A.1 THE DECIMAL SYSTEM

In everyday life, we use a system based on decimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9) to represent numbers and refer to the system as the decimal system. Consider what the number 83 means. It means eight tens plus three:

$$83 = (8 \times 10) + 3$$

The number 4728 means four thousands, seven hundreds, two tens, plus eight:

$$4728 = (4 \times 1000) + (7 \times 100) + (2 \times 10) + 8$$

The decimal system is said to have a **base**, or **radix**, of 10. This means that each digit in the number is multiplied by 10 raised to a power corresponding to that digit's position:

$$83 = (8 \times 10^1) + (3 \times 10^0)$$

$$4728 = (4 \times 10^3) + (7 \times 10^2) + (2 \times 10^1) + (8 \times 10^0)$$

The same principle holds for decimal fractions but negative powers of 10 are used. Thus, the decimal fraction 0.256 stands for 2 tenths plus 5 hundredths plus 6 thousandths:

$$0.256 = (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

A number with both an integer and fractional part has digits raised to both positive and negative powers of 10:

$$472.256 = (4 \times 10^2) + (7 \times 10^1) + (2 \times 10^0) + (2 \times 10^{-1}) + (5 \times 10^{-2}) + (6 \times 10^{-3})$$

In general, for the decimal representation of $X = \{\ldots d_2 d_1 d_0 . d_{-1} d_{-2} d_{-3} \ldots\}$, the value of X is

$$X = \sum_i (d_i \times 10^i)$$

A.2 THE BINARY SYSTEM

In the decimal system, 10 different digits are used to represent numbers with a base of 10. In the binary system, we have only two digits, 1 and 0. Thus, numbers in the binary system are represented to the base 2.

To avoid confusion, we will sometimes put a subscript on a number to indicate its base. For example, 83_{10} and 4728_{10} are numbers represented in decimal notation or, more briefly, decimal numbers. The digits 1 and 0 in binary notation have the same meaning as in decimal notation:

$$0_2 = 0_{10}$$

$$1_2 = 1_{10}$$

APPENDIX A

NUMBER SYSTEMS

18.15 The following FORTRAN program is to be executed on a computer, and a parallel version is to be executed on a 32-computer cluster.

```
L1:      DO 10 I = 1, 1024
L2:           SUM(I) = 0
L3:           DO 20 J = 1, I
L4: 20             SUM(I) = SUM(I) + I
L5: 10    CONTINUE
```

Suppose lines 2 and 4 each take two machine cycle times, including all processor and memory-access activities. Ignore the overhead caused by the software loop control statements (lines 1, 3, 5) and all other system overhead and resource conflicts.

a. What is the total execution time (in machine cycle times) of the program on a single computer?

b. Divide the I-loop iterations among the 32 computers as follows: Computer 1 executes the first 32 iterations (I = 1 to 32), processor 2 executes the next 32 iterations, and so on. What are the execution time and speedup factor compared with part (a)? (Note that the computational workload, dictated by the J-loop, is unbalanced among the computers.)

c. Explain how to modify the parallelizing to facilitate a balanced parallel execution of all the computational workload over 32 computers. By a balanced load is meant an equal number of additions assigned to each computer with respect to both loops.

d. What is the minimum execution time resulting from the parallel execution on 32 computers? What is the resulting speedup over a single computer?

18.16 Consider the following two versions of a program to add two vectors:

```
L1:      DO 10 I = 1, N
L2:        A(I) = B(I) + C(I)
L3: 10   CONTINUE
L4:      SUM = 0
L5:      DO 20 J = 1, N
L6:        SUM = SUM + A(J)
L7: 20   CONTINUE
```

```
      DOALL K = 1, M
        DO 10 I = L(K-1) + 1, KL
          A(I) = B(I) + C(I)
10      CONTINUE
        SUM(K) = 0
        DO 20 J = 1, L
          SUM(K) = SUM(K) + A(L(K-1) + J)
20      CONTINUE
      ENDALL
```

a. The program on the left executes on a uniprocessor. Suppose each line of code L2, L4, and L6 takes one processor clock cycle to execute. For simplicity, ignore the time required for the other lines of code. Initially all arrays are already loaded in main memory and the short program fragment is in the instruction cache. How many clock cycles are required to execute this program?

b. The program on the right is written to execute on a multiprocessor with M processors. We partition the looping operations into M sections with $L = N/M$ elements per section. DOALL declares that all M sections are executed in parallel. The result of this program is to produce M partial sums. Assume that k clock cycles are needed for each interprocessor communication operation via the shared memory and that therefore the addition of each partial sum requires k cycles. An l-level binary adder tree can merge all the partial sums, where $l = \log_2 M$. How many cycles are needed to produce the final sum?

c. Suppose $N = 2^{20}$ elements in the array and $M = 256$. What is the speedup achieved by using the multiprocessor? Assume $k = 200$. What percentage is this of the theoretical speedup of a factor of 256?

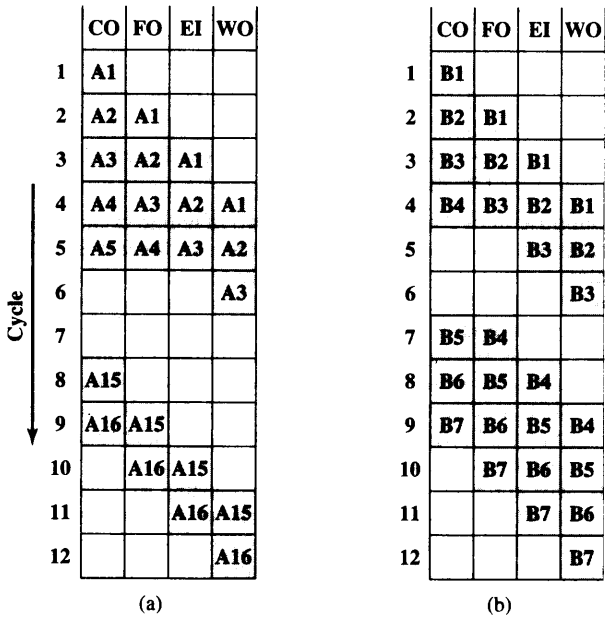

Figure 18.24 Two Threads of Execution

where R1, R2, and R3 are processor registers, and $\alpha,\beta,\gamma,\theta$ are the starting main memory addresses of arrays B(I), C(I), A(I), and D(I), respectively. Assume four clock cycles for each Load or Store, two cycles for the Add, and eight cycles for the Multiplier on either a uniprocessor or a single processor in an SIMD machine.

a. Calculate the total number of processor cycles needed to execute this code segment repeatedly 64 times on a SISD uniprocessor computer sequentially, ignoring all other time delays.
b. Consider the use of an SIMD computer with 64 processing elements to execute the vector operations in six synchronized vector instructions over 64-component vector data and both driven by the same-speed clock. Calculate the total execution time on the SIMD machine, ignoring instruction broadcast and other delays.
c. What is the speedup gain of the SIMD computer over the SISD computer?

18.13 Produce a vectorized version of the following program:

```
      DO 20 I = 1, N
      B(I, 1) = 0
      DO 10 J = 1, M
      A(I) = A(I) + B(I, J) × C(I, J)
10    CONTINUE
      D(I) = E(I) + A(I)
20    CONTINUE
```

18.14 An application program is executed on a nine-computer cluster. A benchmark program took time T on this cluster. Further, it was found that 25% of T was time in which the application was running simultaneously on all nine computers. The remaining time, the application had to run on a single computer.

a. Calculate the effective speedup under the aforementioned condition as compared to executing the program on a single computer. Also calculate α, the percentage of code that has been parallelized (programmed or compiled so as to use the cluster mode) in the preceding program.
b. Suppose that we are able to effectively use 18 computers rather than 9 computers on the parallelized portion of the code. Calculate the effective speedup that is achieved.

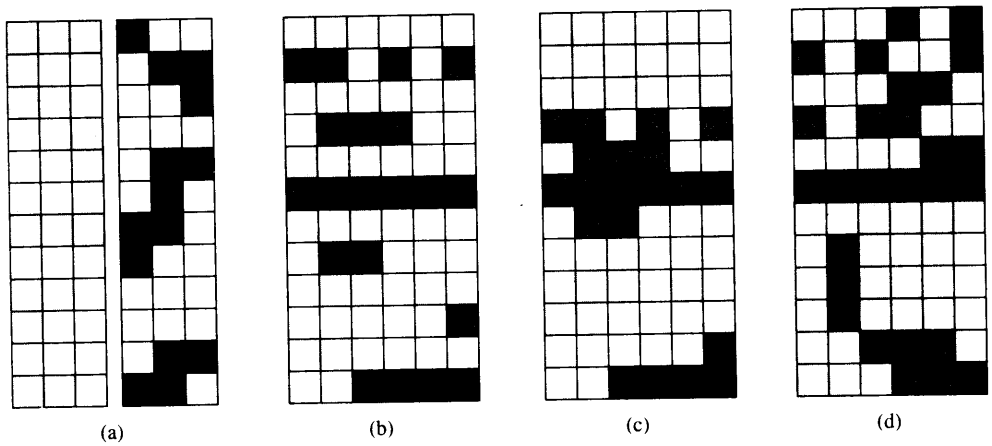

Figure 18.23 Diagram for Problem 18.9

lines in the data cache among lines replaced. Assume a write-back policy and determine the effective memory access time in terms of the parameters just defined.

b. Now assume a bus-based SMP in which each processor has the characteristics of part (a). Every processor must handle cache invalidation in addition to memory reads and writes. This affects effective memory access time. Let f_{inv} be the fraction of data references that cause invalidation signals to be sent to other data caches. The processor sending the signal requires t clock cycles to complete the invalidation operation. Other processors are not involved in the invalidation operation. Determine the effective memory access time.

18.9 What organizational alternative is suggested by each of the illustrations in Figure 18.23?

18.10 In Figure 18.8, some of the diagrams show horizontal rows that are partially filled. In other cases, there are rows that are completely blank. These represent two different types of loss of efficiency. Explain.

18.11 Consider the pipeline depiction in Figure 12.13b, which is redrawn in Figure 18.24a, with the fetch and decode stages ignored, to represent the execution of thread A. Figure 18.24b illustrates the execution of a separate thread B. In both cases, a simple pipelined processor is used.

a. Show an instruction issue diagram, similar to Figure 18.8a, for each of the two threads.

b. Assume that the two threads are to be executed in parallel on a chip multiprocessor, with each of the two processors on the chip using a simple pipeline. Show an instruction issue diagram similar to Figure 18.8k. Also show a pipeline execution diagram in the style of Figure 18.24.

c. Assume a two-issue superscalar architecture. Repeat part (b) for an interleaved multithreading superscalar implementation, assuming no data dependencies. *Note:* there is no unique answer; you need to make assumptions about latency and priority.

d. Repeat part c for a blocked multithreading superscalar implementation.

e. Repeat for a four-issue SMT architecture.

18.12 The following code segment needs to be executed 64 times for the evaluation of the vector arithmetic expression: $D(I) = A(I) + B(I) \times C(I)$ for $0 \leq I \leq 63$.

Load R1, B(I)	/R1 ← Memory (α + I)/
Load R2, C(I)	/R2 ← Memory (β + I)/
Multiply R1, R2	/R1 ← (R1) × (R2)/
Load R3, A(I)	/R3 ← Memory (γ + I)/
Add R3, R1	/R3 ← (R3) + (R1)/
Load D1, R3	/ Memory (θ + I) ← (R3)/

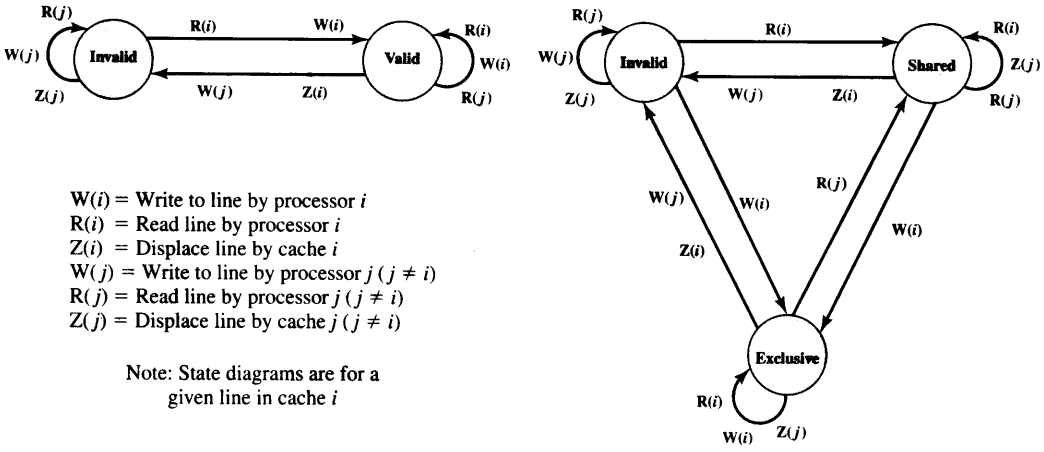

Figure 18.22 Two Cache Coherence Protocols

3. P1 writes to x (label the line in P1's cache x″).
4. P2 reads x.

18.5 Figure 18.22 shows the state diagrams of two possible cache coherence protocols. Deduce and explain each protocol, and compare each to MESI.

18.6 Consider an SMP with both L1 and L2 caches using the MESI protocol. As explained in Section 18.3, one of four states is associated with each line in the L2 cache. Are all four states also needed for each line in the L1 cache? If so, why? If not, explain which state or states can be eliminated.

18.7 An earlier version of the IBM mainframe, the S/390 G4, used three levels of cache. As with the z990, only the first level was on the processor chip [called the processor unit (PU)]. The L2 cache was also similar to the z990. An L3 cache was on a separate chip that acted as a memory controller, and was interposed between the L2 caches and the memory cards. Table 18.4 shows the performance of a three-level cache arrangement for the IBM S/390. The purpose of this problem is to determine whether the inclusion of the third level of cache seems worthwhile. Determine the access penalty (average number of PU cycles) for a system with only an L1 cache, and normalize that value to 1.0. Then determine the normalized access penalty when both an L1 and L2 cache are used, and the access penalty when all three caches are used. Note the amount of improvement in each case and state your opinion on the value of the L3 cache.

18.8 a. Consider a uniprocessor with separate data and instruction caches, with hit ratios of H_d and H_i, respectively. Access time from processor to cache is c clock cycles, and transfer time for a block between memory and cache is b clock cycles. Let f_i be the fraction of memory accesses that are for instructions, and f_d is the fraction of dirty

Table 18.4 Typical Cache Hit Rate on S/390 SMP Configuration [MAK97]

Memory Subsystem	Access Penalty (PU cycles)	Cache Size	Hit Rate (%)
L1 cache	1	32 KB	89
L2 cache	5	256 KB	5
L3 cache	14	2 MB	3
Memory	32	8 GB	3

Problems

18.1 Let α be the percentage of program code that can be executed simultaneously by n processors in a computer system. Assume that the remaining code must be executed sequentially by a single processor. Each processor has an execution rate of x MIPS.

a. Derive an expression for the effective MIPS rate when using the system for exclusive execution of this program, in terms of n, α, and x.

b. If $n = 16$ and $x = 4$ MIPS, determine the value of α that will yield a system performance of 40 MIPS.

18.2 A multiprocessor with eight processors has 20 attached tape drives. There are a large number of jobs submitted to the system that each require a maximum of four tape drives to complete execution. Assume that each job starts running with only three tape drives for a long period before requiring the fourth tape drive for a short period toward the end of its operation. Also assume an endless supply of such jobs.

a. Assume the scheduler in the OS will not start a job unless there are four tape drives available. When a job is started, four drives are assigned immediately and are not released until the job finishes. What is the maximum number of jobs that can be in progress at once? What are the maximum and minimum number of tape drives that may be left idle as a result of this policy?

b. Suggest an alternative policy to improve tape drive utilization and at the same time avoid system deadlock. What is the maximum number of jobs that can be in progress at once? What are the bounds on the number of idling tape drives?

18.3 Can you foresee any problem with the write-once cache approach on bus-based multiprocessors? If so, suggest a solution.

18.4 Consider a situation in which two processors in an SMP configuration, over time, require access to the same line of data from main memory. Both processors have a cache and use the MESI protocol. Initially, both caches have an invalid copy of the line. Figure 18.21 depicts the consequence of a read of line x by Processor P1. If this is the start of a sequence of accesses, draw the subsequent figures for the following sequence:

1. P2 reads x.
2. P1 writes to x (for clarity, label the line in P1's cache x').

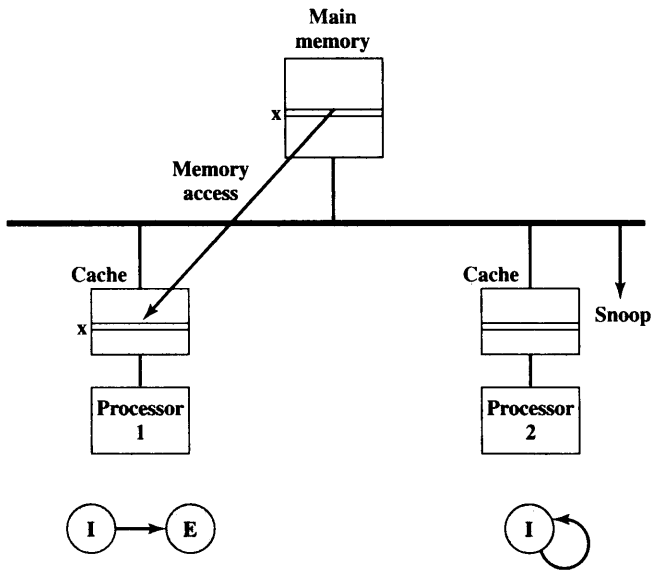

Figure 18.21 MESI Example: Processor 1 Reads Line x

MILE00 Milenkovic, A. "Achieving High Performance in Bus-Based Shared-Memory Multiprocessors." *IEEE Concurrency*, July-September 2000.

PFIS98 Pfister, G. *In Search of Clusters.* Upper Saddle River, NJ: Prentice Hall, 1998.

STON93 Stone, H. *High-Performance Computer Architecture.* Reading, MA: Addison-Wesley, 1993.

TOMA93 Tomasevic, M., and Milutinovic, V. *The Cache Coherence Problem in Shared-Memory Multiprocessors: Hardware Solutions.* Los Alamitos, CA: IEEE Computer Society Press, 1993.

UNGE02 Ungerer, T.; Rubic, B.; and Silc, J. "Multithreaded Processors." *The Computer Journal*, No. 3, 2002.

UNGE03 Ungerer, T.; Rubic, B.; and Silc, J. " A Survey of Processors with Explicit Multithreading." *ACM Computing Surveys*, March, 2003.

WEYG01 Weygant, P. *Clusters for High Availability.* Upper Saddle River, NJ: Prentice Hall, 2001.

Recommended Web Site:

- **IEEE Computer Society Task Force on Cluster Computing:** An international forum to promote cluster computing research and education.

18.9 KEY TERMS, REVIEW QUESTIONS, AND PROBLEMS

Key Terms

active standby	MESI protocol	symmetric multiprocessor (SMP)
cache coherence	multiprocessor	uniform memory access (UMA)
cluster	nonuniform memory access (NUMA)	uniprocessor
directory protocol	passive standby	vector facility
failback	snoopy protocol	
failover		

Review Questions

18.1 List and briefly define three types of computer system organization.
18.2 What are the chief characteristics of an SMP?
18.3 What are some of the potential advantages of an SMP compared with a uniprocessor?
18.4 What are some of the key OS design issues for an SMP?
18.5 What is the difference between software and hardware cache coherent schemes?
18.6 What is the meaning of each of the four states in the MESI protocol?
18.7 What are some of the key benefits of clustering?
18.8 What is the difference between failover and failback?
18.9 What are the differences among UMA, NUMA, and CC-NUMA?

Most of the instructions in Table 18.3 are self-explanatory. The two summation instructions warrant further explanation. The accumulate operation adds together the elements of a single vector (ACCUMULATE) or the elements of the product of two vectors (MULTIPLY-AND-ACCUMULATE). These instructions present an interesting design problem. We would like to perform this operation as rapidly as possible, taking full advantage of the ALU pipeline. The difficulty is that the sum of two numbers put into the pipeline is not available until several cycles later. Thus, the third element in the vector cannot be added to the sum of the first two elements until those two elements have gone through the entire pipeline. To overcome this problem, the elements of the vector are added in such a way as to produce four partial sums. In particular, elements 0, 4, 8, 12, ..., 124 are added in that order to produce partial sum 0; elements 1, 5, 9, 13, ..., 125 to partial sum 1; elements 2, 6, 10, 14, ..., 126 to partial sum 2; and elements 3, 7, 11, 15, ..., 127 to partial sum 4. Each of these partial sums can proceed through the pipeline at top speed, because the delay in the pipeline is roughly four cycles. A separate vector register is used to hold the partial sums. When all elements of the original vector have been processed, the four partial sums are added together to produce the final result. The performance of this second phase is not critical, because only four vector elements are involved.

18.8 RECOMMENDED READING AND WEB SITE

[CATA94] surveys the principles of multiprocessors and examines SPARC-based SMPs in detail. SMPs are also covered in some detail in [STON93] and [HWAN93].

[MILE00] is an overview of cache coherence algorithms and techniques for multiprocessors, with an emphasis on performance issues. Another survey of the issues relating to cache coherence in multiprocessors is [LILJ93]. [TOMA93] contains reprints of many of the key papers on the subject.

[UNGE02] is an excellent survey of the concepts of multithreaded processors and chip multiprocessors. [UNGE03] is a lengthy survey of both proposed and current multithreaded processors that use explicit multithreading.

[PFIS98] is the book to read for anyone interested in clusters; the book covers the hardware and software design issues and contrasts clusters with SMPs and NUMAs; the book also contains a solid technical description of SMP and NUMA design issues. A thorough treatment of clusters can be found in [BUYY99a] and [BUYY99b]. [WEYG01] is a less technical survey of clusters, with good commentary on various commercial products.

Good discussions of vector computation can be found in [STON93] and [HWAN93].

BUYY99a Buyya, R. *High Performance Cluster Computing: Architectures and Systems.* Upper Saddle River, NJ: Prentice Hall, 1999.

BUYY99b Buyya, R. *High Performance Cluster Computing: Programming and Applications.* Upper Saddle River, NJ: Prentice Hall, 1999.

CATA94 Catanzaro, B. *Multiprocessor System Architectures.* Mountain View, CA: Sunsoft Press, 1994.

HWAN93 Hwang, K. *Advanced Computer Architecture.* New York: McGraw-Hill, 1993.

LILJ93 Lilja, D. "Cache Coherence in Large-Scale Shared-Memory Multiprocessors: Issues and Comparisons." *ACM Computing Surveys,* September 1993.

Table 18.3 18.3 IBM 3090 Vector Facility: Arithmetic and Logical Instructions

Operation	Data Types: Floating-Point Long	Data Types: Floating-Point Short	Data Types: Binary or Logical	Operand Locations			
Add	FL	FS	BI	V + V → V	V + S → V	Q + V → V	Q + S → V
Subtract	FL	FS	BI	V − V → V	V − S → V	Q − V → V	Q − S → V
Multiply	FL	FS	BI	V × V → V	V × S → V	Q × V → V	Q × S → V
Divide	FL	FS	—	V/V → V	V/S → V	Q/V → V	Q/S → V
Compare	FL	FS	BI	V · V → V	V · S → V	Q · V → V	Q · S → V
Multiply and Add	FL	FS	—		V + V × S → V	V + Q × V → V	V + Q × S → V
Multiply and Subtract	FL	FS	—		V − V × S → V	V − Q × V → V	V − Q × S → V
Multiply and Accumulate	FL	FS	—	P + · V → V	P + · S → V		
Complement	FL	FS	BI	−V → V			
Positive Absolute	FL	FS	BI	|V| → V			
Negative Absolute	FL	FS	BI	−|V| → V			
Maximum	FL	FS	—			Q · V → Q	
Maximum Absolute	FL	FS	—			Q · V → Q	
Minimum	FL	FS	—			Q · V → Q	
Shift Left Logical	—	—	LO	· V → V			
Shift Right Logical	—	—	LO	· V → V			
And	—	—	LO	V&V → V	V&S → V	Q&V → V	Q&S → V
OR	—	—	LO	V|V → V	V|S → V	Q|V → V	Q|S → V
Exclusive-OR	—	—	LO	V⊕V → V	V⊕S → V	Q⊕V → V	Q⊕S → V

Explanation:

Data Types
- FL Long Floating point
- FS Short floating point
- BI Binary integer
- LO Logical

Operand Locations
- V Vector register
- S Storage
- Q Scalar (general or floating-point register)
- P Partial sums in vector register
- · Special operation

registers are to be processed for a particular operation. The vector-status register contains control fields, such as the vector count, that determine how many elements in the vector registers are to be processed. The vector-activity count keeps track of the time spent executing vector instructions.

Compound Instructions As was discussed previously, instruction execution can be overlapped using chaining to improve performance. The designers of the IBM vector facility chose not to include this capability for several reasons. The System/370 architecture would have to be extended to handle complex interruptions (including their effect on virtual memory management), and corresponding changes would be needed in the software. A more basic issue was the cost of including the additional controls and register access paths in the vector facility for generalized chaining.

Instead, three operations are provided that combine into one instruction (one opcode) the most common sequences in vector computation, namely multiplication followed by addition, subtraction, or summation. The storage-to-register MULTIPLY-AND-ADD instruction, for example, fetches a vector from storage, multiplies it by a vector from a register, and adds the product to a third vector in a register. By use of the compound instructions MULTIPLY-AND-ADD and MULTIPLY-AND-SUBTRACT in the example of Figure 18.19, the total time for the iteration is reduced from 10 to 8 cycles.

Unlike chaining, compound instructions do not require the use of additional registers for temporary storage of intermediate results, and they require one less register access. For example, consider the following chain:

$$A \longrightarrow VR1$$
$$VR1 + VR2 \longrightarrow VR1$$

In this case, two stores to the vector register VR1 are required. In the IBM architecture there is a storage-to-register ADD instruction. With this instruction, only the sum is placed in VR1. The compound instruction also avoids the need to reflect in the machine-state description the concurrent execution of a number of instructions, which simplifies status saving and restoring by the operating system and the handling of interrupts.

The Instruction Set Table 18.3 summarizes the arithmetic and logical operations that are defined for the vector architecture. In addition, there are memory-to-register load and register-to-memory store instructions. Note that many of the instructions use a three-operand format. Also, many instructions have a number of variants, depending on the location of the operands. A source operand may be a vector register (V), storage (S), or a scalar register (Q). The target is always a vector register, except for comparison, the result of which goes into the vector-mask register. With all these variants, the total number of opcodes (distinct instructions) is 171. This rather large number, however, is not as expensive to implement as might be imagined. Once the machine provides the arithmetic units and the data paths to feed operands from storage, scalar registers, and vector registers to the vector pipelines, the major hardware cost has been incurred. The architecture can, with little difference in cost, provide a rich set of variants on the use of those registers and pipelines.

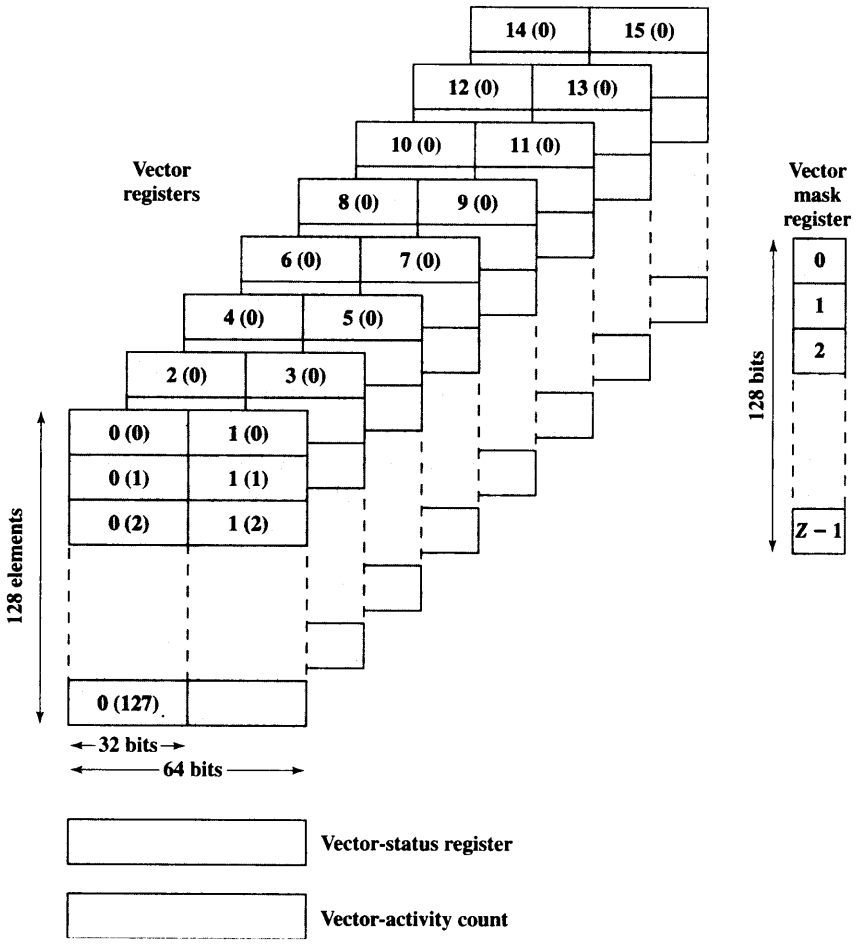

Figure 18.20 Registers for the IBM 3090 Vector Facility

Figure 18.20 illustrates the registers that are part of the IBM 3090 vector facility. There are sixteen 32-bit vector registers. The vector registers can also be coupled to form eight 64-bit vector registers. Any register element can hold an integer or floating-point value. Thus, the vector registers may be used for 32-bit and 64-bit integer values, and 32-bit and 64-bit floating-point values.

The architecture specifies that each register contains from 8 to 512 scalar elements. The choice of actual length involves a design trade-off. The time to do a vector operation consists essentially of the overhead for pipeline startup and register filling plus one cycle per vector element. Thus, the use of a large number of register elements reduces the relative startup time for a computation. However, this efficiency must be balanced against the added time required for saving and restoring vector registers on a process switch and the practical cost and space limits. These considerations led to the use of 128 elements per register in the current 3090 implementation.

Three additional registers are needed by the vector facility. The vector-mask register contains mask bits that may be used to select which elements in the vector