2.6	0.029
16	2.8	0.0125

Note. LRU algorithm.

TABLE 3.11 GE-645 Multics performance.

3.4.3 *TLB Impact on Virtual Memory Performance*

There is little published information on the performance of a virtual memory system as the various parameters (for example, page size, TLB size and organization, and main memory size) are varied. The little data that does exist is some material on performance: the papers on Multics [SHEM66, SCHR71] and the VAX 11/780 [CLAR85].

Schroeder [SCHR71] reports on the performance of a virtual memory system as the TLB size is varied. The system is the GE-645; the operating system is Multics, which is a paged-segmented system. The page size is 1K words or 4.5 Kbytes. The system was evaluated with associative TLBs having 0, 4, 8, and 16 entries. The time for a TLB search is between 200 and 600 ns, while the main memory is 1.2 us—a ratio of 2:6 to one. This ratio is probably valid today under the assumption the tables are cached. Table 3.11 shows the normalized number of instructions executed per unit of time. This metric is a relatively good measure of the TLB's effectiveness; the larger the number, the more effective is the TLB. Table 3.11 also shows the TLB miss ratios as the size is varied.

These data indicate that an associative TLB with 8 sectors is probably large enough, nor do these data support the large TLBs found in modern memory systems; the reason being that the GE-645 is a microprogrammed machine that takes many clocks to execute an instruction, unlike modern pipelined processors with CPIs of 1–2. Thus, with modern pipelined processors, name translation delays have a greater impact on overall performance. Additional information on the impact of TLBs on performance is found in [SHEM66].

3.4.4 *Instruction Support for TLBs*

TLBs are generally considered to be invisible to the program, as with early caches, and serve only to reduce the time of name translation.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000216

Unfortunately, this simple view is not completely correct. Recall the use of processor instructions for enforcing MLI, described in Table 3.5. Two classes of instructions are required. First, some of the TLB registers must be loaded explicitly under program control; second, the TLB must be invalidated in order to maintain coherency. The complexity of invalidation is illustrated by the PowerPC 601 *translation lookaside buffer invalidate entry* (*tibie*) instruction, which is used for MLI control.

A name translation of the page name of the *tibie* instruction (a supervisor-level instruction) is attempted. If there is a hit on the TLB, the TLB entry is invalidated by setting the valid bit to zero. A TLB invalidate is also broadcast on the system bus so that coherency can be maintained across the system.

With a multiprocessor system, the broadcast *tibie* must be in a critical section controlled by software locking so that only one *tibie* can be issued at a time. Resynchronization is established by issuing a *sync* instruction after every *tibie* at the end of the critical section. When a processor receives a broadcast *tibie* instruction, it halts the execution of new load, store, cache control, and *tibie* instructions; waits for the completion of all memory operations; and then invalidates both sectors in the user TLB.

3.5 Virtual Memory Accessing Rules

This section discusses the rules that are followed when a virtual memory system is accessed for reads and writes. The effect of a cache on this process is ignored; however, the combination of a cache with virtual memory is addressed in Chapter 4. The reader should note that all of the design ideas and much of the published performance data on virtual memory systems are from an earlier time when small memory was the rule. Today, with significantly larger memories, these designs may no longer be appropriate. Caution should therefore be exercised in the use of specific design data.

3.5.1 *Read Accesses*

The various policies that govern the actions when a paged virtual memory is accessed are discussed in this section. The various design issues involved are described here with illustrations taken from an actual virtual memory system. The basic steps in performing a read access into a virtual memory system consist of the following.

1. Determine if a referenced page is in real memory; detect for a page fault.
2. On a hit (the referenced page is determined to be in real memory)

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000217

translate the virtual page name to the page frame address and fetch the AU into the processor.

3. On a miss (the referenced page is determined to be absent from real memory):

(a) Determine if a page must be evicted because of a capacity miss; identify the page to move; evict the page to the disk. This policy is known as the *replacement policy*.
(b) Depending upon the organization of the virtual memory, a determination of the page frame to receive the loaded page may need to be made. Fetch the page from the disk into the real memory and adjust the tags and valid bits in the name translation tables. This policy is known as the *placement rule*. Translating the virtual address to the disk address is discussed in Section 3.5.9.
(c) Complete the read of the AU into the real memory.

Note that if the system is multitasking, a new task may be swapped in during this process. There are commonly three successful name translation paths through a virtual memory with a TLB and a cache; the fourth path is the page miss path. These paths are shown in Table 3.12.

Path 1. The page frame address is found in the TLB.

Path 2. The page frame address is not in the TLB but is found in the page tables that have been placed in the cache. After the reference, the TLB is updated with page name translation information for use by a future reference.

Path 3. The page frame address is found by table walking in real memory. After the reference, the cache and the TLB are updated with page name translation information.

Path 4. The translation misses on the TLB, cache, and tables in memory. This is a page fault path and requires the allocation of the missed page into real memory. All translation tables are updated.

The three successful cases above are for the name translation only. The assumption is made that a name translation miss on a page that is resident in real memory is not permitted. For Path 4, the total failure

Path	Page Name Translation Path Hit/Miss		
	TLB	Cache	Memory
1	Hit	X	X
2	Miss	Hit	X
3	Miss	Miss	Hit
4	Miss	Miss	Miss

TABLE 3.12 Name translation paths.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000218

of name translation indicates that the page is not in real memory and results in a page fault.

3.5.2 *Allocation*

This section discusses the issues of allocating new pages or segments into real memory following a page fault—a problem that can be different from late select cache allocation. Recall that congruence-mapped, n-way set associative (including direct) late select caches dictate the sector into which a new allocation must be placed. Only the block within the sector is open to choice. Associative caches, on the other hand, can allocate into any segment with the choice based on one of the replacement policies discussed in Chapter 2.

Virtual memory page or segmentation organization is early select, which is an associative accessing process. Therefore, allocation of pages to real memory is a matter of (1) finding a vacant location and (2) if a vacant location cannot be found, creating a vacant space using one of the policies discussed in Section 2.1.7. In addition to the allocation problems associated with finding space for a page frame or a segment, the name translation information must be allocated to the name translation tables as well. The problem here is quite similar to the allocation problems of caches, noted above. The organization of the translation tables dictates the allocation of name translation information.

The techniques for performing allocation in virtual memory systems have their historical roots in the problems of allocating files to disks, discussed in many texts on operating systems. Also, because allocation is highly dynamic and interactive in LISP systems, research in these systems has made significant contributions. Even interactive programming environments such as Lotus 1-2-3 must have solutions to the requirement for dynamic allocation and de-allocation of segments.

Page Allocation

The procedure for allocating pages to page frames is the same regardless of the name translation method. That is, either direct or inverted mapping systems use the same procedure. The operating system maintains a *free page frame list* (a linked list of vacant page frames in real memory) that contains the starting addresses of all unallocated pages. The list can be maintained as a linked list or a LIFO stack. Upon a page fault, the free page frame list is searched; if there is a free page frame, it is allocated. Recall that because a paged virtual memory is early select, congruence mapped, and associative, any free page frame can be allocated. If a page frame is not available, a replacement policy is then invoked.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000219

There can be additional policies imposed by the operating system on page allocation from the free list. For example, should there be reserved page frames for the operating system and the user? If the system is multiprogrammed, should there be page frames reserved for each of the resident programs? These issues are beyond the scope of this book but are treated in books on operating systems.

The hardware must support reading and writing the free page frame list. Thus linked list or stack manipulation is needed. Most processors designed today provide for stack manipulation, which is the implementation of choice. Processors that have been optimized for list or LISP processing use linked lists for the free page frame list. The size of the free page frame list is not excessive, as there must be, at most, one entry for each real memory page. For a system with a 4-Kbyte page and 16 Mbytes of real memory the free page frame list has only 4K entries. If an entry is 4 bytes, 4 pages are occupied by the free page frame list, approximately 0.1% of the total real memory space.

The question of whether or not the free page frame list itself is paged or locked into an unpaged region of real memory is addressed in Section 3.5.4. Recall that most virtual memory systems today provide for an identity page name translation for operating system access to memory. In addition, some systems provide block name translation facilities to enhance the management of the operating system in virtual memory space. A block composed of a number of pages eliminates much of the paging that could occur if only pages are implemented.

Segment Allocation

For the discussions to follow, it is helpful to remember that space in memory, either page frames or space for segments, can be in one of three states.

1. *Free*. The space is vacant and available for allocation.
2. *Allocated*. The space is occupied by a valid segment, either clean or dirty.
3. *Garbage*. The space is occupied by a segment that has been deallocated.

Segmented systems pose additional, and significant, allocation problems to those of paged systems. There must be a way to locate a contiguous region of free space in real memory that is equal to or larger than the segment to be allocated. This requirement means that a *free space list* (called spaces because they are not yet allocated to segments) must contain not only the starting address of free space but the length of each space as well. Note that the size of a free space list is not bounded, as is a free page frame list, because in the limit, every space can be one

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000220

addressable unit and the free space list would occupy all of real memory! Segments can overlap for sharing, thereby creating further allocation problems. Allocation for nonoverlapping segments follows three steps.

1. Find a free space that is large enough to receive the allocated segment.
2. If step 1 fails, determine if there is enough total free space in real memory. If there is, compact enough available free space to create a space for the segment to be allocated.
3. If steps 1 and 2 fail, find an allocated segment that is large enough and can be evicted based on criteria discussed in Chapter 2.

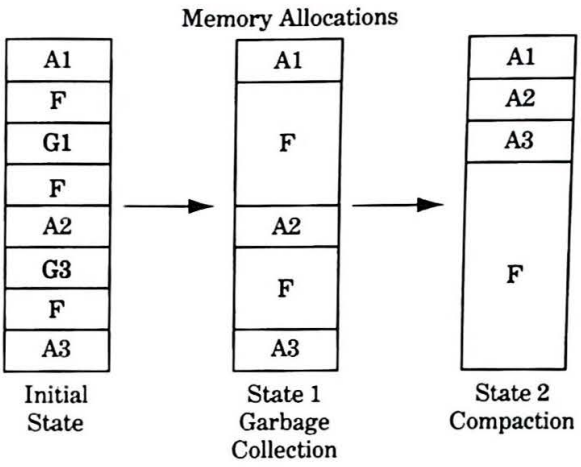
Step 1. This step is similar to page allocation. A *free space list* is maintained that lists the starting addresses and length of all free space. A number of algorithms have been proposed [KNEE68, DENN70] for segment allocation from the free space list. A detailed discussion of these algorithms is outside the scope of this book; nevertheless, four algorithms are

1. *First-fit*. Search the free space list, and find the first space into which the segment will fit regardless of the efficiency of space use. Place any unallocated portion of this space on the free space list.
2. *Best-fit*. Search the complete free space list, and find the free space that gives the most efficient use of space. Place any unallocated portion of this space on the free space list.
3. *Worst-fit*. Search the complete free space list, and find the free space segment that gives the least efficient use of space. Place any unallocated portion of this space on the free space list.
4. *Buddy*. Free space is initially divided into groups having even power of two AUs as with pages. When a page is allocated one of three allocations is made: (i) half the group with the other half placed on the free space list; (ii) the full group; or (iii) two groups, four groups, etc. This requires that all free space groups and allocated segments must be even power of two in length.

Knuth [KNUT68] reports on simulations that show that the first-fit policy gives the best results, defined to be that allocations continue longer before Step 1 fails and a garbage collection pass is required.

Step 2. When a new segment is to be allocated the total free space may be sufficient to hold a segment but the space may not be contiguous, which is a condition required for segmentation. A full discussion of the many techniques for compacting fragmented free space and de-allocated space, generally known as *garbage collection*, is outside the scope of this book. However, I want to provide a brief discussion of the issues involved. A comprehensive survey of garbage collection techniques can

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000221



A: Allocated, F: Free, G: Garbage.

FIGURE 3.39 Garbage collection.

be found in [COHE81]. Note that not all algorithms for managing free space are known as "garbage collectors."

One of many types of compacting garbage collection systems is illustrated by the example shown in Figure 3.39. In the initial state the map of memory shows areas that are of the three states noted above: free, allocated, and garbage. The state of each area is denoted by means of a tag in either the space lists or by two control bits that impart *coloring* to the three states.

The initial state of the memory is shown and consists of three allocated spaces, three free spaces, and two garbage spaces. The first step, to State 1, in the garbage collection process searches out the garbage spaces and turns them into free space by placing these spaces on the free space list. The tag bits are also changed as required, and no movement in memory takes place. The second step moves both the allocated and free spaces into two compact regions of real memory. Thus all of free space is in one compact space and is available for allocation.

Step 3. If garbage collection and compaction do not yield a space that is large enough for allocating the new segment, an already allocated segment must be evicted. The algorithms for making the selection for eviction are based on (1) selection of a segment of appropriate size, (2) the usage criteria, and (3) the choice of a modified or nonmodified segment (clean or dirty).

If a modified segment is chosen for eviction, it must be copied back onto the disk. The free space thus made available is placed on the free space list and allocation Step 1 is initiated. In the chosen segment is not

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000222

modified, it is only necessary to place the segment on the free space list and proceed with allocation Step 2.

From the above brief discussion of segment allocation it is clear that considerable operating system overhead is required. Hardware assist for this process consists of stack and block move support. The amount of overhead is proportional to the size of the segment being allocated and the state of the memory prior to the allocation. This overhead can be observed in Lotus 1-2-3 by the difference in time taken to return to the READY state after allocating small and large numbers of cells by, for example, a COPY command.

3.5.3 *Write Access*

Write accesses proceed just like read accesses. This is because virtual memory systems are early select and the presence or absence of a page or segment is determined before real memory is accessed.

If the write access is a hit, the write is made to the AU in real memory and the page table dirty bit, if used, is set. A dirty bit is used in conjunction with the valid bit to manage the de-allocation and allocation of pages or segments in real memory. Unlike caches, there are no known implementations of the equivalent of a write through strategy for a virtual memory system, because the time to write a single AU to the disk is quite large and would block the disk from other activity. Write buffers would not eliminate this problem. Thus, all virtual memory systems use a write back strategy.

If the write access is a miss, the missed page is allocated into real memory and the write operation is performed again. There are no known no-write-allocate strategies, like those of caches, with virtual memory systems. The reason for this is the same as for not using a write through strategy: the very long transport time makes writing a single AU to disk unjustifiable.

3.5.4 *Location of Tables*

The first virtual memory machine, the Atlas shown in Figure 3.15, had its page table in an associatively searched hardware register file. As the size of the virtual address space has increased, larger tables have been required and placed in real memory.

One might ask if the tables in real memory are addressed in real memory space or virtual space? To my knowledge, (except for Multics) the tables are never placed in just any virtual space but in dedicated pages (identity name translations) of virtual memory space. If the page tables are paged, there can be a deadlock in paging as one name transla-

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000223

tion is forced to wait on another name translation that is waiting on the first, and so on. Thus, access to page tables is mapped by identity and the time for name translation via a page table tree is eliminated. The RS/6000 accesses its table in real addresses without the identity translation step [IBM90].

In addition to storing the page tables in real memory (or an identity translation), if the system has a data cache the page tables may be cached as well. Caching of page tables can further reduce the name translation time, as is described in the model found in Section 3.6. There is no known published data on cache P_{miss} when storing page tables. It may well be that a page table cache could be a useful variation of caches. On the other hand, any chip area needed for this cache may be more usefully devoted to larger TLBs.

Section 3.4 discusses the Motorola MC88200 [MOTO90a], which provides a dual TLB facility. A separate TLB known as the *block address translation cache* (BATC) translates the virtual address into a pointer into a 512-Kbyte block of real memory. Management of the BATC is under program control as there is no recourse to a multilevel translation table if there is a TLB miss. With a miss, the operating system is invoked to perform all of the management functions and is supported by some special instructions for loading the BATC.

3.5.5 *Table Walking Methods*

The MC68040 multilevel direct page tables are stored in real memory as shown in Figure 3.40. If a successful name translation cannot be accomplished in the TLB, the address is translated with the memory resident tables. The various address fields and control bits of the table entries are allocated into groups of the AUs. The use of real memory for page tables leads to the requirement that the tables must be traversed by fetching and decoding the entries of each valid level of the hierarchy, a function called *table walking*.

Early systems such as the GE-645 Multics [DENN72, ORGA72, MATI80] and at least one contemporary system, the MIPS R2000/R4000 [MIPS87], perform table walking with a program using the normal instruction set of the processor. For these systems, referencing each level of the page table requires a sequence of instructions such as

1. Load page entry.
2. Select valid bit.
3. Branch to 8 on not valid.
4. Select pointer field.
5. Extract Index value from address.
6. Concatenate to form memory address.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000224

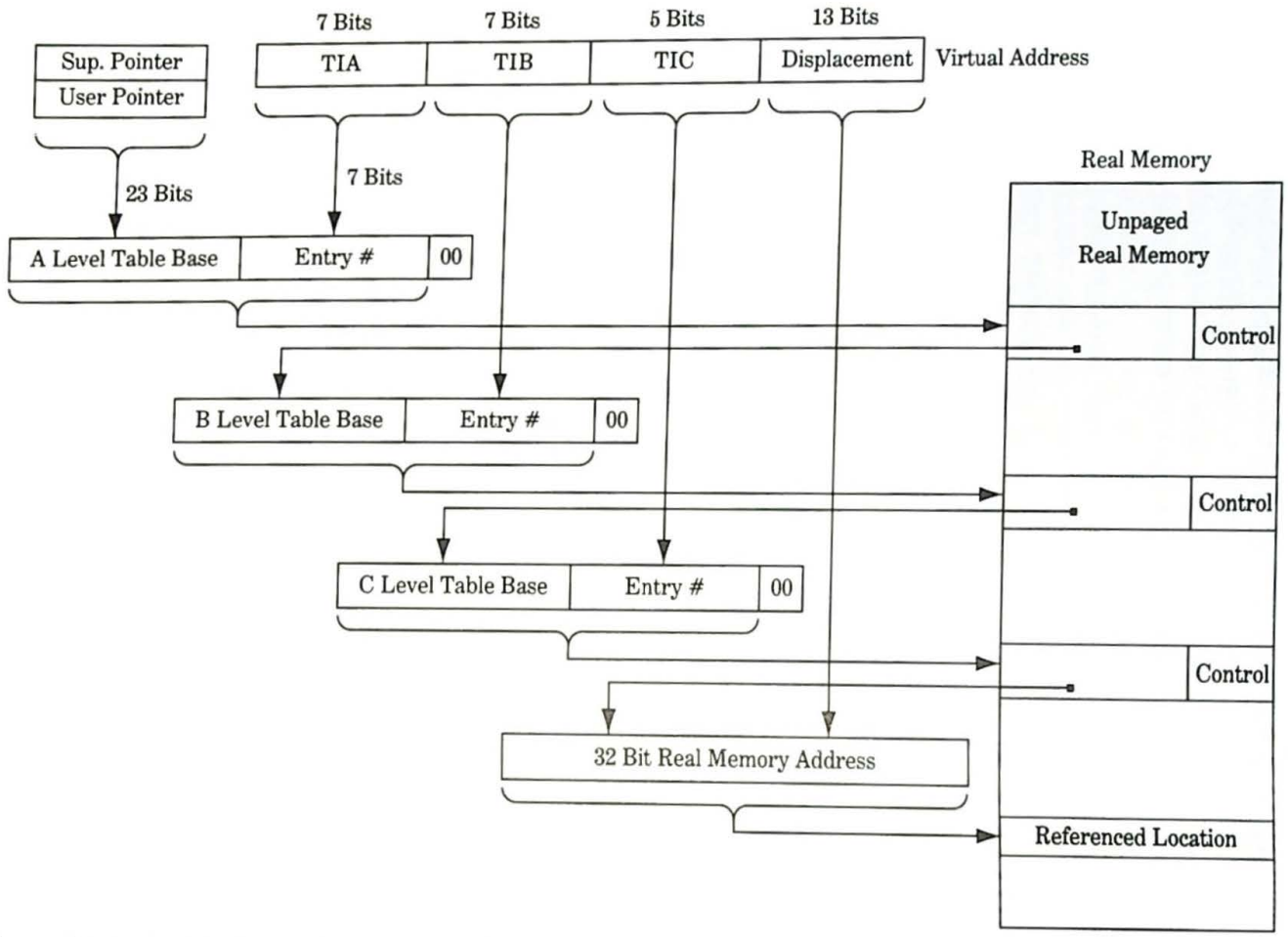


FIGURE 3.40 MC68040 multilevel page table storage.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000225

7. Leaf node? no, branch to 1; yes branch to 9.
8. Fault exit.
9. Continue.

Each level of the page table will have an overhead of approximately eight instructions. A three-level system has an overhead of approximately 24 instructions or 30 memory cycles for each memory reference that is not translated in the TLB. The actual time required for a translation is a function of whether or not the page tables are cached and whether or not there is an instruction cache that may be holding the short program. Also, if the program is paged and the page is not "locked in" there could even be page faults during a table walk.

This software approach is still possible, but many modern memory management units have special logic that performs these functions. The logic is known as *table walking hardware,* and it significantly reduces the time required to access the page tables. Motorola calls this operation *translation table searching,* a function implemented in microcode.

The page table walking overhead burden is reduced by incorporating a state machine controller that implements the table walking function; an example of this controller is found in the MC88200 Memory Management Unit. Using hardware table walking eliminates most of the memory cycles and the possibility of cache misses and page faults. The net result is that the time required to translate is approximately one memory cycle per level of translation.

Inverted page tables, such as those used in the RS/6000, use a form of table walking in their linked list structure. For these systems, the control of searching down the list is vested totally in hardware. There are no known systems that are software-based. The reduction in weighted average name translation time with hardware over software table walking is dramatic and is modeled for the VAX 11/780 in Section 3.6.

3.5.6 *Instruction Set Support for Virtual Memory*

As I noted previously, a major portion of the work (but not the ti.. associated with servicing a page or segment fault is performed by the processor. Instruction set support is found in the three general areas of (1) table walking, (2) table management, and (3) interrupt support. For multitasking systems, the time to execute a table walk is important if it cannot be overlapped with other processing. For single-user virtual memory systems, the table walk time is small with respect to the disk time and can usually be ignored.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000226

Table Walking Support

Table walking with software requires instruction set support to execute the algorithm noted above. The instruction set should include the usual field extraction, logical operations, and testing instructions found in general purpose instruction sets. In general, no additional instructions are required. Table walking code can be provided as a subroutine that is called when there is a miss on the TLB.

If page faults are frequent, a special subroutine call/return, designed for this purpose only, can eliminate most of the overhead for this rather special subroutine. It may also be helpful if the subroutine is locked into the cache to eliminate instruction cache misses.

Table Management Support

Table management support is required due to the large number of tables that must be loaded and stored under program control. The extension register(s) portion of the virtual address are loaded by the operating system with special load instructions. Special load instructions are also necessary for the segment registers for those systems that use this method for extending the effective address.

The IBM RS/6000 is one example of the special load instructions. There are a number of hardware registers collectively named *storage control registers*. First, there are 16 *segment registers* (SR); the most significant bit of each signifies if the segment is an I/O or processing (normal) segment. Another register is the *transaction ID register* (TID); this register contains a 16-bit segment identification number. Two other registers are named the *storage description registers* (SDR0 and SDR1). Special instructions are provided to load all of these registers. Note that these registers are never modified by a translation or a side effect of a translation; however, they can be modified as the result of a page fault. Thus, a store instruction must also be provided to store these tables for a context switch. Loads and stores are accomplished with move to sri (*mtsri*) and move from sri (*mfsri*) instructions that are moves between the special registers and the general purpose registers.

An inverted hash access system has other tables, the hash anchor table and page tables, that must be loaded and stored under program control. These tables are located in real memory, and the usual instructions that manipulate memory locations serve to load and store these tables. TLBs and other cachelike translation tables must be loaded and stored as well. The need to store the tables is a consequence of the fact that these tables can be modified between context switches.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000227

Page Fault Interrupt Support

An executing program references memory with name translations via the TLB and the page tables assuming that the referenced pages are in real memory. If the page is not in real memory, the present bit in the translation system will trigger an interrupt, an action known as a *page fault* or *memory exception trap*. When there is a page fault, there are a number of tasks that must be accomplished. The following list is for a uniprogramming environment. Multiprogramming or multitasking requires other steps and is outside the scope of this book.

1. Trap the operating system.
2. Save the user and process state.
3. Write out a "dirty page" if necessary (virtual memory uses write back).
4. Allocate the new page to real memory.
5. Translate the virtual address to a disk file address.
6. Issue a read to the file system.
7. Read page into the allocated location.
8. Update translation tables and TLB.
9. Restore user and process state.
10. Resume processing.

The actual time required to execute these steps is not as great as the length of the list would suggest. For example, assume eight of the steps (not counting the transport times of steps 3 and 7) require an average of 1,000 instructions and each instruction requires 50 ns. The total time to service a page fault interrupt is 0.4 ms, a trivial part of the one or two 20-ms to 30-ms disk latencies.

3.5.7 *Memory Access Control*

This section discusses two issues: *protection* and *proper access*. Protection has to do with assuring that a reference to memory is authorized. Proper access has to do with the correctness of the access, as in not attempting to execute a floating point datum as an instruction. The issues of protection of objects or abstract data types and system access security are outside the scope of this book, and references to these subjects can be found in most books on operating systems.

Protection

Early research into the implementation of multiuser time sharing systems pointed to the need to protect programs and data from unauthorized access [DENN65, GRAH65, WILK68, GRAH72]. For reasons of

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000228

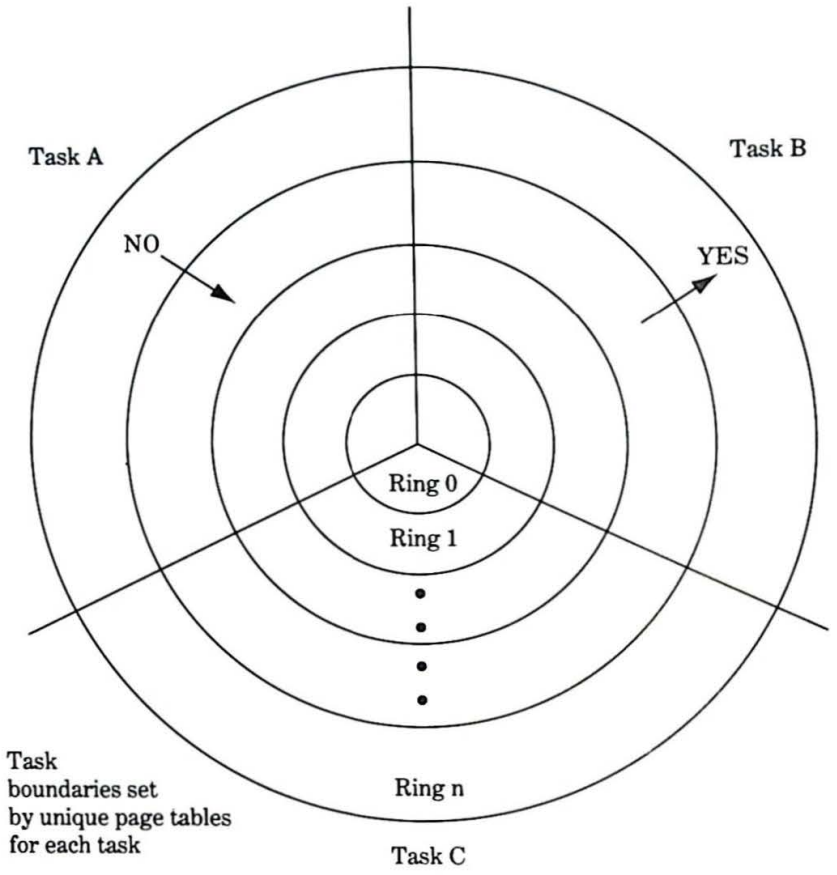


FIGURE 3.41 Rings of protection.

efficiency, however, it is desirable to permit sharing between users of programs and some data, a desiderata that introduced the need for protection. Graham [GRAH72] lists seven levels of protection for multiuser time sharing systems; each level requires additional hardware support for their implementation. The requirement for sharing and protection is just as important in processors that execute a number of tasks (multiprogramming or multitasking) that share resources.

The view of protection we have today has its roots in the MIT Project MAC and Multics. Graham [GRAH65] published the idea of *rings of protection,* which is shown in Figure 3.41. Briefly stated, Graham's idea is that a "process executing in ring i has no access whatever to any segment in ring j, where $j < i$." This view of protection leads immediately to the requirement that somewhere in the name translation system there must be a tag to denote the level of protection. Because control bits are somewhat limited in the page table entries, there is a limit to the number of rings of protection.

For paged virtual memory systems, protection is exerted at the page

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000229

level, or at the block level if block translation is employed. Examples of the use of control bits that provide protection can be found in a number of processors. The i386, i486, and Pentium devote two bits in the descriptor tables, which provide four levels of protection. These levels are: kernel, system services, custom extensions, and application. Two protection bits are also used in the page frame table entry of the RS/6000. In the most general implementation of rings of protection, if control is being transferred to a ring other than the current one, an interrupt occurs and the operating system is invoked to perform the proper housekeeping tasks.

Proper Access

Protecting memory from an improper access is the second issue of access control. One task should not be able to interfere with another task without mutual agreement. Four possible ways [SITE80] of providing memory protection are

1. bounds registers,
2. memory keys,
3. translation table keys,
4. distinct addresses spaces.

These four protection methods are discussed individually below. . should be noted, however, that combinations of the four methods are implemented in some specific computers.

Bounds registers were first used in nonvirtual memory machines. Bounds registers provide two pointers: one to an upper address limit, the other to a lower address limit. Bounds can also be specified with a starting address and length. If a reference is found to be outside these bounds, an error interrupt is signaled. The bounds registers are loaded and stored by the operating systems using privileged instructions.

For a system that will have a number of tasks resident in memory at one time, there may be a number of bounds register pairs, one for each task. The register pairs are addressed by a pointer that is set when a task is switched in. The use of bounds registers assumes that a task, program, and/or data, is allocated within a contiguous address space. Bounds registers permit sharing between tasks of an address space by identifying overlapping spaces by the bounds registers. Bounds registers, however, present restrictions with, for example, a single word containing a semaphore that is accessed by a number of tasks. The semaphore would have to be in the highest address of one task and in the lowest address in a second task. Access by three tasks is difficult.

For paged systems, protection is at the page or block level and for segmented systems at the segment level. For example, the segment

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000230

length field of a segment table operates in the fashion of a bounds register. An example of this is the block translation system of the PowerPC 601 that has a block partition of memory that is similar to a segment. A block name is presented to the block TLB and is checked for being within the starting address and the length of the block. Block length is encoded with six bits, rather than having an upper address, in the BAT registers discussed previously. The length of the block is a binary progression: 128 Kbytes, 256 Kbytes, 512 Kbytes, . . . , 8 Mbytes, an encoding designed to facilitate length checking [MOTO93].

Memory keys specify the type of access that is permitted to a memory location, page, or segment. Typical access types are: Unlimited, Read Only, Execute Only, and/or a task I.D. The executing program key is usually found in the processor status register that, in combination with the type of access, is compared to the "lock" associated with the memory and found in the name translation system. If the key matches the lock, the requested access is performed. There is usually a "skeleton key" that can be used by the operating system to access all of memory regardless of the key associated with each of the keyed memory spaces.

The memory keys are usually contained in the name translation maps and/or the TLB. Thus, an improper access is terminated before the read or write is actually performed. Memory keys permit access to noncontiguous page frames in real memory, unlike bounds registers. Sharing is more difficult, however, as sharing is at the block level. Shared variables may need to be relocated with a context switch increasing context switch overhead. If different accesses are legal for different tasks, the protection fields in the TLB must be changed when tasks are switched.

The PowerPC 601 illustrates the use of memory keys. The supervisor mode has the keyed accesses of read/write and read only, while the user mode has the keyed accesses of no access, read only, and read/write. The operating system can block user programs from accesses by evoking the no access key. This description is not complete; additional information can be found in [MOTO93].

Translation table keys provide protection by having an access table for each task. These tables, one for each task, contain the legal access methods for each page of real memory and permit different legal accesses for different tasks with a minimum of overhead. As there are multiple tables, only a switch of a pointer is needed to select the table for the new task. Images of the keys are found in the page tables and are checked along with the check of the virtual address tag.

Distinct address spaces can be used to provide protection by assuring that different tasks do not share the same virtual address space. This protection is accomplished by making the task name a part of the virtual address. With the task name in the most significant bits, there can be no

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000231

overlap between addresses in different tasks. One example of a distinct address space system is found in the ix86 processors that have segment registers associated with each type of access, for example, code and stack.

3.5.8 *Choice of Page Size*

This section addresses four tradeoffs in determining page size.

1. Reducing internal fragmentation.
2. Reducing disk/memory transfer time.
3. Matching the page size to the levels of the name translation tables.
4. Reducing the miss ratio.

Internal fragmentation was introduced in Section 3.1. When a page size is selected, there is a waste of usable memory space. First, because the allocation block is not an exact multiple of the page size, internal fragmentation occurs. Internal fragmentation is reduced with small pages because the wasted space is limited to a fraction of a smaller page. Second, when a program and its data is allocated to real memory, space must be provided for the page table entry of the allocated page. Page table space is not available for the program and data; it is pure overhead space. As the page size decreases, more pages must be allocated and the space used for page table entries increases. Thus, what page size will balance these two counterforces and minimize the loss of memory? Early work on this design problem, and the model derived below, is found in [WOLM65], who attributes the first version of the model to J. B. Kruskal. This model determines an optimum page size to minimize wasted real memory.

Assume that the number of AUs that must be allocated for a program (instructions and data) is n; the page size is p; and a AUs are required for each page table entry. A direct one-level page table, in memory, is assumed. Thus, the use of main memory by page tables and internal fragmentation represents a loss of real memory to the user.

Memory loss = loss due to internal fragmentation + loss for allocating a page table entry,

$$\text{Loss due to internal fragmentation} = \frac{p}{2},$$

$$\text{Loss for allocating a page table entry} \approx \frac{an}{p},$$

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000232

$$\text{Memory loss} = C = \frac{p}{2} + \frac{an}{p}.$$

The derivative of C with respect to p is taken and set to zero as

$$p_{\text{opt}} = \sqrt{2an}.$$

This model shows that in order to minimize the loss of real memory due to internal fragmentation and page table entries, large pages are best for large programs and data sets that are becoming more common. However, with lower cost memory resulting in larger real memory, internal fragmentation is not viewed to be the problem it was three decades ago.

As described in Section 3.3.1, today, page tables are allocated into pages themselves, making this model incorrect as the first allocation requires a full page. With the internal fragment loss of one-half page and with page tables allocated to pages that lose one-half page, the total loss is one page and is invariant with page size or the number of allocations. Thus, the size of a page is more-or-less not an issue as far as real memory loss is concerned.

Reducing disk/memory transfer time is the second tradeoff option for determining page size. Pages transported between the disk and real memory incur a significant overhead due to the long access time of the disk. The time per byte transferred is reduced as the page size is increased. As with the discussion of block size for caches found in Chapter 2, if the page size is very large, the transfer time per byte approaches the data transfer time with the access time overhead totally prorated. Note, however, that page sizes today are far from this limit. The larger main memories of modern computers and the reduction in disk access and transfer time, discussed in Chapter 5, tend to drive the page size upward.

Matching the page size to the levels of the name translation tables is a determining factor. The choice of page size can determine the number of levels in a direct multilevel page table. This issue is discussed in Section 3.3.1. Larger page sizes require fewer levels for name translation.

Reducing the TLB miss ratio is another consideration in the selection of a page size. The larger the page size, the smaller the TLB miss ratio and the smaller can be the TLB. This is the reason for the use of two TLBs for two different page sizes in a number of processors such as the MC88200 as described in Section 3.3. Larger page sizes led to smaller TLBs, which is an important consideration in some designs. A large page has another virtue because a cache can be placed in virtual memory

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000233

space. That is, the displacement in the virtual address addresses the cache while the virtual name translation is being performed. This cache architecture is discussed in Chapter 4.

From the above discussion, it can be seen that page size selection is a tradeoff process to achieve a balance between transport time, internal fragmentation, and name translation time. It is interesting to note that the flexibility in page size provided by the Motorola 6851 has been largely abandoned by designers for a fixed page size. However, the IBM RS/6000 gives a choice of two page sizes while the Digital Alpha provides page sizes of 8 Kbytes, 64 Kbytes, 512 Kbytes, and 4 Mbytes. The evolution of virtual memory systems has arrived at the consensus of a 4-Kbyte page size. The page sizes of some contemporary processors are given below.

VAX	512 bytes
IBM S/370	4096 bytes
RS/6000	4096 bytes
PowerPC 601	4096 bytes
i486	4096 bytes
Pentium	4096 bytes
MC88200	4096 bytes
MC88100	4096 bytes
R2000	4096 bytes
SPARC	8192 bytes

3.5.9 *Addressing the Disk*

This section discusses the way that the disk address is found when a new page is allocated following a page fault. Recall from Section 3.1 that the required name translation is referred to as using Map 3. This name translation problem is similar to the issues discussed in the preceding sections, which deal with translating virtual names to page frame addresses. Here we need to translate a virtual name to the disk addresses of a page, a process that requires a table(s) to hold translation information. Note that as with virtual memory, only page names need translating as the disk is never addressed at a lower level.

There are two ways of providing translation tables. First, the disk page address is contained in the page table entry [TANN84]. With this approach, the leaf page table entry would contain not only a field for the page frame address of the page in real memory but also a field that contains the address of the page on the disk. These disk address fields

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000234

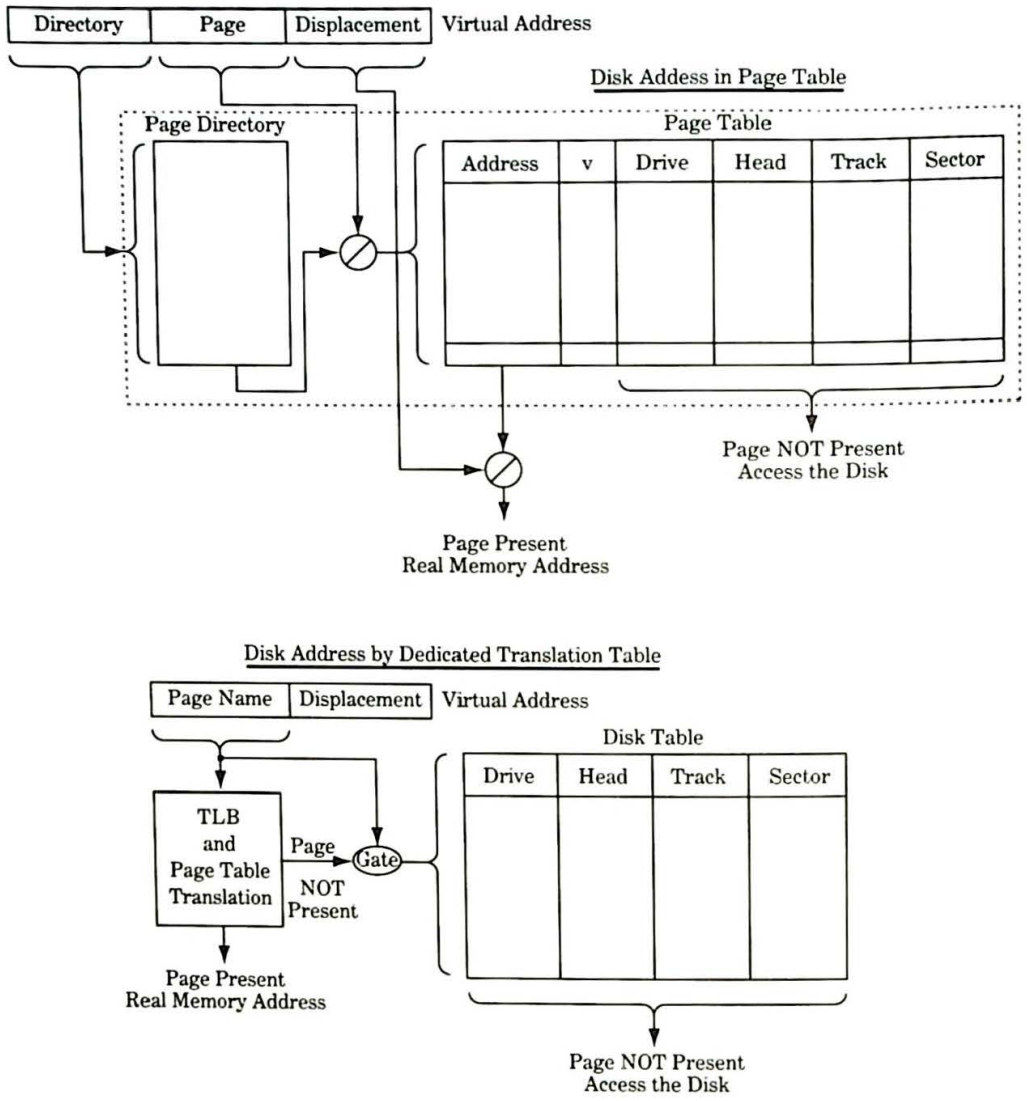


FIGURE 3.42 Disk translation table placement.

would have information on drive, head, track, record, and the like, which as Figure 3.42 indicates, is a relatively large number of bits.

A variation of this technique is found in the i386, i486, and Pentium [INTE90]. Because multilevel tables are direct with an entry for each page in virtual memory, the page table entries can be either the translation address or the disk address (MAP 2 or MAP 3). Only one map is valid at a time; the valid bits indicate which address is stored. When there is a page fault, the disk address is used to access the disk and is then replaced with the page frame address and the various control bits.

Neither version of the technique of placing disk address information

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000235

in the page tables is suitable for inverted page table name translation. The reason is that an inverted page table contains only the name translation information of pages resident in real memory. Thus, another technique for disk address translation is needed.

This second technique, also shown in Figure 3.42, starts anew, upon a page fault, with the virtual address that caused the page fault, and translates that address via a dedicated translation table (MAP 3). With this method, the same number of bits are required to address the disk, but they are managed in a separate table. The translation of the virtual address into the disk address can be via direct multilevel translation or the file management system of the operating system.

3.6 Virtual Memory Performance Issues and Models

There are two major performance issues with virtual memory systems: name translation time and page fault time. The time to service a page fault has been discussed previously; thus this section deals only with name translation time. A rather old annotated bibliography of virtual memory performance measurements is found in [PARM72].

Name translation time is the weighted average time needed to translate a name and is added to the access time of main memory for systems without a cache and is added to cache access time for real address caches, as with the VAX 11/780 discussed in Chapter 4. Name translation by accessing tables can be a lengthy process, and multilevel translation tables create a significant performance problem. Each virtual memory reference requires one or more real memory references to access the tables, and if page table walking hardware is not provided, a number of instructions must be executed by the processor to interpret the contents of each page table entry.

A simple model for the time required to make a name translation is given below. This model is based on the unified TLB design of the VAX 11/780 [SATY81, CLAR83, EMER84, CLAR85]. The steps of translating an address are

1. Search the TLB and use the page frame address on a hit.
2. If the TLB search fails, the address is translated with cached page table information. On a translation hit, the translation information is placed in the TLB and the name translation is restarted in the TLB. Note that while a normal cache access requires a translated address, microcode has untranslated access to the cache for name translation purposes.
3. If the cache translation fails, the name is translated with page table

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000236

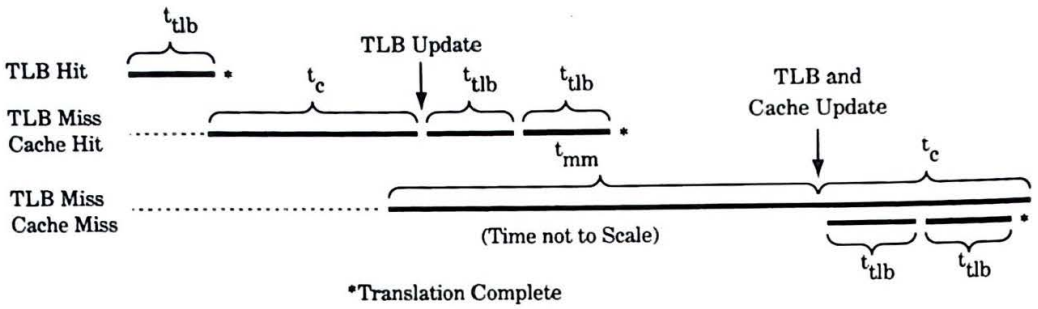


FIGURE 3.43 Name translation time.

information found in real memory. The translated information is placed in the cache and, in parallel, the TLB. The name translation is restarted in the TLB. Note that microcode table walking is provided for untranslated access to the page tables in real memory.

The VAX 11/780 memory performs name translation on data accesses and instruction accesses that miss the small instruction TLB. A timing diagram of the name translation process is shown in Figure 3.43.

The weighted average translation time (WATT) for a successful translation is

$$\text{WATT} = f_t \times t_{tlb} + f_c(t_c + 3t_{tlb}) + f_m(t_c + 3t_{tlb} + t_{mm})$$

where

f_t = fraction translated in TLB,
f_c = fraction translated in cache,
f_m = fraction translated in memory,
t_{tlb} = time for a TLB translation or update,
t_c = time for translation in the cache and a cache cycle (read or write),
t_{mm} = time for a translation in main memory.

The following assumptions are made:

$$2t_{tlb} \ll t_{mm}, \qquad f_t + f_c + f_m = 1.$$

Note that the fraction of translations performed at each level are equivalent to global hit ratios of multilevel caches. Clark [CLAR85] provides values for the parameters for this model derived from simulation. These parameters are

$$f_t = 0.97, \quad f_c = 0.012, \quad f_m = 0.018,$$
$$t_{tlb} = 1 \text{ clock}, \quad t_c = 6 \text{ clocks}, \quad t_{mm} = 18 \text{ clocks}.$$

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000237

Using these parameters in the model gives a weighted average name translation time of 1.56 clocks. For the VAX 11/780, the average number of clocks per instruction, without TLB miss delays, is approximately 10. This time includes the one clock for translating the data addresses through the TLB. Thus, for the parameters above, 0.56 clocks must be added to the 10 clocks of the basic execution time. TLB misses, therefore, add approximately 6% to the total execution time. This result compares favorably to the measured results of [CLAR85] of 5.1% to 8.0%, for various benchmarks. From the same parameters, 19.8 clocks are spent for each miss on the TLB. This result also compares favorably to the measured results of 21.0 to 22.1 clocks [CLAR85].

A name translation time of 1.56 clocks is a reasonable penalty for a processor that spends 10 clocks per instruction. For pipelined processors, however, discussed in Chapters 6 to 10, the clocks per instruction is approximately 1.5 (without translation delays). If there is an additional 0.64 clocks for name translation, the total execution time will be increased by approximately 30%. This is a major reason why real addressed caches are not used in most pipelined processors (described in Chapter 4).

Another issue can be addressed with this model. What is the performance benefit of hardware table walking? Assume that t_{mm} is 48 clocks rather than 18 with table walking. The weighted average translation time is 3.08 clocks rather than 1.56. The performance impact on the VAX 11/78 would be approximately 20% rather than 6%, a clear performance advantage for a processor with a cache addressed with real addresses.

3.6.1 *Published Page Fault Ratio Data*

This section discusses the identified published page fault ratios; unfortunately, little current information has been found. Rather old information on page fault ratios, as a function of page and real memory size (expressed as number of allocated pages), is provided by [CHU74] and excerpted in Table 3.13. Note that this data was taken in the era of small memory on the Sigma 7 and may not be representative of contemporary systems.

The presentation of this data is similar to other tables displaying miss ratio data. Diagonals from lower left to upper right are equal real memory sizes. This data shows very interesting properties. First, and this is to be expected, for any page size, as the size of real memory is increased permitting more pages to be allocated, the page fault ratio becomes constant at the value required to page in/out the working set. Once a working set is in memory, no further paging is required until the working set is paged out to the disk. When the memory is large, a larger

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000238

Memory Partition in Pages	Page Size in 24-Bit Words			
	64	128	256	512
8				.0018
16			.0012	.0002
32		.0012	.00002	.00001
64	.0012	.000035	.00002	
128	.00007	.000035		
256	.00007			

Note. FORTRAN compilation, LRU replacement.

TABLE 3.13 Page fault ratios.

page size will give a smaller page fault ratio because fewer page faults are required to page in the working set. Loading and storing the working set presents a minimum page fault ratio regardless of the page size, thus larger pages will reduce the page fault ratio.

Another interesting observation is that for a fixed memory size, the page size should also be as large as possible. For example, with a partition of 256 pages and a page size of 64 words, the fault ratio is 0.00007, while with a partition of 32 pages and a page size of 512 words, the fault ratio is 0.00001.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000239

4

Memory Addressing and I/O Coherency

4.0 Overview

The issues addressed in this chapter are frequently viewed as relate to multiprocessors. However, contemporary uniprocessors with caches, virtual memory, and concurrent input/output data transfers have the same characteristics and problems that must be solved. This chapter addresses these issues in the uniprocessor context.

In the 1970s, before virtual memory systems gained acceptance, processor designers debated the advantages and disadvantages of the type of address generated by the processor; should this address be real or virtual? Morris and Ibbett present the arguments in [MOOR79]. With the almost universal acceptance of virtual memory, however, the issues now become the nature of the address presented to the cache and the address domain of the I/O system. This chapter considers two issues: (1) the addresses presented to the cache and I/O in a virtual memory system, and (2) coherency (also known as consistency) maintenance between the various spaces in a computer and its memory. These issues are present in even rather simple systems and become quite complex with larger multiprocessors.

Virtual memory computers must cope with the coexistence of virtual and real addresses in the system. Problems arise from two sources: multiple virtual addresses and replicated memory spaces. Examples are

Multiple virtual addresses
- Concurrent processors, such as I/O and the main processor
- Multiple processes;

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000240

4.3.4 *Summary*

Consider the four methods of cache access, discussed previously, combined with the two variations of DMA I/O system addressing, real and virtual, giving eight possible system configurations. Table 4.8 shows these configurations and identifies known computers for each.

Six of the eight organizations are represented by actual implementations. I can only speculate on the reason for the omission of the real cache—virtual DMA I/O and restricted virtual cache—real DMA I/O.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000241

5

Interleaved Memory and Disk Systems

5.0 Overview

This chapter discusses two important subjects, interleaved memory and disk systems, that are usually given only minor attention. Interleaved memory, which is used to provide high bandwidth, was of great importance prior to the advent of practical high-speed memories for caches. Currently, innovative architectures of modern DRAMS and various forms of interleaving are receiving design and research attention. Disk technology has had a remarkable life since the first commercial disks in 1956, the IBM 350 RAMAC. Disks have survived all predictions that the technology could go no further; indeed, cost and performance of disks have continued to improve.

5.1 Interleaved Memory

Interleaved memories provide high-memory bandwidth by distributing the read or write accesses over a number of memory modules, which is a form of parallelism. The array disk to be described in Section 5.2.3 is another instance of memory interleaving. Early research into interleaving was concurrent with and a "back up" to hierarchical memory research. In the late 1950s and 1960s no one was sure if cache technology development would be successful.

The first known reference to interleaving is found in a description of the IBM Stretch [BLOC59]: "The memories themselves are interleaved so that the first two memories have their address distributed modulo 2 and the other four are interleaved modulo 4."

A later reference to the IBM S/360 [FAGG64] points out that: "When

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000242

accesses are made to sequential storage addresses, the storage units operate in an interleaved fashion."

Pirtle [PIRT67] observes: "... the addresses are distributed over an n module memory so that address 0 is in $m(0)$, address 1 is in $m(1)$ and, in general, address X is in $m(i)$ where $i \equiv X$ modulo n."

We can see from these statements that there are two uses of the word *interleaved*. One use concerns the *units* or *modules*. Interleaved modules subdivide the module cycle time into clock periods that establish the streaming rate of the memory system. The degree of module interleaving is m and is formally defined as the number of requests that can be issued by the processor in one memory module cycle [BURN70]. Note that the clock period of the memory and the bus system are related as $m = t_m/t_c$ where t_m denotes the memory cycle time and t_c the bus or processor clock time. The other use of the term interleaving concerns *addresses*. Interleaving addresses across modules establishes the sequence of module access for a stream of requests. The degree of address interleaving is n, which is defined as the number of memory modules over which the addresses are interleaved [BURN70].

In this chapter, memory systems are described for $1 < m \leq n$. With many systems, the two degrees of interleaving are equal; that is, $m = n$. However, with some systems, such as those found with supercomputers, the degree of address interleaving is greater than the degree of module interleaving; that is, $m < n$, which is called *superinterleaving* and is discussed in Section 5.1.3.

Interleaving presents two questions of interest to researchers. First, can interleaved memories provide enhanced bandwidth to supply instruction and data streams to a processor? And second, what are the design parameters of an interleaved memory that supports vector reads and writes of supercomputers? With the success of caches, the first research objective has almost disappeared as an interesting topic. The second research topic is still an area of active research and is discussed in more detail in Section 5.1.3. Note that the IBM RT uses a noncache interleaved memory system [ROWL86] to provided instruction and data bandwidth.

The requirement for accessing vectors comes from two sources. The first source is found in the main memory references of supercomputers. Scientific programs frequently access arrays of data stored in memory as vectors. The second source is found in the interface between a cache and the main memory. A block of AUs are transferred between these two memories; this block is a vector of length 4 to 16 and is discussed in Chapter 2.

Interleaved memories are ideal for providing high bandwidth access to a long vector of data (either read or write). An interleaved memory requires two addresses: the address of the module and the address of

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000243

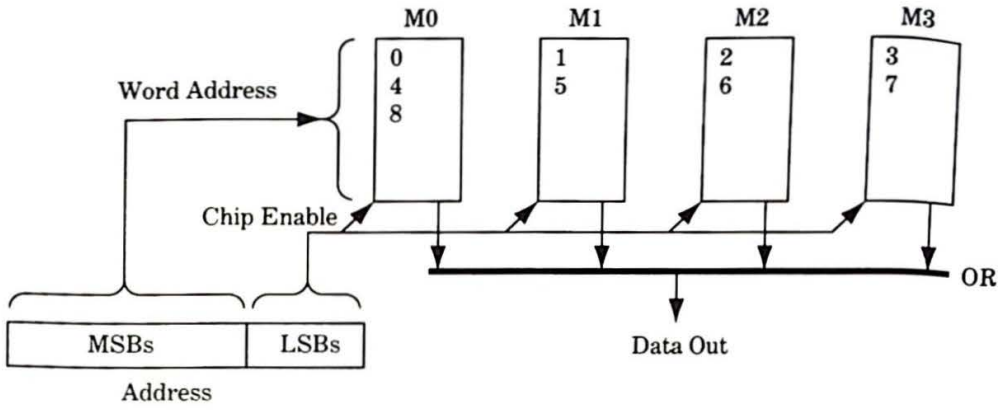


FIGURE 5.1 Four-way interleaved memory.

the word within the module. Low-order interleaving generates these addresses by dividing the memory address by the address interleaving degree n: the quotient is the word address presented to each module and the reminder selects the module. This division by an integer power of two is a trivial operation for designs having a degree of interleaving that is an integer power of two.

Consider a memory of total size 2^s, with $m = n = 4$, as shown in Figure 5.1. The four modules are addressed by the two LSBs of the address, and the word within the addressed module is selected by the high-order bits. This process is called *low-order interleaving*.

Addressable units are stored in the modules according to the sequence

Module	Addresses
0	$0, 4, \ldots, 2^s - m + 0$
1	$1, 5, \ldots, 2^s - m + 1$
2	$2, 6, \ldots, 2^s - m + 2$
3	$3, 7, \ldots, 2^s - m + 3$

A major disadvantage of low-order interleaving is the difficulty of expanding the installed memory. If the installed memory is to be expanded by a factor of 2, for example, how is this to be done? Should the degree of interleaving be increased by 2, or should the module size be increased by a factor of 2? Increasing the interleaving factor requires a major hardware modification. Increasing the size of each of the modules is a rather straightforward task. However, both expansion methods become completely intractable if the expansion is something other than an integer power of 2.

Another form of interleaving is called *high-order interleaving*, in which the quotient selects the module and the remainder selects the

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000244

word within the module. High-order interleaving does not enhance the bandwidth of the system over the bandwidth of a single memory module but instead provides for easy memory expansion. High-order interleaving also permits the easy intermixing of different memory technology such as a combination of RAM and ROM in a small dedicated controller.

5.1.1 *Performance Models*

Performance models for various types of accessing patterns were developed during early research on interleaved memories. The following paragraphs will briefly describe the models of Flores [FLOR64], Hellerman [HELL67], and Burnett and Coffman [BURN70]. These models were created to assist designers evaluate the performance impact of various interleaved memories with different workloads.

There are three useful metrics for measuring and evaluating the performance of an interleaved memory. These are

1. Speedup, S = ratio of the bandwidth of an interleaved memory to the bandwidth of a single module.
2. Acceptance ratio, AR = steady-state ratio of accepted memory requests to total memory requests [CHEU86], $0 \leq AR \leq 1$. AR is an indicator of how effectively the memory design sustainable bandwidth is being used.
3. Mean acceptance ratio, MAR = acceptance ratio that prorates the latency of accessing the first module over all of the AUs accessed [CHEU86]; $0 \leq MAR \leq 1$.

Instruction and Data Streams

The models for estimating the performance of an interleaved memory are of three forms: (1) instruction references, (2) data references, and (3) combined references. The addressing patterns of these three streams are quite different and require different performance models. Note that the problems or conflicts associated with interleaved memory systems are special cases of *structural dependencies* (discussed in Chapter 8).

Flores [FLOR64] modeled the situation where a processor and I/O channels are contending for access to an interleaved memory. He modeled the time that a request must wait, waiting time, as a function of the degree of interleaving, the ratio of I/O time to processor time, and the fraction of time that the memory is busy. For an equal ratio of I/O time and as the memory busy time approaches one, the model gives the expected result that the speedup (ratio of waiting time of an interleaved memory to a single module memory) is equal to m, the number of interleaved modules.

The next published model is by Hellerman [HELL67]. This model

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000245

can be used to predict the performance of an interleaved memory when operands are being fetched. Hellerman assumed an equal probability of access to each module for the first fetch, followed by fetches to sequentially addressed modules until the Kth reference, as

$$S = \sum_{K=1}^{m} \frac{(m-1)!\,K^2}{(m-k)!\,m^K}$$

where K denotes the length of a sequence of reads ending in a nonsequential access. With an approximate solution

$$S = m^{0.56} \approx \sqrt{m} \quad \text{for } 1 \le m \le 45.$$

The implication of Hellerman's model is that if four memory modules are interleaved, the effective bandwidth is approximately twice the bandwidth of one module for the data streams of a typical general purpose computer.

Coffman and Burnett [COFF68] developed performance models for instruction, data, and combined streams. Their performance model for interleaved memories supplying an instruction stream is

$$\text{instruction speedup} = \frac{1-(1-\lambda)^m}{\lambda}$$

where λ, the probability that an instruction is a taken branch, is equal to $1/K$. For example, consider a program stream with $\lambda = 0.2$ and $m = 8$. The speedup over a single memory module is 4.16, while Hellerman's model suggests a speedup of 2.82 for a random data reference stream, which is not a surprising result. The Hellerman model should have poorer spatial locality than an instruction stream, hence a smaller speedup.

A data stream model is developed by Burnett and Coffman [BURN70] and described in [BAER80]. This model, called the α–β model, estimates speedup for a data request stream. The assumption is made that the first module is selected at random and that for each subsequent request there is a stationary probability $1-\lambda$ that the next module in sequence is accessed on the next request. Any one of the other modules is accessed with equal probability $\beta = (1-\lambda)/(m-1)$. This model, without a closed form solution, is solved numerically. This solution convincingly shows the advantage memories where $n > m$.

Models such as the α–β model are no longer useful to designers because most computers buffer instructions, and the data references are

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000246

for vectors with a stride of one. Models that have stride as a parameter are discussed in the following sections.

Vector Data Streams

Interleaved memory has been recognized as an ideal organization for providing high bandwidth for long vectors of reads or writes. Vector access of memory has a characterization parameter, *stride*, that is not present with the models above.

Stride is the difference between addresses of successive references. The addresses, starting at 0, for various strides are

Stride = 1 0,1,2,3, . . . ,
Stride = 2 0,2,4,6, . . . ,
Stride = 3 0,3,6,9, ...

A basic performance model assumes that a vector of consecutive AUs (stride = 1) is fetched from an interleaved memory. For a stride of one (illustrated in Fig. 2.12), the time per AU is equal to

$$t_{AU} = \frac{t_m}{t_c}\left(1 + \frac{m-1}{K}\right).$$

The speedup of the interleaved memory is

$$\text{speedup} = S = \frac{m}{1 + (m-1)/K}.$$

The time per AU model assumes that the first reference is to a module that is nonbusy. Subsequent references are to non-busy modules as guaranteed by the degree of interleaving and stride. Table 5.1 shows the performance limits of vector data streams on an interleaved memory over a single module memory.

From the speedup limit we can see that the speedup of an in-

	Vector Length K	
	1	∞
t_{AU}	t_m	t_m/m
Speedup	1	m

TABLE 5.1 Vector performance limits.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000247

terleaved memory accessing a vector that has a stride of 1 is in the range $1 \leq S \leq m$.

The acceptance ratio and mean acceptance ratio indicate how well the memory system supports the demands placed upon it. For $AR = 1$, all requests are being serviced without delay. For $MAR = 1$, the vector length must be very long, $K \rightarrow \infty$, or the effective startup overhead must be reduced by concurrency.

5.1.2 *Reducing the Effect of Strides ≠ 1*

The models presented above show, for strides of 1, the influence that the degree of interleaving has on performance. As the models will show, strides other than 1 reduce the performance of an interleaved memory. For this reason, there is a body of research directed to decoupling the effect of stride and leaving only the degree of interleaving in the performance models. Most of the reported research was performed in the 1960s; however, there is renewed interest in this line of research today.

When low-order interleaved memories were applied to pipelined and array computers, performance problems were discovered. For example, if a matrix is stored by row in an interleaved memory and is read by column, the read *AR* may not be the same as the write *AR*. Instead of the addresses smoothly progressing from module to module as with the writes, there can be conflicts in the modules when the array is read. Figure 5.2 illustrates an example of this problem. In this example, a 4×4 matrix is stored by column in a 4-way interleaved memory. That is, successive AUs of each column are placed in different memory modules with a stride of 1. If the matrix is then read by row, each AU of ε row is read from the same module with a stride of 4 resulting in $S = 1$ and $AR = 0.25$. The first row (1,1 1,2 1,3 and 1,4) is contained within memory module 0 and can be read no faster than the cycle time of one module.

The reduction in potential speedup when accessing this matrix can be generally attributed to conflicts resulting from the interactions of stride values and degree of interleaving. Reducing conflicts with interleaved memories has been a research topic for over two decades

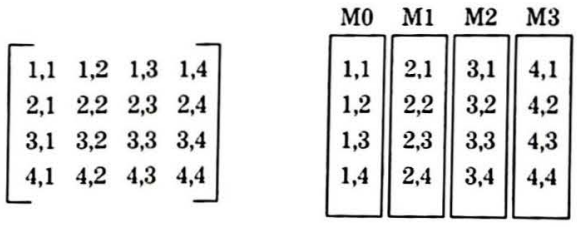


FIGURE 5.2 Matrix stored in interleaved memory.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000248

Stride	S	AR
1	4	1
2	2	0.5
3	4	1
4	1	0.25
5	4	1

TABLE 5.2 Speedup and AR for various strides, $m = 4$.

[BUDN71, HARP91, RAU91]. Access conflicts can result in a significant performance decrease with vector processors when, for example, the output of one vector operation is stored by row and the next operation needs to read the matrix by column.

Conflictfree access is possible only if the stride is not a factor or a multiple of the degree of interleaving, except for a stride of 1. In the example above, the stride is 4 when reading by column and the degree of interleaving is also 4. A stride of 3 would be accessed with no degradation. Table 5.2 shows the acceptance ratio of a 4-way interleaved memory for various strides.

For strides of 1, 3, and 5, $AR = 1$ while for a stride of 4, $AR = 0.25$ because all of the references are to the same module. Several approaches have been proposed and used to solve the stride-induced conflict problem for interleaved memories having $m = n$. These approaches are discussed below.

1. Skewed addressing;
2. Dynamic skewed addressing;
3. Pseudorandom skewed addressing;
4. Prime number interleaving;
5. *Superinterleaving*, $m < n$ (discussed in Section 5.1.3).

Skewed Addressing

Skewed addressing maps an address into the modules in such a way as to reduce the conflicts [BUDN71]. An example of skewed storage is shown in Figure 5.3, which shows the 4×4 matrix mapped into skewed storage. If the matrix is stored by row, the AUs of row 1 are stored in M0, M1, M2, M3. However, the second row is skewed one memory module, beginning with M1 and wrapping around to M0. The second row is stored with an additional skew of 1 and the third with an additional skew of 1 and so on. Note that the word addresses from the high-order bits are unchanged from the nonskewed case. However, the low-order bits of the address must be mapped into the module selection bits.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000249

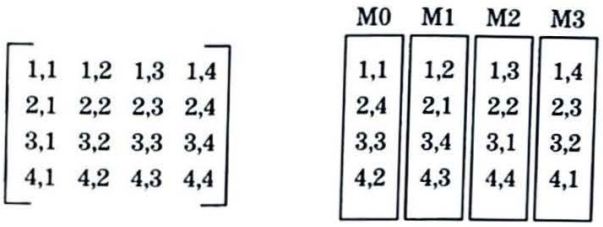


FIGURE 5.3 Skewed address interleaving.

The full speedup, $S = m$ and $AR = 1$, can now be obtained with either row or column accessing. Observe that there are strides that will interfere with the degree of interleaving and reduce S. For example, if one of the diagonals of the matrix is read, the AUs 2,2 and 3,3 are both found in M0 while 2,2 and 4,4 are both in M2, which results in $S = 2$ and $AR = 0.5$. The other diagonal finds all AUs in M3, resulting in $S = 1$ and $AR = 0.25$. Skewed storage requires that a skew factor be built into the hardware and used by all accesses. This poses a design problem for selecting the skew factor that will give the best performance over the anticipated workload.

Table 5.3 shows the addresses that needed to read the third column of the matrix with both normal and skewed (skew = 1) interleaving. The word address is the high-order bits, while the module address is the low-order bits. To read a column, the stride is four; for normal interleaving, the module address is unchanged as the word address is incremented. With skewed addressing, the word address increases as with normal addressing while the module address is formed by adding the skew factor to the current module address modulo the degree of interleaving. That is, add the skew factor to the module address but do not propagate a carry into the word address. The normal interleaved case works the

Interleaving Method			
Normal		Skewed	
Word	Module	Word	Module
00	10	00	10
01	10	01	11
10	10	10	00
11	10	11	01

TABLE 5.3 Skewed address interleaving.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000250

same way: the skew of zero is added to the module address, keeping the module address unchanged.

An interesting application of skewed storage is found in [MATI 89]. This study describes the design of a cache that uses multiple chips, requiring skewed storage. With this design, reading words from the cache into the processor and loading a block from memory into the cache will not have stride conflicts.

Dynamic Skewed Addressing

Unlike the skew scheme discussed above, dynamic skewed addressing selects a skew for the anticipated access pattern of the data that is currently active [HARP91]. This technique assumes that the compiler can determine the skew that is added to the remainder of the address. The addition is done dynamically and can be implemented for a relatively low hardware cost.

In order for dynamic skewed storage to work properly, there must be some way to bind the skew value that is used for a write to a subsequent read of the same data. In other words, a read must have access to the corresponding write skew.

Pseudorandom Skewed Addressing

Rau [RAU91] points out that any scheme for reducing conflicts that depends upon a fixed skew mapping of any type is suspect because there are some address patterns that will conflict. As an alternative, he describes and evaluates pseudorandom skewing of addresses.

Two schemes for providing pseudorandom skewing are discussed. First, the module address is XORd with a key generated with a pseudorandom generator; and second, the module address is XORd with a polynomial whose coefficients are in the Galois Field. As with dynamic skewed addressing, the skew factor must be stored by a write and be available for future read operations.

Prime Number Interleaving

Recall that conflicts will occur if, and only if, the stride is a factor or a multiple of the degree of interleaving. Thus, by selecting m to be prime, the number of strides that can produce conflicts is significantly reduced. For example, if $m = 5$, then $AR = 1$ for all strides except 5, 10, 15, ... For many problems the stride is an even power of two, thus these strides can be process conflictfree.

The first known proposal for prime number interleaving in the literature is found in [BUDN71]. I believe that prime number interleaving was used on some early drum computers to assist in minimum latency

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000251

scheduling. The Burroughs Scientific Processor used prime number interleaving with $m = 17$ [LAWR82].

Yang [YANG93] describes a cache for vector processors with a prime number of sets rather than an even power of two. The cache is late select with the sets interleaved to achieve a high streaming rate. With prime sets, the addressing patterns of various strides is broken up, permitting the cache to stream without interference. Recall from Chapter 2 that a variation of the SPARC has a prime number of sectors [WEIS92].

The problem with prime number interleaving is that the address, as with all interleaved memory systems, must be divided by the prime number degree of interleaving. The quotient is the high-order word address into the modules while the remainder is the module select address. Division by an even power of two is much simpler than division by a prime number. However, it is possible with available VLSI technology to perform the division, so prime number interleaving may become a viable alternative. The Yang design is based on interleaving with a Mersenne prime number of the set $(3, 7, 15, \ldots, (2^c - 1))$, thus facilitating the generation of the cache address.

5.1.3 *Superinterleaving*

Another technique for reducing the effect of stride on speedup and of an interleaved memory is to provide more memory modules than the address degree of interleaving, that is, $m < n$; this method is called *superinterleaving*. Recall that for any m, the clock rate of the system is the memory module cycle time divided by m. By implementing a memory with $m < n$, the number of strides that will produce a conflict in a module is reduced.

Computers such as the CDC6600 have addresses interleaved over more memory modules than the module degree of interleaving; $m < n$. This technique is also used in the Cray-1 and other computers of the Cray family. For the CDC 6600, the memory module cycle time is 1,600 ns, the system clock is 100 ns, and there are 32 memory modules giving $m = 16$ and $n = 32$ [THOR70]. The Cray-1 has 16 50 ns modules with $m = 4$, $n = 16$, and a clock period of 12.5 ns.

Figure 5.4 illustrates, by example, the benefit of superinterleaving. For this example, the addresses are interleaved eight ways ($n = 8$) while the modules are interleaved four ways ($m = 4$). Recall that m is set by the clock period of the processor and the cycle time of the memory modules. Eight cases are shown as the stride is varied from 1 to 8. A form of reservation table is used to illustrate the operation of this memory; reservation tables will be described in greater detail in Chapter 6. A

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000252

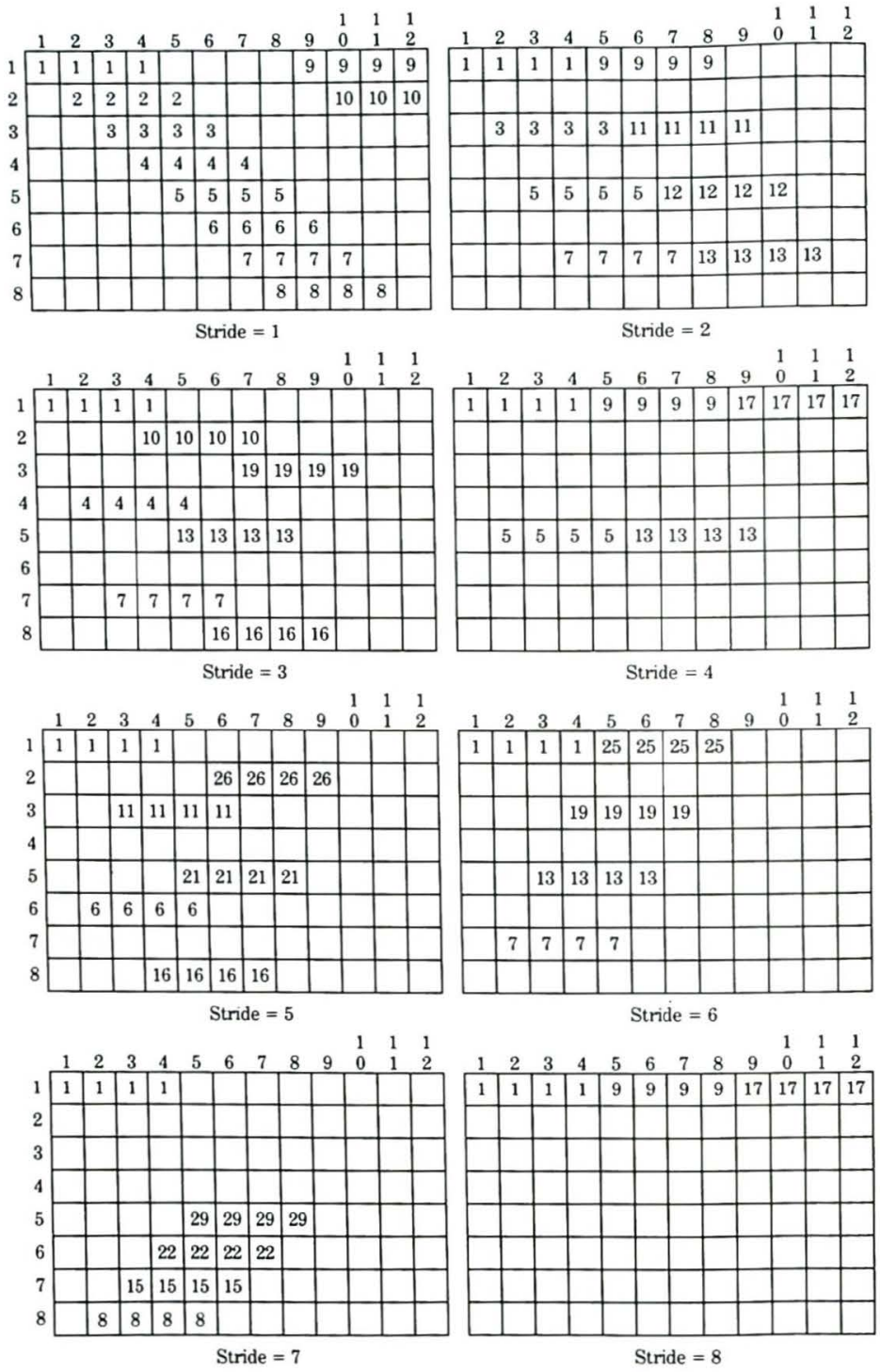


FIGURE 5.4 Superinterleaving example, $m = 4$, $n = 8$.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000253

Stride	S	AR	Memory Utilization
1	4	1	0.5
2	4	1	0.5
3	4	1	0.5
4	2	0.5	0.25
5	4	1	0.5
6	4	1	0.5
7	4	1	0.5
8	1	0.25	0.125

TABLE 5.4 Superinterleaving performance, $m = 4$, $n = 8$.

reservation table is a display of space (the memory modules) on the vertical axis and time (in clocks periods) shown on the horizontal axis.

For the first case with a stride of one, the addresses are applied to modules 1, 2, 3, . . . , 8, 9, 10, ... Each module is busy for four clocks, indicated by the address number in the table. Because $m = 4$, each module is busy for four clocks when it is cycled. For this case, $AR = 1$.

Considers a stride of 3. The modules addressed are 1, 4, 7, . . . , 2, 5, ... The memory can sustain one reference per clock, $AR = 1$, after the initial 3 clocks of latency. Note that $AR = 1$ is obtained because each of the modules is used half of the time. In other words, there is a 2X overdesign in bandwidth that is devoted to ensuring that $AR = 1$ is sustainable.

Drawing from the information presented in Figure 5.4, Table 5.4 tabulates S, AR, and memory utilization for the different strides. For strides except four and eight, this memory has $S = 1$ and $AR = 1$. Interference resulting from a stride of four reduces to $S = 2$ and $AR = 0.5$, while a stride of eight will have $S = 4$ and $AR = 0.25$—a performance that is no better than a single module because all of the references are to one memory module.

The reservation tables clearly show how excess latent bandwidth is used to improve the performance for different strides. Except for the cases where the stride is a factor or multiple of n; the utilization of the memory modules is

$$\text{utilization} = \frac{\text{used bandwidth}}{\text{available bandwidth}} = \frac{AR \times m}{n}.$$

With this example of superinterleaving, if the strides are uniformly distributed, the weighted average AR is 0.84 as compared to the $m = n = 4$ case shown in Table 5.1 with a weighted average AR of 0.68.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000254

Processor	Memory Words per Clock	Bus Words per Clock	Memory Utilization; $AR = 1$
Cray-1	16/4 = 4	1	0.25
Cray X-MP	32/4 = 8	3	0.375

TABLE 5.5 Cray memory system designs.

The Cray-1 memory is superinterleaved and has one port between the memory and its internal registers. The memory is organized $m = 4$, $n = 16$. This means that approximately 75% (12/16) of the latent bandwidth of the memory is wasted to reduce stride conflicts. The Cray X-MP extends the address interleaving to $n = 32$, which provides eightfold bandwidth over the capacity of one port [CHEN84]. With this overcapacity, three ports (two input and one output) to/from the internal registers can sustain continuous transfers for many strides. However, the Cray X-MP memory must support multiprocessors plus an I/O port per processor that reduces the bandwidth available for any one processor. The memory of these two processors is described in Table 5.5.

The memory words per clock indicates the maximum bandwidth of the memory. Bus words per clock represents the number of buses, and the maximum memory utilization is for $AR = 1$ and includes the effect of the number of memory ports.

Memories with one port to the processor and a relatively small interleaving factor, as is the case of the Cray-1, are relatively simple to implement. When the address interleaving is increased along with an increase in the number of ports to memory, the complexity of the interleaving schemes becomes costly to implement. The memory system of the TI ASC, for example, had eight memory modules ($m = n = 8$) and eight user ports, requiring two racks of equipment for the crossbar switch [CRAG89] (which is a very expensive system component).

Two of the major changes from the Cray-1 to the Cray X-MP are an increase in the number of memory modules from 16 to 32, and an increase in the number of ports from one to four (two read, one write, one I/O). The memory module cycle time is 36 ns, and with $n = 32$ the potential streaming rate out of the memory system is one word every 1.125 ns. Given the complexity of a full 32×4 crossbar switch, some hierarchical interconnection or multiplexing system is called for, as shown in Figure 5.5.

The 32 memory modules are grouped into four sections of eight modules each. The modules of a section pass data via eight *lines* (8 to 1 multiplexer-demultiplexer) that are then selected by a 4×4 crossbar switch. With this interconnection scheme, the potential bandwidth of the

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000255

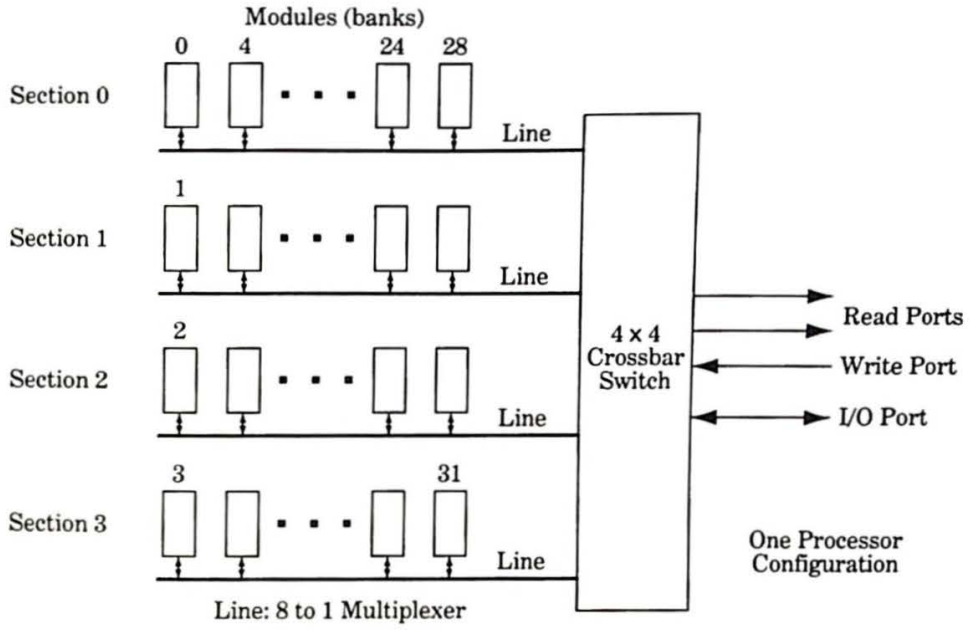


FIGURE 5.5 Cray X-MP-2, one processor, memory organization.

memory is reduced from 32 possible memory references per clock to 4 memory references per clock because of the constrictions imposed by the four lines. With four ports active, there is no excess bandwidth to mitigate the effects of conflicts. Note that for each processor in the system the four lines and the crossbar switch are replicated.

The crossbar switch of Figure 5.5 is assumed to be conflictfree for all accesses from all ports. However, conflicts exist with the Cray X-MP-2 memory in addition to the stride conflicts of the Cray-1. The conflicts are:

1. Module (bank) conflict due to access from one port—a *stride conflict*;
2. Module (bank) conflict due to access of an idle module by two or more ports—a *simultaneous bank conflict*;
3. Line conflict due to access by two or more ports—a *section conflict*.

The effects of these three types of conflicts have been extensively studied [CHEU84, CHEU86, OED85, WEIS92, CALA88, CALA88a]. Cheung and Smith [CHEU84] used reservation table analysis of memory conflicts to evaluate *MAR* values; excerpted results are shown in Table 5.6. Their reservation tables include memory banks and lines as resources. In all of the cases evaluated, stride is set to 1. Their results indicate that *MAR*s of 0.8 or better are achievable with the Cray X-MP-2 memory.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000256

Conditions	*MAR*
Two-vector stream	0.975
Three-vector stream	0.80

TABLE 5.6 Cray X-MP-2 memory conflicts.

The line conflict problem of the Cray X-MP-2 suggested an improvement for the Cray X-MP-4. This improvement *renumbers* the memory modules, which is a technique similar to skewed storage described in Section 5.1.2. The address interleaving (module numbering) scheme for these two processors is shown in Table 5.7 for the first 32 references, starting with 0. For example, for the X-MP-2, address 13 is found in Bank 3, Module S1. With renumbering, the X-MP-4 address 13 is found in Bank 1, Module S3.

Cheu [CHEU86] shows that the renumbering scheme of the X-MP-4 improves the *MAR* over that of the X-MP-2. Specifically, this reassignment changes *MAR* from 0.929 to 1.0, an improvement of approximately 7%.

Note that the number of interleaved addresses per line, called

	Cray X-MP-2							
	Bank							
	0	1	2	3	4	5	6	7
Section 0	0	4	8	12	16	20	24	28
Section 1	1	5	9	**13**	17	21	25	29
Section 2	2	6	10	14	18	22	26	30
Section 3	3	7	11	15	19	23	27	31

	Cray X-MP-4							
	Bank							
	0	1	2	3	4	5	6	7
Section 0	0	1	2	3	16	17	18	19
Section 1	4	5	6	7	20	21	22	23
Section 2	8	9	10	11	24	25	26	27
Section 3	12	13	14	15	28	29	30	31

TABLE 5.7 Cray X-MP-2 and X-MP-4 address interleaving.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000257

	Number of Sections (*NS*)			
NBPS	2	4	8	16
1	0.5	0.25	0.125	
2	1.5	0.75	0.38	
4	3.5	1.75	0.88	
8	7.5	3.75	1.88	0.94
16	15.5	7.75	3.87	1.93

TABLE 5.8 Cray X-MP memory latency (clocks).

number of banks per section (*NBPS*), can be 1 (X-MP-2), 2, 4 (as shown above), or 8. Another parameter is the *number of sections* (*NS*). The selection of *NBPS* for a given *NS* can have a pronounced effect on the performance of the memory system. However, improving *MAR* by module renumbering works at the expense of increasing the latency (sometimes called *startup delays*) while transient conflicts are resolved. Simulation results on the latency for various *NBPS* and *NS* from [CALA88] are given in Table 5.8 for a two-port system.

The change in interleaving for the Cray X-MP-4 results in an additional 1.25 clocks of latency. Similar results for a three-port access show that the Cray X-MP-2 has a delay of 0.67 clocks and that the Cray X-MP-4 has a delay of 5.11 clocks. Latency is reduced as the number of sections increases. This is not a surprising result because if the number of sections is equal to the number of modules, then a full crossbar switch is specified. For one section, all memory modules are connected by one line or a multiplexer offering the maximum potential for conflict from references from another port.

The memory designer must face the issue of balancing *MAR* and latency. As discussed in Chapter 11, for short vectors latency should be minimized while for long vectors *MAR* should be minimized. Additional recent research on the issue of high-performance interleaved memory for vector processors is found in [CHEN91] and [HARP91].

An interesting comparison of the memory configurations for various members of the Cray X-MP family is shown in Table 5.9.

The data in this table is extracted from [THOM86] and shows the number of memory modules in **bold** and the number of modules per port in *italics*. For example, a four-processor configuration with 8 Mwords of memory has 32 memory modules. The number of modules/port is based on three ports per processor; the I/O port is ignored. For example, the four-processor system with 64 memory modules has 12 ports and 5.33 modules per port.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000258

	Memory Modules No. of Processors			Modules/Port No. of Processors		
Memory Size	1	2	4	1	2	4
16 Mwords		**32**	**64**		*5.33*	*5.33*
8 Mwords	**32**	**32**	**32**	*10.6*	*5.33*	*2.66*
4 Mwords	**16**	**16**		*4.33*	*2.66*	
2 Mwords	**16**	**16**		*4.33*	*2.66*	
1 Mwords	**16**			*4.33*		

TABLE 5.9 Cray X-MP memory modules and modules/port.

With $n = 4$, the observation can be made that some models of this processor do not have enough memory modules to support the number of ports. Any module/port value less than 4 is marginal in memory bandwidth and may suffer performance degradation for some benchmarks. The X-MP-4 with 8 Mwords of memory can be a marginal performer.

5.1.4 *Interleaving for Multiport Access*

Memory module interleaving is also used to permit concurrent access to a memory when two or more addresses are not to the same module. This capability is a feature of the Cray memories described in Section 5.1.3. However, a simpler system is used in the Intel Pentium for its data cache [ALPE93]. This processor is a superscalar implementation (discussed in Chapter 10) that has the need to perform two cache transactions in one clock period. This memory system is discussed in this chapter because the technique is applicable to multi-access memories in general, not just to caches for superscalar processors.

The basic idea is that with addresses interleaved across m modules ($m = n$), accesses to adjacent address will not conflict and can be processed in parallel. Only when there is a module conflict will one of the requests be deferred. A simplified block diagram of the Pentium data cache is shown in Figure 5.6. This cache has 8 modules, $m = n = 8$, with each module organized 256×4 bytes. The total cache capacity is 8 Kbytes.

Two addresses can be presented at the same time to the cache. If the addresses are not the same, the two addressed memory banks are cycled and the two AUs are gated out through the multiplexers. If there is a conflict, the U access has priority and the V access stalls for one cycle. The designers of the Pentium considered a dual-port cache as well

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000259

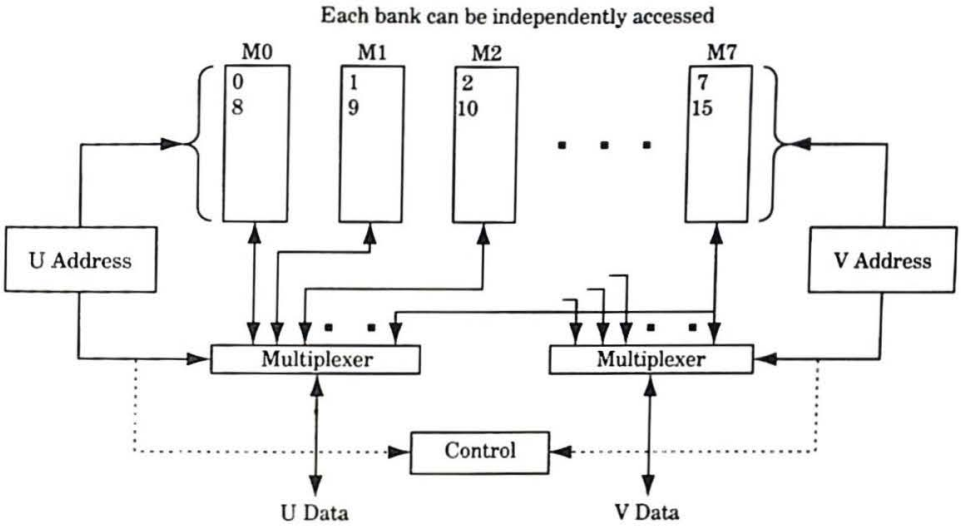


FIGURE 5.6 Pentium interleaved data cache.

as this interleaved cache and selected this design based on its smaller area and reduced complexity for handling data dependencies. This cache is discussed further in Section 10.2.2.

5.1.5 *Cache-Memory Transfers*

The use of interleaved memory as a technique for reducing the transport time of a block to/from a cache is discussed in Section 2.2.1 and Figure 2.13. Because a block is a vector with a stride of one, an efficient transfer is achieved. For example, the IBM S/360/85 has a 4-way interleaved core memory, each module having a word length of 16 bytes and cycle time of 1.04 μs. The processor and cache have a cycle of 80 ns. The interleaving is specifically designed to transfer a 64-byte block. After the first module cycle is initiated, the second module read starts 80 ns later, then the third, then the fourth. The first 16 bytes of the block are transferred at the end of the access portion of the cycle, and each of the other 16 bytes comes 80 ns later. For this design, $a = 1.04$ ns and $b = 80$ ns. Note that the parameters a and b are defined in Chapter 2 in terms of clocks, not time.

The performance effect of block size and interleaving is modeled in [SMIT87]. The length of the block can have a critical impact on the cache performance if the cache is blocked during a block load. However, if one of the methods of load forward or fetch bypass is used, this performance penalty can be reduced.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000260

5.1.6 *Nibble, Page, and Static Column Mode Drams*

This section briefly describes memory chips that achieve an efficient transfer similar to that possible with interleaving. Nibble-mode DRAMS supply four bits in one cycle. A page-mode DRAM accesses the array and holds the accessed row in a buffer that is then accessed by the row (low-order bits) address. The size of the buffer is the square root of the number of bits in the chip. For example, a 1 M-bit device will have a 1024-bit buffer, which provides a possible block size of 128 bytes. The static column mode is similar to the page-mode DRAM and provides a faster transfer of bits after accessing the row.

The Texas Instruments TMS44C256 DRAM serves as an example of a nibble-mode device, shown in Figure 5.7. The TMS44C256 is organized 256 Kbits × 4 bits (1 M-bit DRAM). Internally it consists of 4, 256-Kbit arrays as shown in Figure 5.6. This device is supplied in three speed ranges; this example uses the highest speed one.

A read cycle is initiated with the presentation of the address (18 bits), and each of the four arrays are read. As shown in the cut-out, each array is organized 512 × 512 bits. Nine bits select a word in each array, and the output bit is selected by the other nine bits. The read cycle places 512 bits in each of four buffers for a total of 2 Kbits. These bits are gated into the output, either as a 4-bit nibble or, if in the page mode, up to 512 nibbles in sequence.

The performance model for the nibble, page, and static column devices is the same as that of interleaved memory. That is, there is a time a required to fetch the first AU and then a time b to fetch each of the successive AUs. The total transport time increases monotonically with the block size unless another fetch is required to start the cycle over again. For the TMS44C256, the read and write cycle time is 150 ns and the read or write page mode is 50 ns. The parameters for computing the transport time are a = 150 ns and b = 50 ns.

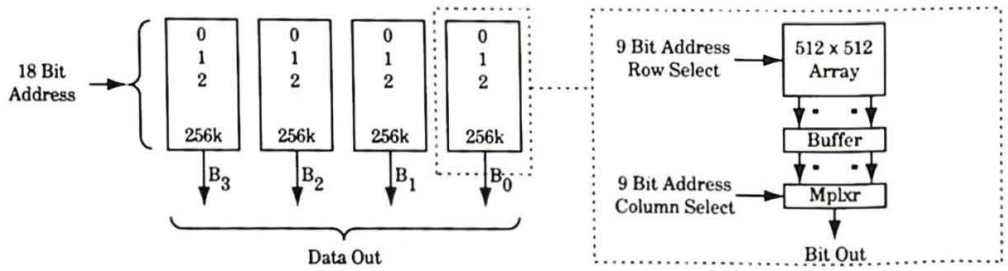


FIGURE 5.7 DRAM organization.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000261

5.1.7 *Wide-Word Interleaved Modules*

Another method for increasing the bandwidth of a memory is to increase the width, in bits, of a memory module to a multiple of the addressable unit (2X, 4X, . . .). In addition, a number of these modules can be module interleaved for even greater bandwidth.

Consider a memory module with a width of two AUs and with four of these modules interleaved as shown in Figure 5.8. The matrix of Figure 5.2 is stored in this memory as shown. The high-order address bits select the word in each of the modules; the low-order address bits select the module and the left-hand or right-hand word of the addressed module.

If the matrix is accessed by row, the maximum bandwidth is obtained because the stride is one. Under these conditions, the speedup is $S = m \times W$; where W = number of AUs per memory word = 2, $S = 4 \times 2 = 8$.

On the other hand, there can be addressing patterns that will reference only one module. Under these conditions, $S = 1$. Thus we see that $1 \leq S \leq (m \times W)$.

Experience has shown that wide-word interleaving yields a speedup that is similar to module interleaving. Expressing this relationship in the terms of Hellerman's model for random data addressing patterns, the effect of interleaving is $\sqrt{m}$ and the effect of having a wide-word is $\sqrt{W}$. Furthermore, the effect of these to enhancement steps is multiplicative;$S = \sqrt{(m \times W)}$.

It is interesting to note that von Neumann [BURK46] proposed wide word interleaving as a memory organization. The memory is discusse in Chapter 2 in terms of being a queue. In the simple example given in

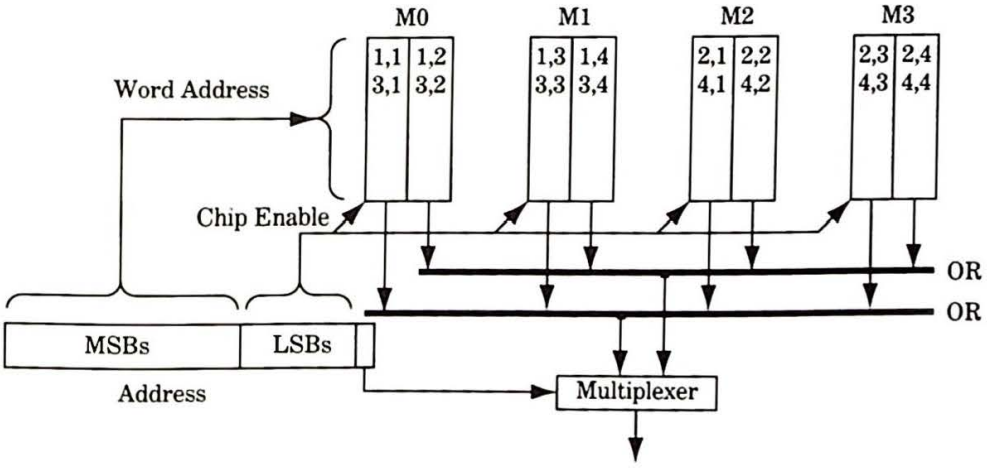


FIGURE 5.8 Wide-word interleaving.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000262

Chapter 2, three memory cycles are required to read five instructions. A memory that is only 20 bits wide would require five memory cycles. Thus, the speedup is 5/3 or 1.666, which is rather close to $\sqrt{1 \times 2} = 1.414$. The simple model and example give speedup predictions that are reasonably close to each other.

The most compelling reason for wide-word memories today is to reduce the cache miss transport time. Wide-word memories complement wide busses, reducing the number of clocks required to transfer a block to/from a cache. Recall from section 2.2.1 that the memory modules of the IBM S/360/85 are eight bytes wide. Thus only four clocks are needed to transport a 32-byte block from the memory to the cache.

5.2 Disk Systems

The input/output of a computer has always been a bottleneck in system performance. During the early years, punched cards were the primary I/O media, followed later by magnetic tape. With the advent of the first commercial disk in 1956, cards have disappeared and magnetic tape has become archival media. The IBM 350 RAMAC, introduced in 1956, had a capacity of 40 Mbits (not bytes!) on 100 surfaces with an access time of 0.5 seconds [MATI 77]. Access to the RAMAC is random when compared to magnetic tape, and the name RAMAC is taken from Random Access Memory with the then-common suffix AC. Even though magnetic drums were used for mass storage and virtual memory as early as 1948, they were severely limited in capacity and the RAMAC type disk was a welcome replacement.

A disk is organized in a hierarchy of data storage areas described in Figure 5.9. The basic storage unit is the *sector*, which contains 32 Kbytes to 4 Kbytes. A sector is the unit of transfer between the disk and main memory and is usually the page in a virtual memory system. A track consists of 4 Ksectors to 8 Ksectors, and there are 30 to 2,000 tracks on a surface. Note that the disk sector and cache sector are not the same thing. The tracks are accessed with Read/Write (R/W) heads on an arm

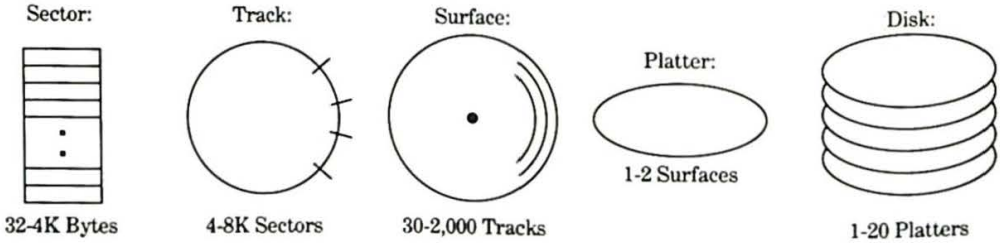


FIGURE 5.9 Disk storage hierarchy.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000263

positioned over a track by an actuator. Each platter has one or two surfaces, and 1 to 20 surfaces make a disk unit. Needless to say, these hierarchy numbers change with time, generally increasing, and are only a guide to the relative dimensions.

Another component of disk organization concerns the relationship between the arms with their R/W heads and the tracks; each surface has at least one arm with its R/W head. Disk systems with more than one arm (either more than one arm for one surface or more than one surface each with one arm) can have either independent arm movement (independent actuators) or the arms can be ganged together in groups. The tracks under a ganged group of heads are known as a *cylinder*, a term taken from drum technology. A cylinder consists of the tracks that are under the heads at one time. Note that for a system with independent arm movement, cylinders can be virtualized by positioning the arms over the tracks of a cylinder.

The discussion above highlights heads that have the combined function of read and write. However, disk technology is moving in the direction of independent read and write heads. The reason for this is that recording and reading are no longer complementary with the higher areal densities and purpose-designed heads are required as well as accompanying arms and positioning circuits.

Two examples of disk organization hierarchy are shown in Table 5.10: the Cray DD-49 disk unit, as reported in [KATZ89], and the EX-2, 5-1/4″ floppy reported by [SARG86]. The first is a high-capacity disk for a supercomputer. These parameters reflect formatted disks.

Disk surface areal density (bits/inch × tracks/inch) has increased at a rate of 26% per year since 1971, and these increases have translated almost directly into cost reduction. However, even with improvements in both latency and transfer rate of the early disks as compared to magnetic tape, disk systems remain a limiting factor in computer system performance because of their mechanical characteristics that limit lat-

	Cray EX-1	Cray EX-2 (floppy)
Bytes/Sector	42	512
Sectors/Track	4096	15
Tracks/Surface	443	80
Surfaces/Platter	2	2
Platters/Disk	8	1
Total Capacity	1.2 Gbytes	1.2 Mbytes

TABLE 5.10 Example disk parameters.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000264

ency reductions. In this chapter several issues concerning disk systems are addressed, including disk latency, bandwidth requirements, performance, cost projections, interleaved disks (RAID), and disk caches.

5.2.1 *Disk Requirements*

There is very little solid information published on the question of the disk bandwidth and storage requirements based upon workloads that can serve as a design guide [GOLD87]. The usual design approach is to assume that all of the disk parameters are inadequate to the task and the designer should provide the needed capacity and then just provide the best performance possible with the selected disks. In fact, most of the research in the use of disks has been directed toward this second step: overcoming the inadequacies of the disks. For example, [MILL91] reports on measurements of I/O activity made on a Cray Y-MP 8/832 when executing scientific benchmarks. Unfortunately, most of the data is referenced to CPU time and are therefore of limited use for design guidance.

Five parameters are commonly used to characterize a disk system.

1. Capacity. Bytes of storage (formatted).
2. Bandwidth. Bytes transferred/unit of time.
3. Service rate. Number of service requests satisfied/unit of time.
4. Response time. Time between the start and completion of an event.
5. Cost. $/Mbyte.

The reader should keep in mind that the requirements for disk systems are different for mainframes and single-user computers. In the former case, capacity and service rate are of prime importance; for the latter, response time is of prime importance. These differences are discussed in the following paragraphs.

Capacity

A study by Goldstein [GOLD87] indicates, for mainframe computers, that the average disk storage capacity is 3.7 Gbytes per MIPS; see Figure 5.10. The plot shows the capacity per MIPS from a customer survey taken in 1985. This data is for large mainframe applications and does not reflect the requirements of work stations and other single-user computer systems. However, servers should have approximately the same disk capacity per MIPS as mainframes.

Other data from Goldstein's study, reflected in Table 5.11, shows that the capacity index has remained relatively constant at 3.7 Gbytes/MIPS since 1980.

The capacity of the hard disk of a typical personal computer has

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000265

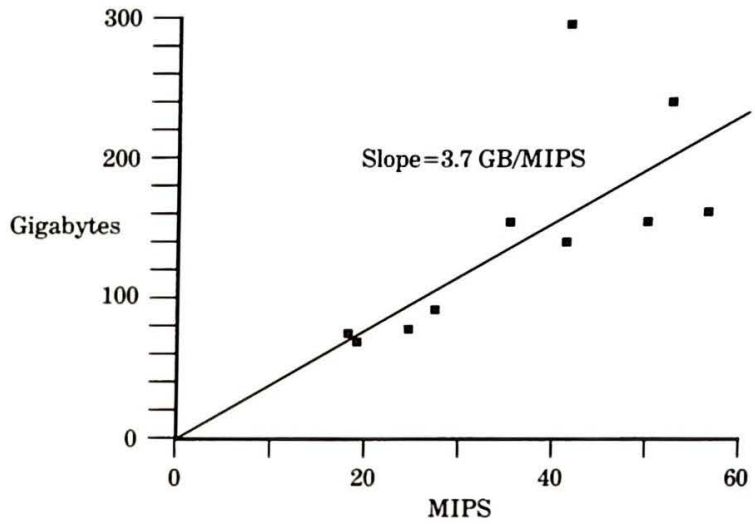


FIGURE 5.10 Mainframe disk capacity versus MIPS.

Year	GB/MIPS
1972	1.5
1974	2.0
1976	2.6
1978	3.5
1980	3.7
1982	3.6
1983	3.7

TABLE 5.11 Mainframe disk capacity versus year of survey.

grown from 5 Mbytes in 1988 to 250 Mbytes in 1994, which is an absolute capacity growth rate of 32% per year. A typical PC of 1980 and 1994 both have approximately 5 Mbytes per MIPS; installed hard disk capacity is tracking the increase in processing rate. Note, however, that the PC ratio of 5 Mbytes per MIPS is 1/740 that of the Goldstein mainframe ratio noted above. We can see from this information that there can be a major difference in ratio of disk capacity to MIPS depending upon the particular use and application of the computer.

Bandwidth

One of the famous computer design "laws" is the Amdahl Law, also called the CASE/AMDAHL law [HENN90, AKEL91], that states that a balanced system should provide a bit of I/O transfer for every instruc-

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000266

Problem Type	Bits Transferred per Instruction Executed
Scientific	1
Commercial	2
Interactive	4–8

TABLE 5.12 I/O bandwidth requirements.

tion executed. However, I cannot find a reference to this law in the published literature attributable to either Amdahl or Case. I can only conclude that this law is a rule-of-thumb in the computer art with no recognized reference to an author. A balanced system is one in which all of the queue depths are stable and are approximately equal to one and one component is not significantly limiting the performance of the system.

The definition of bandwidth in Chapter 2 is changed when referring to Amdahl's law in that the normalizing function is an executed instruction rather than time. Taken at its face value, Amdahl's law means that a processor executing at the rate of 10 MIPS will require an average I/O transfer of 10 million bits (1.25 Mbytes) per second. And, because the transfers would be in bursts, the peak transfer rate will be much higher.

Another view of the bandwidth requirement is given by Matick [MATI77] and shown in Table 5.12. The data shown is a composite view of data published by other researchers in 1970 and 1971 and is quite old.

Notice that for scientific processing, one bit is transferred for each instruction executed, which is the same value cited as Amdahl's Law. However, for interactive processing, the I/O requirement is 4 to 8 times more demanding, and transaction processing should have approximately the same requirement. As interactive and transaction processing increase in importance, the need to provide high I/O bandwidth is an important design problem.

A small amount of contemporary data exists on this issue. An I/O demand of 0.73 bits per instruction has been reported for a VAX 8800 executing a mix of batch, system, and interactive tasks [CLAR88]. For a scientific program, an atmospheric simulation model on a Cray-2, 0.32 bytes per instruction are required [CATL92].

Data presented in [KATZ89] shows that the bandwidth of a single-disk system is in the range of 0.3–3.0 Mbytes per second. Based upon one bit per instruction, these disks are capable of supporting approximately 1 MIPS of interactive processing. Thus, a single disk is inadequate for high-performance processors in most programming environments.

Current research concerns techniques for improving disk system

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000267

bandwidth by spreading the data over a number of disks, or disk array. An early survey of disk array research is found in [KATZ89]. A simple example of the disk array technique is to consider that 32 disks are used to store data. If the basic disk bandwidth per disk is 2 Mbytes per second, the aggregate bandwidth is 64 Mbytes per second. The technique of arraying disks is similar to interleaving memory modules discussed previously. Some of the major design issues with disk arrays are data layout, buffering, reliability, and error recovery (discussed further in Section 5.2.3). The first known example of a commercial disk array can be found in the Connection Machine Data Vault [KATZ89].

Latency

The deficiency in disk bandwidth or transfer rate is usually masked by the very long latency of a disk access. Latency of a disk consists of two components: (1) seek time and (2) rotational latency. Seek time is the time needed to position the head under the desired track and consists of the arm start time, traverse time, and stop time. Complex hardware mechanisms have been designed to reduce seek time and improve the positioning accuracy of the head. Nevertheless, seek time can be in the range of 2 ms to 10 ms depending upon the number of tracks to be crossed.

In addition to hardware approaches to reducing seek time, arm scheduling algorithms can be employed to reduce effective seek time. Effective seek time reduction can be achieved by taking requests off the queue such that the request for a close track is served before a request for a distant one. For heavily loaded systems that have a large number of requests in the queue, arm scheduling works well. However, arm scheduling does not always improve seek time. For example, systems that are balanced with regards to processing and I/O should have an average queue depth of less than one; consequently, arm scheduling becomes ineffective [KIM 87]. Also, arm scheduling is of little benefit to single-user systems unless there are background batch jobs executing that will fill the request queue.

Rotational latency is the time required for the desired sector to reach the R/W head and is set by the rotational speed of the disk. Rotational latency is, on average, one half of the disk's rotation time. Many disks rotate at 3,600 RPM, giving an average rotational latency of 8.3 ms.

One approach to improving the rotational latency of a disk is *rotational positioning sensing* (RPS). One use of RPS is to schedule disk transfers of the queue based on serving the first sector that will be under the heads, which is similar to arm scheduling. Another use of RPS is to improve the utilization of the disk channels. With RPS, after the access command is presented to the disk, the channel is disconnected and made

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000268

available for another access. Then the position of the disk is sensed and as the desired sector approaches the head the channel is reconnected. RPS is discussed further in Section 5.2.2.

Ng points out that there is a potential problem with RPS when used to improve channel utilization [NG91]. Because of the great difference between latency and data transfer time, large systems disconnect the channel after a request has been made to the disk for an access. Therefore, the channel is free to transfer data to/from another disk. However, there is a possibility that the channel is still connected to the other disk and cannot be restored to the disk that released it. When this happens, an RPS*miss* occurs, requiring another complete rotation before the request can be serviced. The latency of an RPS*miss* can be greater that the seek time. Ng discusses four methods by which the problem with RPS*miss* can be reduced. These include three forms of data redundancy and a dual actuator approach.

The dual actuator approach has a heritage in the paging drums of the 1960s, which had two heads per track and zero seek time because each track had a head and the average rotational latency was cut in half to approximately 4 ms. Data redundancy is also used on paging drums with only one head per track; redundant data is placed 180 degrees apart. Thus, one copy is always within a half rotation of the head. Another trick for eliminating rotational latency is to have the sector size equal a full rotation of the drum. The sector is transferred into a random access memory; consequently, the transfer could start at any point in the sector. Paging drums are no longer used because of the availability of low-cost RAM used as either main memory or as a disk cache.

The total latency (seek plus rotation) for one actuator of a mainframe class disk has decreased from approximately 30 ms in 1985 to approximately 15 ms in 1990—a reduction rate of 13% per year. Even if the rate of decrease in total latency continues to the year 2000, the latency will still be approximately 4 ms—the same as that obtained with paging drums.

In addition to arm scheduling and RPS, the use of buffers or disk caches has significantly improved the latency of disks where either (1) software prefetching is effective or (2) the pages demonstrate temporal locality. A hypothetical system, circa 1965, is shown at the top of Figure 5.11. This system transfers I/O directly into and out of main memory. Using the Matick data of 1 million bits per 1 million instructions for scientific programs and assuming that the sector size is 8 Kbits, the sector accesses per million instructions is

$$\text{accesses/million inst.} = \frac{1 \times 10^6}{8} \times 10^3 = 125.$$

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000269

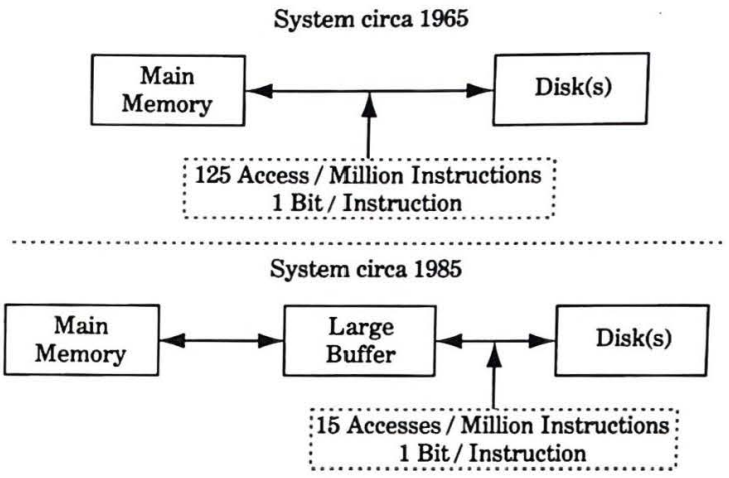


FIGURE 5.11 Disk requirements reduced with disk buffers.

The systems of the 1960s had small buffers that only smoothed the flow of data between the main memory and the disk. However, as the buffers increased in size, the "miss rate" on these buffers decreased and the traffic on the bus between the buffer and the disk decreased. The transition is now underway to convert the buffers into hardware-managed "disk caches." The benefits of buffers can be found in Goldstein's data from a 1985 survey, which shows 15 disk accesses per second are required for each MIPS or 15 accesses/million instructions; see the bottom of Figure 5.11. The 1985 system has a buffer between the main memory and the disk that operates similar to a cache and has a P_{miss} estimated to be 15/125 = 0.12.

Note that the number of bits of real-disk I/O and the number of real-disk accesses per instruction is reduced by the use of buffers only to the extent that files have temporal locality. The buffers decouple many of the disk accesses and transfers from the physical disk by servicing them in the buffer, hence reducing the number of real-disk accesses/instruction by a factor of eight over this twenty-year period. This reduction is 11% each year, which is close to the 10% reported by Goldstein for the period 1980–1985.

Because of increasing instruction processing rates, ever higher access rates and I/O bandwidth are required. Table 5.13 shows projections of bytes/instruction, bytes/second, and accesses/second for the disk system (that is, buffer accesses of the disk). This projection is based on an increase in MIPS of 30% per year and a decrease in I/O bytes per instruction of 10% per year resulting from buffering. By the year 2000, the disk must be capable of 6 Mbytes/sec and 225 accesses/sec.

The developments in disk arrays will ensure that the bandwidth requirements are achieved. However, research into short latency disks

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000270

		Disk, After Buffering		
Year	MIPS	Bytes/Inst.	Bytes/Second	Accesses/Sec.
1985	5	0.10	500,000	15
1990	20	0.06	1,200,000	36
2000	300	0.02	6,000,000	225

TABLE 5.13 Disk bandwidth requirement projection.

has not yet indicated a way to achieve the goal shown in the table. As with caches and virtual memory, buffers must still be loaded with the working set and, this time, may be a major determinant in the response time of a system, regardless of the use of a buffer. Long response time or disk latency is particularly detrimental to single-user workstations and PCs.

Service Rate

Service rate (also called *throughput* and *bandwidth*) is defined as "the amount of work completed in a specific interval of time" [DONO72]. There are two views of service rate: The theoretical maximum service rate of a system and the actual service rate of a system that is receiving work requests. The latter is always less than the former. Factors such as the number of disk channels and load on the CPU play significant roles in setting the service rate of a disk system.

The combination of latency, bandwidth, and scheduling produce a service rate measure of the disk in terms of transactions/second. The first commercial disks, circa 1965, could perform approximately 3 accesses per second. Referring again to the data of [KATZ89], for the disks listed, the maximum service rate is in the range of 0.8 to 50.0 I/Os per second per actuator, not counting queue time.

Analytic modeling techniques have been developed that permit designers to evaluate trial designs and to assess tentative system changes to improve performance. Buzen developed a canonical model of a processor system [BUZE71, DENN78] that is used to evaluate processor and disk configurations and the level of multiprogramming. The issues discussed in the next paragraphs concern how the mean service rate varies as the system parameters are changed.

The Buzen model, as shown in Figure 5.12, is called the *central server model*. The central server model has one CPU and m I/O devices with service rates m_i. As a task completes processing, it is put into one of the I/O queues or exits and is replaced by an identical task that is placed on the CPU queue. The probabilities of these paths are given by

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000271

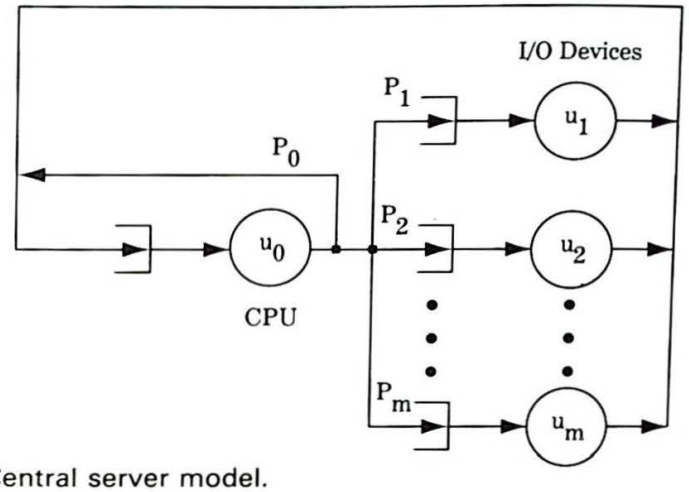


FIGURE 5.12 Central server model.

P_m. After an I/O event, the job is placed on the CPU queue. The I/O devices are assumed to be disks, drums, or a combination of the two. The response time solution to this model is found by recursively evaluating the response as the degree of multiprogramming is increased from 1 to 2 and so on. In addition to response time, the utilization of each of the I/O devices and the CPU can be determined.

Allen [ALLE80] provides an example of the use of the central server model for a system with a CPU and two I/O devices. The service rates and transition probabilities are

$$\mu_0 = 100/\text{sec}, \quad P_0 = 0.1;$$
$$\mu_1 = 25/\text{sec}, \quad P_1 = 0.2;$$
$$\mu_2 = 40/\text{sec}, \quad P_2 = 0.7.$$

The level of multiprogramming is set to 4. A proposed change is to replace I/O device #2 with a faster device; what are the original and the improved service rates? The results from the Buzen Central Server model are shown in Figure 5.13. By doubling the service rate of the disk #2 from 40 to 80 I/Os per second, the service rate of the system increases from 5.17 to 7.43 tasks per second—an increase of 43%. Other system changes can be investigated, such as a faster processor and/or increasing the performance of the slowest I/O device. Note that the asymptotic service rate of this system, with I/O devices that are infinitely fast, is 10 tasks per second as 10 passes through the CPU are required to complete a task. With these very fast disks, the system becomes CPU bound.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000272

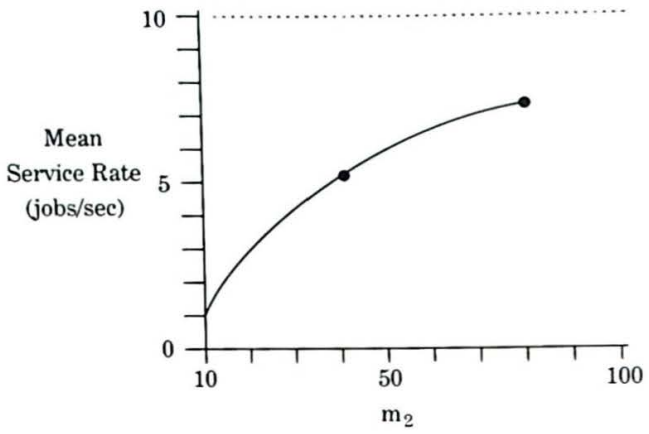


FIGURE 5.13 Service rate, mean jobs per second.

Analysis of systems performing conventional file-based I/O clearly indicates the detrimental effect of slow disk systems on total system performance. Reddy and Banerjee [REDD89] examined various alternative configurations of multiple disks and evaluated their performance for various scientific workloads. Akella and Siewiorek [AKEL91] survey various performance models. They measure the performance of VAX processors in relation to the measured performance of their models.

Response Time

Response time (also called *latency*) is the time between the start and completion of an event. The techniques of queuing theory are useful in evaluating response time for certain classes of systems. A thorough discussion of queuing theory is outside the scope of this book. However, I want to discuss a simple model; interested readers can consult [JAIN92].

For transaction processing and batch systems, response time is an important service parameter. While the system owner is interested in service rate and system utilization, users want rapid response time. A large transaction system may process thousands of transactions per second, and each transaction may require tens of disk accesses. Thus, the service rate of the disk system is an important parameter in satisfying a response time specification.

An open system queue model can provide an estimate of the response time of a transaction-processing system. The model assumes that there is an infinite pool of requesters and that when a request is serviced, the requester leaves the system. The arrival rates and service rates have Poision distribution. Transaction-processing systems, as well as grocery checkout and airline service counters, can be analyzed by this model.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000273

The mean time in the system is

$$t_q = \frac{1}{\mu - \lambda}$$

where t_q denotes the mean time in the system, that is, queue time + service time; μ is the mean service rate; and λ is the mean arrival rate.

Two observations can be made about this model. If the arrival rate approaches zero, the mean time in the system is $1/\mu$, and if the mean arrival rate approaches the mean service rate, the mean time in the system approaches infinity. Both of these results can be observed in real life situations.

We can apply this model to the system described in the discussion on service rate, that is, a system with $\mu = 5.17$ tasks per second. If tasks are arriving at $\lambda = 2$ requests per second, the mean time in the system for a task is 0.315 seconds. A task is in the queue for 0.122 seconds and is processed for 0.193 seconds. With the infinitely fast disks, μ becomes 10 and the mean time in the system increases to 0.125 seconds. With this model, a system designer can trade off system cost with customer response time.

Note that the open system model can give misleading results if not properly applied. The assumption of an infinite source of requesters and no return to the input is not valid for many systems. For example, a system with a number of terminals must be modeled by a closed system model. Denning [DENN78] provide an excellent treatment of closed system models.

Disk Cost Projections

The areal density of disk recording increased from 0.002 Mbits/in^2 in 1957 to 10 Mbits/in^2 in 1980—an increase of approximately 43% per year [HARK81] that continues today. The cost of disk storage is shown in Figure 5.14, using data taken from IBM prices to the user that do not reflect the cost for PC related disks today. The slope of the curve is indicative of the reductions in price that have resulted from the stimulus of cost reductions of PC disks. Note that the figure uses dollars of the year. As inflation has been approximately 5% per year over the period 1965–1990, the reductions in disk price in constant dollars have been an astounding 25% per year. Note that 1 Mbyte of Model 2311 disk cost $3,300 in 1964. By 1993, the user price of the IBM 9377 disk array is $6.64 per Mbyte. There is little reason to doubt that similar cost reductions will continue at approximately the same rate over the next 10 years.

The lowest cost per Mbyte in the 1990s is found in disks used in

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000274

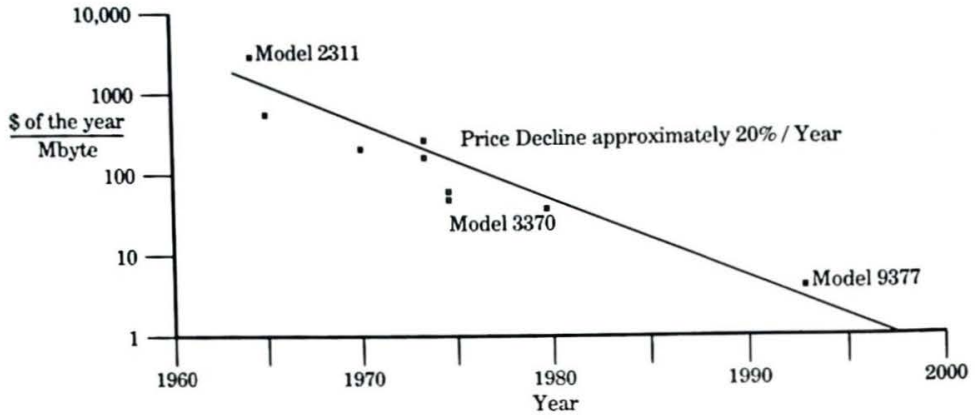


FIGURE 5.14 Disk cost history.

personal computers; high-volume applications costs are one-third to one-half those shown in Figure 5.14. Replacement disk systems for PCs cost approximately 50¢ per Mbyte in 1995. The major reason for this difference in cost, I believe, is that the cost learning curve that has been at work in the semiconductor industry is at work in the disk industry as well. Another reason for this difference is that high-volume PCs are sold with overhead and profit margins significantly lower than those found with mainframe computers.

Summary of Disk Projections

Goldstein's [GOLD87] survey, for moderate to large commercial MVS installations, shows that disk system response time in 1980 was in the range of 40 to 60 ms; an average of 46 ms is used in his studies. One survey shows response time averaged 30 ms—a decrease of 9% per year. It is projected that the trend to shorter response times will continue. Goldstein believes that a reduction of 13% per year in average response time will be required by systems and is possible to be obtained.

Goldstein combined his projections of 1987 and made a forecast of the performance of an "average installation" using a 30% per year growth rate in the installed MIPS. His projections are shown in Table 5.14.

The reduction in response time for the systems of 1990 and 1995 is predicated on caching. These caches are assumed to have a hit ratio of approximately 0.9. For any of the years, the installed disk capacity is in excess of a single drive and requires an aggregation of disk drives. The 510 Gbytes of 1990 requires 68 IBM 3380 units. Large computer installations often speak of their disk "farms" in terms of "acres of disks."

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000275

	1985	1990	1995
Installed MIPS	35	130	483
Gbytes of Disk	130	510	2092
Accesses/Sec (λ)	525	1387	3078
Response Time (t_q)	30 ms	15 ms	4.9 ms
Disk Access Time	34 ms	22 ms	22 ms
Acc/Sec/Gbyte	4.1	2.7	1.8

TABLE 5.14 Average installation disk system projections.

An approach to the design of arrays of small disks is discussed in Section 5.2.3.

5.2.2 *Disks in File I/O and Virtual Memory*

Up to this point, disks have been discussed in terms of a peripheral device. There are two ways that disks are connected to the processor and memory: (1) a file I/O system in which files of variable length are transferred or (2) as the lowest level in a virtual memory system in which pages are transferred. This section briefly discusses the issues of these two uses of disks.

The earliest computer systems used magnetic tape mass storage organized as files, usually of variable length. These files were transferred to/from the main memory under an operating system that managed files. With the advent of disk storage in the mid 1960s, the concept of a variable-length file was difficult to incorporate into operating systems. Because disks are physically divided into a hierarchy (sector, track, and surface), the sector is a natural addressable unit and is the unit of transfer between the tape, disk, and main memory; the sector is similar to a page in a virtual memory system. The operating system allocates a file to a set of consecutive sectors on the disk, hence wasting a portion of the last sector.

Files were user-defined collections of sectors that were managed in much the same way that pages are managed in real memory in a virtual memory system. The system must allocate space for new files on the disk, manage free space, and collect the garbage when blocks are de-allocated. The subject of allocation and de-allocation is discussed in Chapter 3, and reference to a book on operating systems is suggested for further details.

The most widely used operating system today, MS-DOS, is, as its name implies, a disk operating system. The user creates files that are managed, with considerable user intervention, by the operating system.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000276

It is quite likely that disk file operating systems will continue to be used for many years. In fact, as discussed in Chapter 3, the disk file operating system is used in some virtual memory systems to map the virtual address to the disk when a page fault occurs.

The first virtual memory system, Atlas, used a drum for low-level memory backed up with tape. Drum technology was rapidly replaced by disks as they became available. With a virtual memory system, the concept of a file can, in theory, disappear into the very large virtual address space. That is, as every addressable unit is within the scope of the address space, no files are required. The IBM System/38 is an example of a fileless system. The elimination of a conventional file system leads to all sorts of problems that are outside the scope of this book; however, I want to mention a few.

One problem with a fileless system is the naming of virtual pages. In a system with a very long virtual address, a page name is assigned when a page is allocated and that name is never used again. The S/38 has a 55-bit virtual page name, and if 1 million names are assigned every second, over 1,000 years would elapse before the pool of names is exhausted.

Programming systems such as LISP assume a large name space. These systems and multiprocessor configurations of large name space machines present the problem of managing names. For example, how are new names assigned and how does one program that is calling a procedure with a name find the location of that name in the system? A distributed database known as the *namespace database* is used with a network of LISP machines to solve the problem of a very large name space [BROM87].

Another problem with a pure virtual memory system is backup and recovery. With a file-based I/O system, files can be periodically archived on magnetic tape or other removable media. If the system should fail, the system is restored from the backup material. With a virtual memory system, how does one back up an address space of 2^{64}, as is the case with the S/38? Furthermore, how is garbage collection (discussed in Chapter 3) performed on a very large virtual address space?

The problems of virtual memory described above lead to the design of systems that employ their disks in a hybrid configuration. That is, virtual memory demands that paged systems are implemented on top of conventional disk I/O. An example of this hybrid trend is found in the IBM RS/6000 and the various interactive systems implemented on top of MS-DOS, such as Windows and Lotus 1-2-3. I believe that these hybrid systems will be the implementation of choice for many years. There are just too many problems with fileless pure virtual memory system implementation for them to be viable.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000277

Interconnection Topology

I now consider the issue of the connection between the processor and the disk(s). Figure 5.15 shows three typical disk system organizations [KIM86]. A single-disk system is shown in Figure 5.15(a); disk requests arrive at a rate λ, pass through the channel, and are serviced by the disk. The disk must have a service rate greater than λ, otherwise the queue in the processor will overflow. The model described above for the open system queue model applies, approximately, to this system.

Figure 5.15(b) shows a multiple-disk system with a single channel and multiple queues in the processor. By scheduling requests to the disks, the requests can be evenly distributed across the disks in a fashion similar to interleaved memory modules. Because the disks all share a single channel, rotational positioning sensing (RPS) is required to inform the queues when a request can be released to a particular disk. RPS sensing permits seek, latency, and RPS misses to be overlapped.

Figure 5.15(c) shows a system where more than one disk shares a channel, that is, a common system configuration. There is a queue in the processor for each disk, and disk requests are taken off the queues as each disk completes its current transaction. Disk requests arrive at the rate λ and are placed in the proper queues. Experience has shown that in a multiple-disk configuration there is a concentration of requests to a few of the disks, which is a distribution of requests known as *skew* [KIM86]. Reported skews for an eight-disk system are 0.388, 0.225, 0.153 0.12, 0.068, 0.054, 0.01, and 0.001 for each of the eight disks (the conditions or operating system are unknown but are believed to be a data base machine). Thus the requests serviced by each disk are λ Skew(i). If the requests to the disks are balanced, the skew for each disk would be 0.125 for the eight-disk system. Three of the disks are overused and five are underused, leading to a diminution of potential response time of the system. Models show that a system with balanced skews gives the lowest weighted average response, as would be expected.

Shortest Access Time First Scheduling

A form of scheduling that was used prior to RPS with disks is *shortest access time first scheduling* (SATF), which was used with paging drum storage in early multiprogrammed systems [DENN67]. This scheduling technique is now routinely used in disk systems. A typical paging drum performed reads and writes, bit-parallel, with one R/W head per track; there is no head positioning time, and the tracks are divided into sectors of 1,024 bits. For a 32-bit word and with 32 R/W heads, each sector has 4 Kbytes.

Each sector has a queue, as shown in Figure 5.16, and the sectors

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000278

FIGURE 5.15 Disk system organizations.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000279

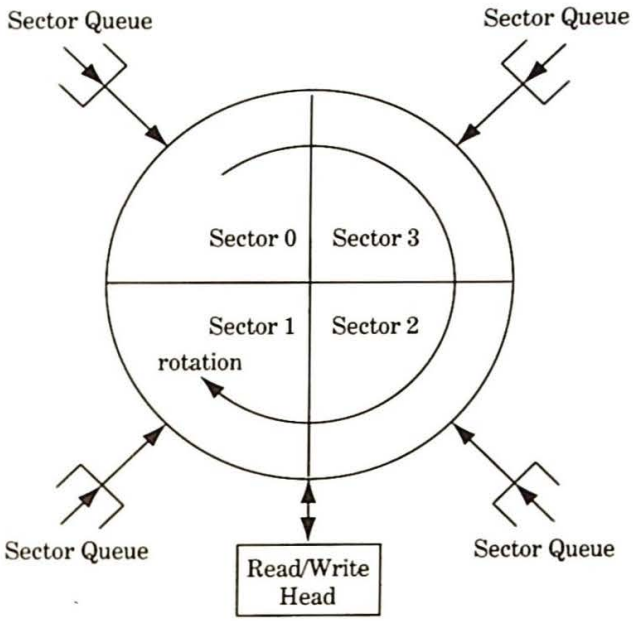


FIGURE 5.16 SATF scheduling.

rotate under the R/W heads in order $(1, 2, \ldots, n, 1, 2, \ldots)$. When, for example, sector 2 rotates to the point that the head could read or write from/to that sector, a request would be taken from the #2 queue and serviced. For a fully loaded system that has requests in all the queues, the drum would be continuously reading or writing, thereby achieving its maximum utilization.

A simple drum or disk without SATF scheduling has an average rotational latency equal to half the rotation time or an average service rate of two requests per revolution or a latency of one half-rotation. With SATF scheduling, the rotational latency is reduced because sectors are served as soon as possible but not necessarily in the order received by the drum system [FULL75]. Fuller and Baskett, whose work is modified by Pohm [POHM81], give an approximate service rate model of a paging drum using SATF scheduling:

$$\mu_{\mathrm{d}} = \frac{s}{t_{\mathrm{r}}}\left[1 - e^{\frac{-(J_{\mathrm{d}}+1)}{s+1}}\right]$$

where μ_d is the paging drum service rate; s is the number of sectors; t_r is the rotation time of the drum; and J_d is the mean number of requests in the disk queues, $1 \leq J_d \leq 10$.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000280

J_d	Service Rate	Utilization
1	106	0.110
2	155	0.161
3	201	0.210
4	244	0.254
5	286	0.298
6	324	0.337
7	361	0.375

TABLE 5.15 Paging drum service rate and utilization.

The utilization of the disk is

$$\text{utilization} = 1 - e^{-\frac{(J_d+1)}{(s+1)}}.$$

An example of the results obtained from this paging drum model for $t_r = 0.0166$ seconds (3600 RPM) and $s = 16$ is shown in Table 5.15. The number of requests in the queues, J_d, is varied from one to seven.

As the load increases, the service rate increases from 106 transactions per second to 361 transactions per second—a 3× increase in service rate. Note that the simple model for one queue is 2/0.0166 = 120 transactions per second, which approximately agrees with the Fuller model results for $J_d = 1$. The utilization of the paging drum also increases by a factor of three from 0.110 to 0.375. For very large values of J_d, the service rate approaches $s/t_r \approx 16/0.0166 \approx 960$, and the utilization approaches 1.

As noted previously, SATF scheduling only improves the service rate if there is a full queue of requests to be serviced. SATF scheduling does nothing to reduce the latency or increase the service rate for an interactive single-user system that has only a small mean number of requests in the disk queue, say, 0 or 1. SATF scheduling can be used on moving head disks if there is more than one request in the queue for the current track or cylinder. Note that sector scheduling is similar to the balanced system of Figure 5.15(b), where each R/W head is equivalent to a disk.

Multiprogrammed Systems

Large mainframe computers or servers are generally multiprogrammed systems for which the processing rate is usually more important than latency or response time. The first, known to me, multiprogramming

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000281

system is the Burroughs B5000. It was built in 1960 and was also the first commercial virtual memory system with pure segmentation [LEVI84]. A restricted form of multiprogramming is found today in systems that support foreground/background processing. The UNIX operating system is an example because its foreground process is usually interactive while its background is batch.

The basic idea of multiprogramming is that when an executing program must wait for an I/O operation, the CPU switches to another program that is resident in main memory. Another form of multiprogramming switch is based on a fixed elapsed time, say every 100 ms. The purpose of multiprogramming is to increase the utilization of the system. The number of resident programs is known as the *level (degree) of multiprogramming*.

Increased CPU utilization is not without cost. To be effective, there is processor overhead not required for a uniprogramming system such as context switching and queue management. In addition, there must be sufficient real memory to hold the additional programs and their data. Also there must be enough I/O channels (either real or shared) to support the concurrent I/O transactions generated by the resident programs.

Consider the issue of the real memory required to support multiprogramming. With a virtual memory system, memory capacity does not, at first, seem to be a problem. With a very long virtual address, each resident program has an almost unlimited memory space. However, as the real memory is finite, programs are competing for the limited real memory space and excessive paging may result. This causes another program to be switched into execution, which in its turn causes more paging until the system fails due to thrashing or lockout. Multiprogramming will not work unless there is enough real memory to support the degree of multiprogramming.

The capacity of the I/O system is also an issue. The executing program calls for an I/O operation, the next program is switched in, and it calls for an I/O operation and so on. Thus there must be sufficient I/O channels and bandwidth to support the degree of multiprogramming if 100% CPU utilization is to be achieved. Some smart I/O channels start the disk access for a read, disconnect while waiting for the disk latency, and can process another access. Nevertheless, the number of disks and channels (real or virtual) must be equal to the degree of multiprogramming and the accesses must be distributed over these disks; otherwise the queues will build up in front of one or more of the disks and further delay the access. The foreground/background type of multiprogramming may not suffer from the problem of large queues and blocking because the foreground may be inactive due to the user's thinking or otherwise not using the processor. There are a number of techniques for scheduling

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000282

the swapping of jobs that are outside the scope of this book. Reference to most books on operating systems will find a description of these techniques.

5.2.3 *Disk Arrays*

Disk arrays are receiving extensive attention from both university and industrial researchers. This attention results from the desire to overcome some of the aforementioned deficiencies of disks (cost, bandwidth, latency).

> A *disk array* is a grouping of a number of physical disks that makes these appear to applications as a single logical disk (paraphrased from [KATZ89]).

Various researchers [PATT88, HENN90] have observed that the cost learning curve has overtaken the law of scale with disk technology. In other words, the cost reductions due to vastly greater manufacturing volume produce a lower cost per bit than the reduction in cost due to the design and manufacturing of large scale disks.

In addition to potential cost advantages, disk arrays can provide, in some situations, a reduction in latency and an increase in bandwidth. If disk requests are interleaved across a number of disks in a balanced organization, a significant reduction in response time is possible if there are a number of active disk I/O requests. Also, because there are more than one active R/W head, the bandwidth is increased and the transport time can be reduced. Thus, proposals have been made to configure large-capacity disk systems from arrays of relatively small disk modules. If the overhead of interconnection can be controlled, an array system will have a lower cost than a single-disk system.

Consider the problem of designing a 100-MIPS computer installation requiring 3.7×10^9 bytes per MIPS. If this installation is served by IBM 3380 class disks having 7.5×10^9 byte capacity, approximately 50 disks are required. On the other hand, if an array of 300×10^6 byte modules are used, approximately 1,250 disks are required. Thus, disk array design must comprehend arraying large numbers of disks to support mainframe computers.

Arrays of disks will also have a higher transfer bandwidth because of the number of R/W heads that can be active at one time. The bandwidth increase is a function of the design approach, to be discussed below. It should go without saying that as the bandwidth of the disks is increased with arrays of disks that the bandwidth of the channel must also in-

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000283

crease. In all of the discussions to follow it is assumed that channel bandwidth is not a limiting factor.

There are a number of instances of arraylike operation with conventional drums and disks. Special purpose drums that provide equal bandwidth for access of a matrix [CRAG68] and operating systems that use more than one disk channel are found on supercomputers such as SCOPE OS [JOHN84]. Thus the current thinking on disk arrays builds upon these early efforts. A recent product announcement from IBM is for the IBM 9337 Disk Array Subsystem [IBM 92], which offers up to seven 3.5-inch disk drives. One model uses 542-Mbyte disks while the other uses 970-Mbyte disks. The implementation is RAID-5, which is discussed later in this section.

Disk Array Taxonomy

There are a number of design options for a disk array that are described in a taxonomy with three dimensions shown in Figure 5.17. These dimensions are (1) the *degree of interleaving*, which concerns the layout of the sectors on the disks; (2) the rotation of the disks, which can be synchronous or asynchronous; and (3) R/W heads, which can be positioned either independently or as a group. This disk array taxonomy is based on [KATZ89].

There are three possible design cases of rotation and arm movement (shown in Figure 5.17) that are discussed later. There is no known implementation of a synchronous disk with independent arm movement Two definitions from [KATZ89] apply to disk arrays.

Stripe unit (SU) is the unit of data interleaving, that is, the amount of data that is placed on one disk before data is placed on the next disk. The stripe unit may be as small as one byte [KIM86] and as large as a disk sector. In terms of interleaved memory discussed in Section 5.1, a stripe unit is an addressable unit.

Data stripe (DS) is a sequence of logically consecutive stripe units.

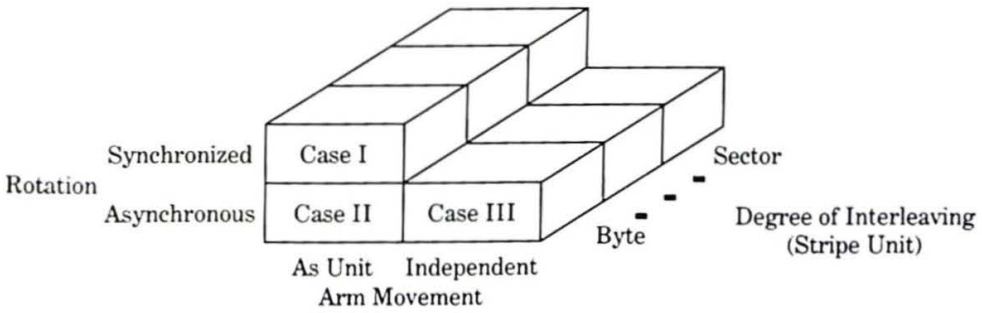


FIGURE 5.17 Disk array design options.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000284

A logical I/O request to a disk array corresponds to a data stripe. In terms of an interleaved memory (discussed earlier), a data stripe is a vector of AUs with a stride of 1.

Degree of interleaving (m) is equal to the number of disks in the system over which the stripe unit is stored. This term is equivalent to the interleaving factor, m (discussed in Section 5.1 in reference to interleaved memory modules).

Degree of Interleaving

The degree of interleaving depends on the number of disks in the array and the size of the stripe unit. In other words, at any one time a number of R/W heads can be active depending on the number of disks in the array. The stripe unit can be, for example, a byte, word, quadword, or sector. The selection of the stripe unit size is a tradeoff between the number of disks in the array, the size of the data stripe and whether it is of fixed or variable length.

Figure 5.18 shows, for expository purposes, a disk array system with four disks, with one track per disk, and with four 1-Kbyte segments per track. The stripe unit is a sector. There are obviously more tracks, but only one track per disk is active at the time in this example.

Assume that a data stripe or file consisting of five stripe units, or five sectors, is to be written to the disk array. There are four possible assignments of these five sectors to the four disks; this assignment is similar to skewed interleaving of memory modules. The time required

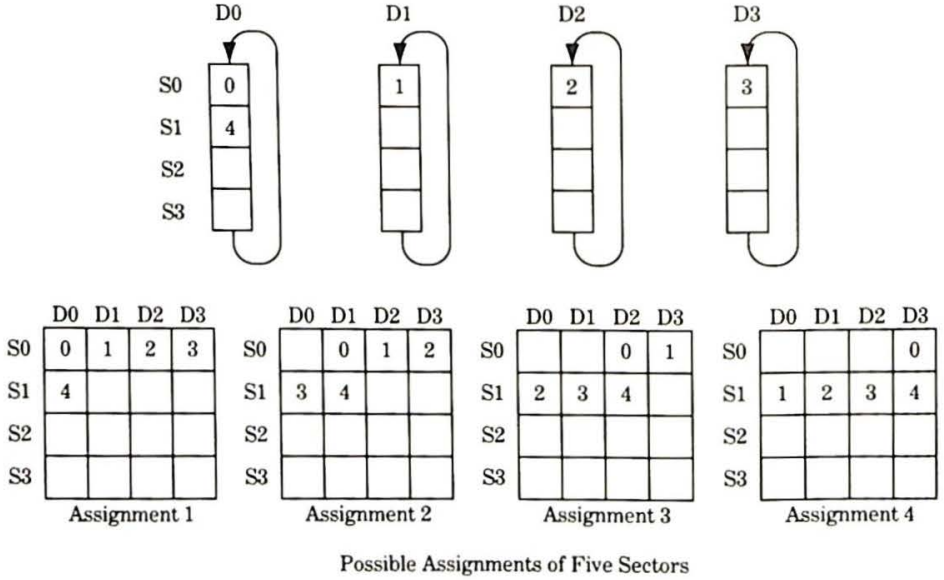


FIGURE 5.18 Disk array layout.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000285

to write these five segments is the latency of the disks (rotation, head position) plus the time to transfer two segments rather than the time to transfer five segments if all of the segments are on one track of one disk. Note that the four disks are assumed to be accessed in parallel. Another assumption is that the disks are rotating synchronously, that is, the same sectors of all disks are under the R/W head at the same time. The other case (to be discussed later) is for asynchronous rotation.

Case I. Synchronous Disk With As-Unit Arm Movement

The synchronous disk organization, as shown in Figure 5.19, is an organization similar to the single-disk system of Figure 5.14(a). Fiducial marks are provided to the drive controller in order to maintain synchrony and to identify the beginning of the first sector, similar to RPS. The m heads are reading or writing in parallel, and the same sector is under the R/W heads of each disk at the same time. The access delays of head positioning and rotation are in parallel. The channel issues a request and, when the correct sector appears under the R/W heads, the request is honored. Kim [KIM86] named this system organization *synchronous disk interleaving* and assumes that the stripe unit is a byte—an assumption that is not a requirement for this organization.

The number of segment read/write times is

$$\text{segment times} = \left\lceil \frac{\text{data stripe size}}{\text{stripe unit size} \times \text{sector size} \times \text{number of disks}(m)} \right\rceil$$

For example, if a data stripe is 16 Kbytes, a stripe unit is 1 Byte, the sector size is 1 Kbyte, and $m = 4$, the data stripe can be transferred in four sector times as compared to sixteen sector times if disk interleaving is not used.

A conversion buffer is required to format a data stripe into the serial

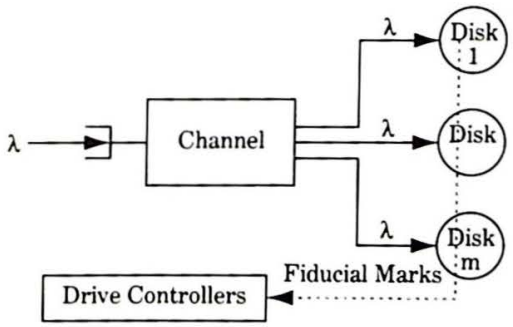


FIGURE 5.19 Synchronous disk organization.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000286

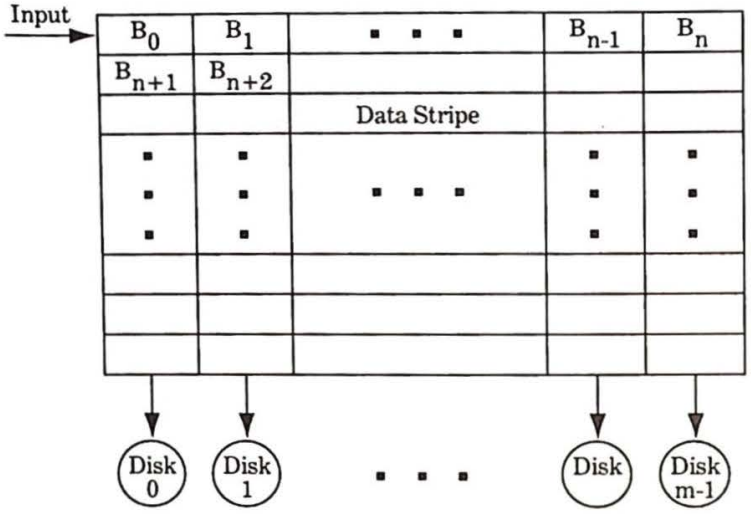


FIGURE 5.20 Conversion buffer.

streams required for each disk, as shown in Figure 5.20. This buffer design assumes that the incoming data stripe is serial by byte and the stripe unit is a byte. After the buffer is loaded horizontally, the buffer is then unloaded vertically into the disk units [KIM86].

There are a number of implementation problems with synchronous disks. The paramount one is providing synchronization of the disk spindles themselves. Kim [KIM86] discusses the problems of using synchronous motors driven from a common clock with feedback control (the fiducial marks noted in Figure 5.19). Another problem is coping with sectors that go bad after the disk array is placed in service. Reliability and failures are discussed later in this section.

Kim provides extensive model results on performance parameters such as service time, peak transfer rates, queuing delays, disk utilization, and weighted average response time. The effect of block (data stripe) size on response time and service rate is also evaluated [KIM86]. Chen and Patterson [CHEN90] have also investigated the impact of the size of the stripe unit on the service rate of this organization. The most significant parameter is the degree of concurrency of the input requests to the disk system.

Case II. Asynchronous Disk With As-Unit Arm Movement

An asynchronous disk organization is shown in Figure 5.21. With this system, after the data stripe has been loaded into a conversion buffer, the access operations (head positioning) of the disks are initiated and

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000287

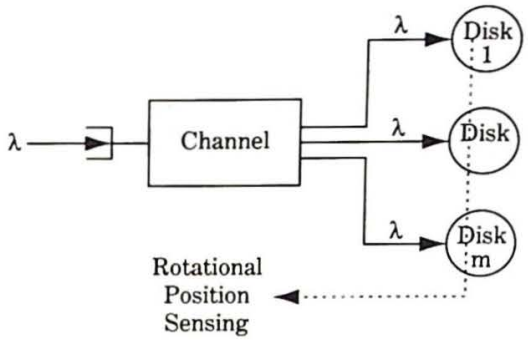


FIGURE 5.21 Asynchronous disk organization.

RPS informs the channel when each sector of each disk is available to receive its allocated stripe units. This system organization has the major advantage of not requiring synchronization of the spindles and the disadvantage of potentially lower performance due to the lack of concurrency in the read/write operations between the disks.

The conversion buffer of Figure 5.20 is modified so that the stripe units are read or written to/from the sectors of the disk at the correct time. This system organization has been named *asynchronous disk interleaving* by M. Y. Kim et al. [KIM87] and *disk striping* by [SALE86].

Case III. Asynchronous Disk With Independent Arm Movement

This system organization not only has asynchronous disk rotation but each arm of each disk can be independently positioned. This system organization appears to provide the greatest possibility for concurrency and therefore should improve the disk system's performance. This system organization is also the most consistent with the concept of an array of independent disk modules as would be created by racking up a multiplicity of PC disk modules.

Kim and Tantawi [KIM87] modeled this system configuration, and they point out that the effective rotational latency approaches that of the disk having the longest latency for a data stripe. That is, the expected seek plus rotational latency is greater than that expected of a synchronous system with unit arm movement. This result is intuitive in that the complete data stripe cannot be transferred until the last stripe units are transferred. Another variant of this system organization is named *declustering* under the assumption that the stripe unit is a sector [LIVN87].

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000288

Data Redundancy and MTBF

A direct consequence of the use of a disk array is the danger of reducing the reliability of the disk system, that is, decreasing the *mean time between failure* (MTBF). A conventional multiple-disk system, as shown in Figure 5.12(b), will have a disk failure from time to time, with the consequence that a portion of the stored data is lost. When the disks are arrayed as described above, a failure of one disk can corrupt all of the stored data because the data is interleaved.

In general, the MTBF of a disk array system is lower than that of a single unified disk. The reason is that disk units have roughly equal MTBFs. For example, the IBM 3380, a large mainframe disk, has an MTBF of 52,000 hours while the Conner CP3100, a smaller PC class disk, has an MTBF of 30,000 hours [KATZ89].

Katz illustrates the MTBF problem by comparing the MTBF of the IBM 3380 to an array of CP3100 disks that have the same storage capacity. Equal capacity, not considering redundancy, is achieved with 75 of the CP3100 disks that have an MTBF of 16.6 days. The single IBM 3380 has an MTBF of 2,166 days. The design of an array of disk modules must anticipate a high failure rate and the corruption of all or part of the stored data. Thus, an important and vital consideration in disk array design is providing a high level of data reliability in the face of poor hardware reliability or low MTBF.

The goal of schemes to cope with the low MTBF of disk arrays is to "fail soft," that is, operation continues while a repair is made. Fail soft schemes all depend upon some form of redundancy. A simple example is the use of error correcting codes in memory or communications systems. The design issue thus becomes one of choosing the best form of redundancy. There is an extensive body of knowledge on the use of redundancy for error detection and correction to give fail soft operation [SIEW82].

Kim [KIM86] proposed expanding the conversion buffer of Figure 5.20 to include horizontal *error correcting codes* (ECC). Check words are stored in an added disk along with vertical end-of-block check bytes. The described form of *X-Y* check bits has been used extensively in magnetic tape drives for many decades [MATI77]. With EBCDIC encoding, there is a parity bit for each byte (giving nine tracks) and a longitudinal parity byte recorded at the end of the record.

Patterson et al. have created a taxonomy of six disk array configurations that can be used to achieve data reliability or fail soft operation [PATT88, KATZ 89]. The term RAID (redundant array of inexpensive disks) is used to describe these systems briefly discussed in the following paragraphs. The six RAID configurations are shown in Figures 5.22 and 5.23.

RAID 1. Basically this reliability technique is simple redundancy—a traditional approach to the reliability of any disk system, arrayed or not. RAID 1 is also known as *mirrored disks* and *shadowing*. All reads

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000289

FIGURE 5.22 RAID 1 to RAID 4 designs.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000290

and writes go to/from two identical disks, and the read and write bandwidth is not compromised. However, the cost is 2 times that of a nonredundant system. Errors are detected by comparing the results of both reads. If the comparison fails, the operating system must intervene to determine the correct data, which is a fundamental problem with duplicate redundancy. This organization may give a marginal performance increase if a read operation selects the disk unit that has the shortest latency using RPS information from both disks.

RAID 2. The data stripe is interleaved bitwise across a number of disks as shown in Figure 5.22. The redundancy is provided by horizontal ECC bits stored in separate disks. The use of ECC across a wide memory word has been used in computers using semiconductor memories for several decades. ECC identifies not only that an error has been made but the location of the bit(s) that are in error. The number of bits, thus ECC disks, required to detect two errors and correct one error (DEDSEC) must satisfy the relationship $2^c \geq d + c + 1$ where c is the number of check bits and d is the number of data bits [HAMM50]. A discussion of the number of required ECC bits is outside the scope of this book, but there are many good references. However, the degree of redundancy required for this scheme is $O(\log_2 m)$. For example, if the data stripe is applied across eight disks, four disks are needed to store the DEDSEC bits. The DEDSEC bits must themselves be interleaved because of the potential for a failure in one of the redundant disks. Simple parity for detecting but not correcting an error can be accomplished with one redundant disk.

Because a fraction of a data stripe will reside in a sector and the data in the sectors of all disks participate in the generation of the ECC bits, writes must be conducted as Read-Modify-Write operations, which reduce the bandwidth of the system. Read-Modify-Write operations are required for any memory where the memory word is larger than the addressable unit.

RAID 3. This method, as shown in Figure 5.22, of providing data correction in the face of a disk failure is a simplification of RAID 2. The stripe unit is a bit or byte. Because there is significant data checking provided on each of the disk drives, it is not necessary to have full ECC capability as with RAID2, simple parity will suffice. The redundant disk contains simple parity that is used to identify and correct the bit(s) in the disk that has failed or produced an error. The failed disk is identified by the internal checking hardware in each of the disks.

For example, assume a byte is interleaved across eight disks with even parity in the parity disk, as shown in Table 5.16. If disk 4 fails, the value of the bit is unknown: it could be a one or zero. The fact that the disk has failed is known, and the parity bit is 1 while there is an

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000291

	Disk								
	1	2	3	4	5	6	7	8	*P*
Normal	1	0	1	1	1	0	0	1	1
Disk 4 fails	1	0	1	x	1	0	0	1	0
Restored	1	0	1	1	1	0	0	1	1

TABLE 5.16 RAID 3 example.

even number of remaining good bits. From this information, the true value of 1 can be reconstructed.

The RAID 3 scheme requires Read-Modify-Write operations to compute the parity bit. However, the incremental hardware cost is small as the degree of redundancy is fixed at one disk module for any degree of disk interleaving.

RAID 4. This redundancy method is a modification of RAID 3 in that the stripe unit is a sector rather than a bit or byte. However, RAID 4 does not interleave across the disks, thus there is no improvement in transfer rate for a single stripe unit compared to a single disk. Some of this performance loss can be recovered with independent head movement that permits more then one read/write to be in process at once. For writes to the disk, the Read-Modify-Write operation is still required Parity is provided at the byte level in the parity disk, and internal checking of each disk unit is used to identify the location of the data failure.

RAID 5. This redundant disk array system, shown in Figure 5.23, distributes the contents of the parity disk of RAID 3 or RAID 4 across all of the disks. The internal checking of RAID 3 and RAID 4 is not shown. A fraction of all of the disks is used to store parity bits rather than a separate disk; the amount of storage is the same even though a parity disk is not used. There are two failure modes that can be handled. First, a disk can fail that does not contain the parity of the accessed data stripe, this case is identical to RAID 3 or 4. For example, for the stripe unit identified •, if disk 1 fails, the data can be recovered because parity is not lost. Second, if the disk that fails contains the parity of the accessed data stripe, the correct data can still be recovered since only the parity bit is unknown. For example, if disk m fails the identified stripe unit can be recovered. As noted previously, the IBM 9337 is a RAID 5 system.

RAID 6. This disk array system extends the two-dimensional organization (one dimension of disks and one dimension of sector) of RAID 5

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000292

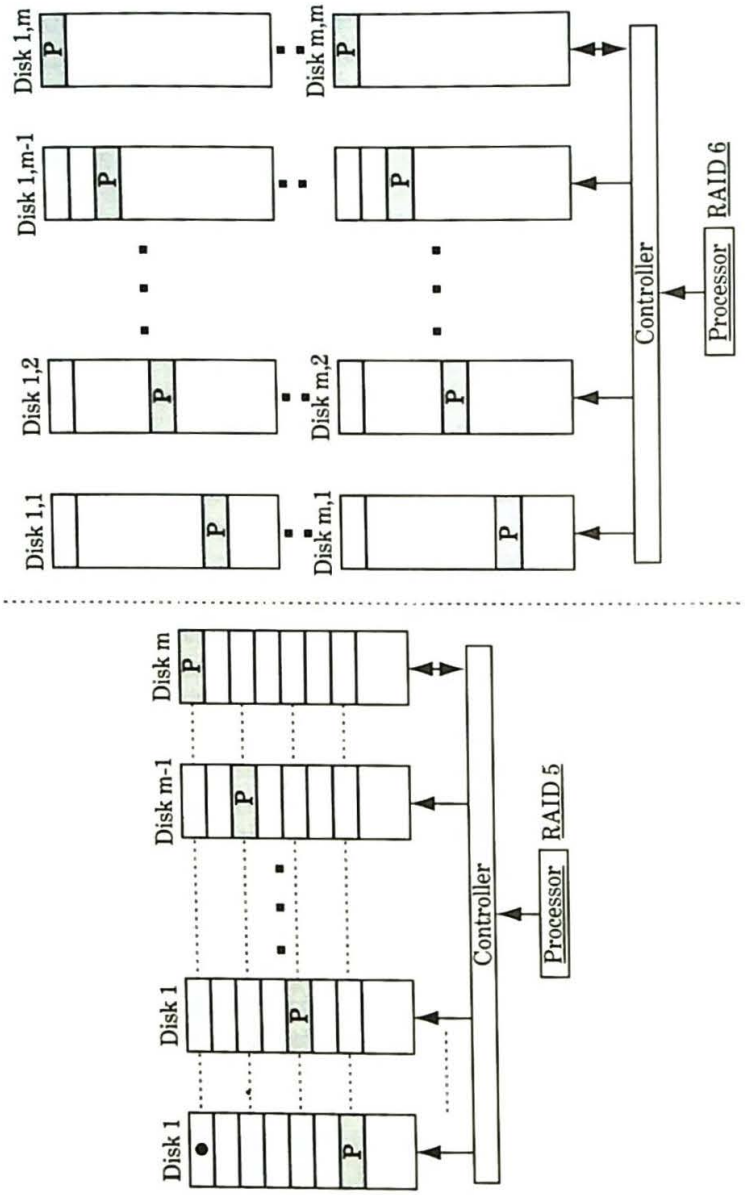


FIGURE 5.23 RAID 5 and RAID 6 designs.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000293

to the three-dimensional (two dimensions of disks and one dimension of sectors). Each disk now contains both row and column parity plus the parity in the segments. Thus, RAID 6, due to its two-dimensional redundancy, can sustain two disk failures and still function. The RAID 6 redundant system checks the data in the column and does not depend on the internal checking hardware. Multidimensional disk arrays and their redundancy codes are discussed in [GIBS89].

5.2.4 *Disk Caches*

The very long latency of disks leads to significant delays for servicing page faults and requests for I/O. As with caches that effectively reduce the latency of main memory, a method for reducing the effect of disk latency is via use of the disk cache. Discussion in Section 5.2.1 shows how the use of software buffers has reduced the number of actual accesses to the disk. The question addressed in this section is whether or not a hardware-managed cache would be effective in reducing latency while transparent to the operating system.

I have no knowledge of specific disk cache implementations, thus I will speculate on their organization. A disk cache would probably be organized early select, direct access. The disk cache sector would probably be a disk track composed of a number of disk sectors that would be the disk cache blocks. The reason for this organization is that the transfer of a track from the disk to the cache sector would not have rotational latency because the read can start at any point of the track and can be mapped into the random access cache sector. A transfer from the cache that evicts a sector to the disk can also start at any point in the track since the source is a random access memory. A track size segment has been used with some paging drums to eliminate rotation latency.

A major design issue is where to place the disk cache in the processor-disk path. Figure 5.24 shows the typical topology for connecting disks to a processor. There can be a multiplicity of channels, a multiplicity of storage controllers, and a multiplicity of string controllers, each connected to a multiplicity of disk units.

The options for placing a disk cache are: (1) main memory, (2) channels, (3) storage controllers, (4) string controllers, or (5) at each of the disks. As the location of the caches is moved toward the processor, more disk accesses are handled by the cache and the utilization of the cache increases. For example, when placed at the main memory, all disk references pass through the cache. If placed at one of the disks, only the accesses to that disk are handled. Because of disk skew, it can be argued that the disk cache should be placed at the disk that has the heaviest

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000294

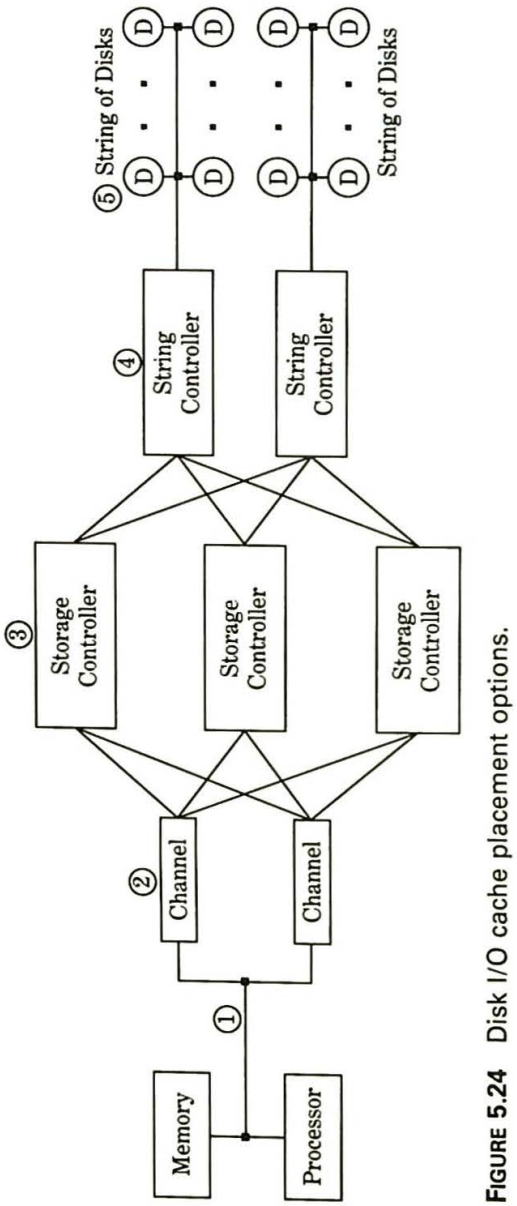


FIGURE 5.24 Disk I/O cache placement options.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000295

load. However, Smith [SMIT85] showed that skew is dynamic and changes from disk to disk over time. One placement, therefore, may be far from ideal for a varying workload on the system. A cache would then be required for all disks. For disk manufacturers, incorporating the cache into the disk permits the manufacturer to sell the cache as well as the disk. However, this is not necessarily the best for overall system cost and/or performance. As discussed below, for a case where there is a fixed amount of memory available for a cache, the memory should be placed at the processor.

Smith's paper on disk caches gives miss ratio data derived from trace data on an IBM S/360/91, 370/165, 370/168s and an Amdahl 470V/6 [SMIT85]. The trace data is then used to evaluate disk cache miss rates for various cache sizes, organizations, and locations in the system. Of the five possible locations for caches shown in Figure 5.24, three locations are evaluated. The results of the simulation are shown in Table 5.17.

Global. One cache at the main memory (location 1).
Controller. Multiple caches, one with each I/O Channel (location 2).
Device. Multiple caches, one with each disk (location 5).

The results of these simulations indicate that for the evaluated disk caches sizes, the best miss ratio is obtained with global disk caches. However, as would be expected, there can be significant variations in the miss ratios for different workloads. Ousterhout [OUST85] reports that for measurements on VAX with a 2-Mbyte disk cache, only 17.7% of all requests are actually served by the disk. Goldstein [GOLD87] uses a miss ratio of 0.1 to justify the use of disk caches in the future—a figure

Total Cache Capacity (Mbytes)	Crocker Bank Business Data Processing			SLAC Scientific Calculations		
	Global	Controller	Device	Global	Controller	Device
1	0.316	0.475	0.610	0.330	0.601	0.630
2	0.259	0.365	0.450	0.226	0.414	0.496
4	0.225	0.275	0.330	0.146	0.326	0.370
8	0.197	0.233	0.266	0.099	0.227	0.271
16	0.172	0.203	0.224	0.070	0.136	0.174
32	0.150	0.177	0.199	0.050	0.085	0.109
64	0.139	0.155	0.175	0.033	0.061	0.071

Note. The total cache capacity is evenly distributed over the controllers or devices.

TABLE 5.17 Disk cache miss ratios.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000296

that is consistent with this data. Therefore, it seems that a miss ratio in the range of 0.1 constitutes a reasonable estimate for design purposes.

There are a number of system issues concerning disk caches. In general, these are the same issues addressed in Chapter 2 on multilevel caches. A disk cache should be write back because of the disk latency, and multilevel inclusion should be enforced. In addition, there are operating system issues such as operating system control of the disk cache and error recovery. Additional issues are problems in coherency and reliability with the disk cache operating in write-back mode. Most of these design issues are discussed at length in [SMIT85]. It is interesting to note that the software-controlled disk cache of MS-DOS places such considerations under user control.

Smith presents an interesting discussion on alternatives to disk caches. Today, with large main memories and large buffers, the disk cache may not be the most effective method of reducing disk accesses. Alternatives discussed by Smith include large cache block sizes that benefit from spatial locality, larger main memory to reduce the need to page temporary files, solid state drums and disks, using the virtual memory system to map the disk address space into the program address space, and others. If a designer has a fixed additional memory, how should this memory be allocated; should it be used as a disk cache, more main memory, or as program managed buffers? The conclusion seems to be that system simplification issues are more important than performance issues in answering this design question.

While not a disk cache, an extended memory as a paging device (solid state drum) has been used with a number of supercomputers. These devices have been very effective in reducing the time required for page-out and page-in when a task is swapped out in a multiprogramming environment.

Advanced Micro Devices, Inc., Pensando Systems, Inc. - Ex. 1022, Page 000297