- *IEEE Symposium on Parallel and Distributed Processing* (IEEE Computer Society, since 1989).

Problem exercises are from Kai Hwang's "Advanced Computer Architecture" book.

Exercises

Problem 1.1 A 40-MHz processor was used to execute a benchmark program with the following instruction mix and clock cycle counts:

Instruction type	Instruction count	Clock cycle count
Integer arithmetic	45000	1
Data transfer	32000	2
Floating point	15000	2
Control transfer	8000	2

Determine the effective CPI, MIPS rate, and execution time for this program.

Problem 1.2 Explain how instruction set, compiler technology, CPU implementation and control, and cache and memory hierarchy affect the CPU performance and justify the effects in terms of program length, clock rate, and effective CPI.

Problem 1.3 A workstation uses a 15-MHz processor with a claimed 10-MIPS rating to execute a given program mix. Assume a one-cycle delay for each memory access.

(a) What is the effective CPI of this computer?

(b) Suppose the processor is being upgraded with a 30-MHz clock. However, the speed of the memory subsystem remains unchanged, and consequently two clock cycles are needed per memory access. If 30% of the instructions require one memory access and another 5% require two memory accesses per instruction, what is the performance of the upgraded processor with a compatible instruction set and equal instruction counts in the given program mix?

Problem 1.4 Consider the execution of an object code with 200,000 instructions on a 40-MHz processor. The program consists of four major types of instructions. The instruction mix and the number of cycles (CPI) needed for each instruction type are given below based on the result of a program trace experiment:

Instruction type	CPI	Instruction mix
Arithmetic and logic	1	60%
Load/store with cache hit	2	18%
Branch	4	12%
Memory reference with cache miss	8	10%

(a) Calculate the average CPI when the program is executed on a uniprocessor with the above trace results.
(b) Calculate the corresponding MIPS rate based on the CPI obtained in part (a).

Problem 1.5 Indicate whether each of the following statements is *true* or *false* and justify your answer with reasoning and supportive or counter examples:

(a) The CPU computations and I/O operations cannot be overlapped in a multiprogrammed computer.
(b) Synchronization of all PEs in an SIMD computer is done by hardware rather than by software as is often done in most MIMD computers.
(c) As far as programmability is concerned, shared-memory multiprocessors offer simpler interprocessor communication support than that offered by a message-passing multicomputer.
(d) In an MIMD computer, all processors must execute the same instruction at the same time synchronously.
(e) As far as scalability is concerned, multicomputers with distributed memory are more scalable than shared-memory multiprocessors.

Problem 1.6 The execution times (in seconds) of four programs on three computers are given below:

Program	Execution Time (in seconds)		
	Computer A	Computer B	Computer C
Program 1	1	10	20
Program 2	1000	100	20
Program 3	500	1000	50
Program 4	100	800	100

Assume that 100,000,000 instructions were executed in each of the four programs. Calculate the MIPS rating of each program on each of the three machines. Based on these ratings, can you draw a clear conclusion regarding the relative performance of the three computers? Give reasons if you find a way to rank them statistically.

Problem 1.7 Characterize the architectural operations of SIMD and MIMD computers. Distinguish between multiprocessors and multicomputers based on their structures, resource sharing, and interprocessor communications. Also, explain the differences among UMA, NUMA, and COMA, and NORMA computers.

Problem 1.8 The following code segment, consisting of six instructions, needs to be executed 64 times for the evaluation of vector arithmetic expression: $D(I) = A(I) + B(I) \times C(I)$ for $0 \leq I \leq 63$.

Load R1, B(I) /R1 ← Memory $(\alpha + I)$/

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)/
Store D(I), R3	/Memory (θ + I) ← (R3)/

where R1, R2, and R3 are CPU registers, (R1) is the content of R1, α, β, γ, and θ are the starting 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 Multiply on either a uniprocessor or a single PE in an SIMD machine.

(a) Calculate the total number of CPU cycles needed to execute the above code segment repeatedly 64 times on an SISD uniprocessor computer sequentially, ignoring all other time delays.

(b) Consider the use of an SIMD computer with 64 PEs to execute the above 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?

Problem 1.9 Prove that the best parallel algorithm written for an n-processor EREW-PRAM model can be no more than $O(\log n)$ times slower than any algorithm for a CRCW model of PRAM having the same number of processors.

Problem 1.10 Consider the multiplication of two n-bit binary integers using a 1.2-μm CMOS multiplier chip. Prove the lower bound $AT^2 > kn^2$, where A is the chip area, T is the execution time, n is the word length, and k is a technology-dependent constant.

Problem 1.11 Compare the PRAM models with physical models of real parallel computers in each of the following categories:

(a) Which PRAM variant can best model SIMD machines and how?

(b) Repeat the question in part (a) for shared-memory MIMD machines.

Problem 1.12 Answer the following questions related to the architectural development tracks presented in Section 1.5:

(a) For the shared-memory track (Fig. 1.17), explain the trend in physical memory organizations from the earlier system (C.mmp) to more recent systems (such as Dash, etc.).

(b) Distinguish between medium-grain and fine-grain multicomputers in their architectures and programming requirements.

(c) Distinguish between register-to-register and memory-to-memory architectures for building conventional multivector supercomputers.

(d) Distinguish between single-threaded and multithreaded processor architectures.

Exercises

Problem 2.1 Define the following terms related to parallelism and dependence relations:

(a) Computational granularity.
(b) Communication latency.
(c) Flow dependence.
(d) Antidependence.
(e) Output dependence.
(f) I/O dependence.
(g) Control dependence.
(h) Resource dependence.
(i) Bernstein conditions.
(j) Degree of parallelism.

Problem 2.2 Define the following terms for various system interconnect architectures:

(a) Node degree.
(b) Network diameter.
(c) Bisection bandwidth.
(d) Static connection networks.
(e) Dynamic connection networks.
(f) Nonblocking networks.
(g) Multicast and broadcast.
(h) Mesh versus torus.
(i) Symmetry in networks.
(j) Multistage networks.
(k) Crossbar networks.
(l) Digital buses.

Problem 2.3 Answer the following questions on program flow mechanisms and computer models:

(a) Compare control-flow, dataflow, and reduction computers in terms of the program flow mechanism used.
(b) Comment on the advantages and disadvantages in control complexity, potential for parallelism, and cost-effectiveness of the above computer models.
(c) What are the differences between string reduction and graph reduction machines?

Problem 2.4 Perform a data dependence analysis on each of the following Fortran program fragments. Show the dependence graphs among the statements with justification.

(a)

```
S1:   A = B + D
S2:   C = A × 3
S3:   A = A + C
S4:   E = A / 2
```

(b)

```
S1:   X = SIN(Y)
S2:   Z = X + W
S3:   Y = −2.5 × W
S4:   X = COS(Z)
```

(c) Determine the data dependences in the same and adjacent iterations of the following Do-loop.

```
                Do 10 I = 1, N
S1:                 A(I+1) = B(I−1) + C(I)
S2:                 B(I) = A(I) × K
S3:                 C(I) = B(I) − 1
10              Continue
```

Problem 2.5 Analyze the data dependences among the following statements in a given program:

S1:	Load R1, 1024	/R1 ← 1024/
S2:	Load R2, M(10)	/R2 ← Memory(10) /
S3:	Add R1, R2	/R1 ← (R1) + (R2)/
S4:	Store M(1024), R1	/Memory(1024) ← (R1)/
S5:	Store M((R2)), 1024	/Memory(64) ← 1024/

where (Ri) means the content of register Ri and Memory(10) contains 64 initially.

(a) Draw a dependence graph to show all the dependences.
(b) Are there any resource dependences if only one copy of each functional unit is available in the CPU?
(c) Repeat the above for the following program statements:

S1:	Load R1, M(100)	/R1 ← Memory(100)/
S2:	Move R2, R1	/R2 ← (R1) /
S3:	Inc R1	/R1 ← (R1) + 1/
S4:	Add R2, R1	/R2 ← (R2) + (R1)/
S5:	Store M(100), R1	/Memory(100) ← (R1)/

Problem 2.6 A sequential program consists of the following five statements, S1 through S5. Considering each statement as a separate process, clearly identify *input set* I_i and *output set* O_i of each process. Restructure the program using Bernstein's conditions in order to achieve maximum parallelism between processes. If any pair of processes cannot be executed concurrently, specify which of the three conditions is not satisfied.

```
S1:   A = B + C
S2:   C = B × D
S3:   S = 0
S4:   Do I = A, 100
          S = S + X(I)
      End Do
S5:   IF (S .GT. 1000) C = C × 2
```

Problem 2.7 Consider the execution of the following code segment consisting of seven statements. Use Bernstein's conditions to detect the maximum parallelism embedded in this code. Justify the portions that can be executed in parallel and the remaining portions that must be executed sequentially. Rewrite the code using parallel constructs such as *Cobegin* and *Coend*. No variable substitution is allowed. All statements can be executed in parallel if they are declared within the same block of a (*Cobegin*, *Coend*) pair.

S1: $A = B + C$
S2: $C = D + E$
S3: $F = G + E$
S4: $C = A + F$
S5: $M = G + C$
S6: $A = L + C$
S7: $A = E + A$

Problem 2.8 According to program order, the following six arithmetic expressions need to be executed in minimum time. Assume that all are integer operands already loaded into working registers. No memory reference is needed for the operand fetch. Also, all intermediate or final results are written back to working registers without conflicts.

P1: $X \leftarrow (A + B) \times (A - B)$
P2: $Y \leftarrow (C + D)/(C - D)$
P3: $Z \leftarrow X + Y$
P4: $A \leftarrow E \times F$
P5: $Y \leftarrow E - Z$
P6: $B \leftarrow (X - F) \times A$

(a) Use the minimum number of working registers to rewrite the above HLL program into a minimum-length assembly language code using arithmetic opcodes *add*, *subtract*, *multiply*, and *divide* exclusively. Assume a fixed instruction format with three register fields: two for sources and one for destinations.
(b) Perform a flow analysis of the assembly code obtained in part (a) to reveal all data dependences with a dependence graph.
(c) The CPU is assumed to have two *add units*, one *multiply unit*, and one *divide unit*. Work out an optimal schedule to execute the assembly code in minimum time, assuming 1 cycle for the add unit, 3 cycles for the multiply unit, and 18 cycles for the divide unit to complete the execution of one instruction. Ignore all overhead caused by instruction fetch, decode, and writeback. No pipelining is assumed here.

Problem 2.9 Given the following assembly language code. Exploit the maximum degree of parallelism among the 16 instructions, assuming no resource conflicts and

multiple functional units are available simultaneously. For simplicity, no pipelining is assumed. All instructions take one machine cycle to execute. Ignore all other overhead.

1:	Load R1, A	/R1 ← Mem(A)/
2:	Load R2, B	/R2 ← Mem(B)/
3:	Mul R3, R1, R2	/R3 ← (R1) × (R2)/
4:	Load R4, D	/R4 ← Mem(D)/
5:	Mul R5, R1, R4	/R5 ← (R1) × (R4)/
6:	Add R6, R3, R5	/R6 ← (R3) + (R5)/
7:	Store X, R6	/Mem(X) ← (R6)/
8:	Load R7, C	/R7 ← Mem(C)/
9:	Mul R8, R7, R4	/R8 ← (R7) × (R4)/
10:	Load R9, E	/R9 ← Mem(E)/
11:	Add R10, R8, R9	/R10 ← (R8) + (R9)/
12:	Store Y, R10	/Mem(Y) ← (R10)/
13:	Add R11, R6, R10	/R11 ← (R6)+(R10)/
14:	Store U, R11	/Mem(U) ← (R11)/
15:	Sub R12, R6, R10	/R12 ← (R6) − (R10)/
16:	Store V, R12	/Mem(V) ← (R12)/

(a) Draw a program graph with 16 nodes to show the flow relationships among the 16 instructions.

(b) Consider the use of a three-issue superscalar processor to execute this program fragment in minimum time. The processor can issue one memory-access instruction (Load or Store but not both), one Add/Sub instruction, and one Mul (multiply) instruction per cycle. The Add unit, Load/Store unit, and Multiply unit can be used simultaneously if there is no data dependence.

Problem 2.10 Repeat part (b) of Problem 2.9 on a dual-processor system with shared memory. Assume that the same superscalar processors are used and that all instructions take one cycle to execute.

(a) Partition the given program into two balanced halves. You may want to insert some load or store instructions to pass intermediate results generated by the two processors to each other. Show the divided program flow graph with the final output U and V generated by the two processors separately.

(b) Work out an optimal schedule for parallel execution of the above divided program by the two processors in minimum time.

Problem 2.11 You are asked to design a direct network for a multicomputer with 64 nodes using a three-dimensional torus, a six-dimensional binary hypercube, and cube-connected-cycles (CCC) with a minimum diameter. The following questions are related to the relative merits of these network topologies:

(a) Let d be the node degree, D the network diameter, and l the total number of links in a network. Suppose the *quality* of a network is measured by $(d \times D \times l)^{-1}$. Rank

Problem 4.11 Consider a two-level memory hierarchy, M_1 and M_2. Denote the hit ratio of M_1 as h. Let c_1 and c_2 be the costs per kilobyte, s_1 and s_2 the memory capacities, and t_1 and t_2 the access times, respectively.

(a) Under what conditions will the average cost of the entire memory system approach c_2?
(b) What is the effective memory-access time t_a of this hierarchy?
(c) Let $r = t_2/t_1$ be the speed ratio of the two memories. Let $E = t_1/t_a$ be the *access efficiency* of the memory system. Express E in terms of r and h.
(d) Plot E against h for $r = 5, 20$, and 100, respectively, on grid paper.
(e) What is the required hit ratio h to make $E > 0.95$ if $r = 100$?

Problem 4.12 You are asked to perform capacity planning for a two-level memory system. The first level, M_1, is a cache with three capacity choices of 64 Kbytes, 128 Kbytes, and 256 Kbytes. The second level, M_2, is a main memory with a 4-Mbyte capacity. Let c_1 and c_2 be the costs per byte and t_1 and t_2 the access times for M_1 and M_2, respectively. Assume $c_1 = 20c_2$ and $t_2 = 10t_1$. The cache hit ratios for the three capacities are assumed to be 0.7, 0.9, and 0.98, respectively.

(a) What is the average access time t_a in terms of $t_1 = 20$ ns in the three cache designs? (Note that t_1 is the time from CPU to M_1 and t_2 is that from CPU to M_2, not from M_1 to M_2).
(b) Express the average byte cost of the entire memory hierarchy if $c_2 = \$0.2$/Kbyte.
(c) Compare the three memory designs and indicate the order of merit in terms of average costs and average access times, respectively. Choose the optimal design based on the product of average cost and average access time.

Problem 4.13 Compare the advantages and shortcomings in implementing private virtual memories and a globally shared virtual memory in a multicomputer system. This comparative study should consider the latency, coherence, page migration, protection, implementation, and application problems in the context of building a scalable multicomputer system with distributed shared memories.

Problem 4.14 Explain the *inclusion property* and *memory coherence* requirements in a multilevel memory hierarchy. Distinguish between *write-through* and *write-back* policies in maintaining the coherence in adjacent levels. Also explain the basic concepts of *paging* and *segmentation* in managing the physical and virtual memories in a hierarchy.

Problem 4.15 A two-level memory system has eight virtual pages on a disk to be mapped into four page frames (PFs) in the main memory. A certain program generated the following page trace:

$$1, 0, 2, 2, 1, 7, 6, 7, 0, 1, 2, 0, 3, 0, 4, 5, 1, 5, 2, 4, 5, 6, 7, 6, 7, 2, 4, 2, 7, 3, 3, 2, 3$$

(a) Show the successive virtual pages residing in the four page frames with respect to

the above page trace using the LRU replacement policy. Compute the hit ratio in the main memory. Assume the PFs are initially empty.

(b) Repeat part (a) for the circular FIFO page replacement policy. Compute the hit ratio in the main memory.

(c) Compare the hit ratio in parts (a) and (b) and comment on the effectiveness of using the circular FIFO policy to approximate the LRU policy with respect to this particular page trace.

Problem 4.16

(a) Explain the temporal locality, spatial locality, and sequential locality associated with program/data access in a memory hierarchy.

(b) What is the working set? Comment on the sensitivity of the observation window size to the size of the working set. How will this affect the main memory hit ratio?

(c) What is the 90-10 rule and its relationship to the locality of references?

Problem 4.17 Consider a two-level memory hierarchy, M_1 and M_2, with access times t_1 and t_2, costs per byte c_1 and c_2, and capacities s_1 and s_2, respectively. The cache hit ratio $h_1 = 0.95$ at the first level. (Note that t_2 is the access time between the CPU and M_2, not between M_1 and M_2).

(a) Derive a formula showing the effective access time t_{eff} of this memory system.

(b) Derive a formula showing the total cost of this memory system.

(c) Suppose $t_1 = 20$ ns, t_2 is unknown, $s_1 = 512$ Kbytes, s_2 is unknown, $c_1 =$ \$0.01/byte, and $c_2 =$ \$0.0005/byte. The total cost of the cache and main memory is upper-bounded by \$15,000.

(i) How large a capacity of M_2 ($s_2 =?$) can you acquire without exceeding the budget limit?

(ii) How fast a main memory ($t_2 =?$) do you need to achieve an effective access time of $t_{eff} = 40$ ns in the entire memory system under the above hit ratio assumptions?

Problem 4.18 Distinguish between numeric processing and symbolic processing computers in terms of data objects, common operations, memory requirements, communication patterns, algorithmic properties, I/O requirements, and processor architectures.

Let $R(t)$ be the *resident set* of all pages residing in main memory at time t. Let $q(t)$ be the page to be replaced from $R(t)$ when a page fault occurs at time t.

Page Replacement Policies The following page replacement policies are specified in a demand paging memory system for a page fault at time t.

(1) *Least recently used* (LRU) — This policy replaces the page in $R(t)$ which has the longest backward distance:

$$q(t) = y, \quad \text{iff} \quad b_t(y) = \max_{x \in R(t)} \{b_t(x)\} \tag{4.13}$$

(2) *Optimal (*OPT*) algorithm* — This policy replaces the page in $R(t)$ with the longest forward distance:

$$q(t) = y, \quad \text{iff} \quad f_t(y) = \max_{x \in R(t)} \{f_t(x)\} \tag{4.14}$$

(3) *First-in-first-out* (FIFO) — This policy replaces the page in $R(t)$ which has been in memory for the longest time.

(4) *Least frequently used* (LFU) — This policy replaces the page in $R(t)$ which has been least referenced in the past.

(5) *Circular FIFO* — This policy joins all the page frame entries into a circular FIFO queue using a pointer to indicate the front of the queue. An *allocation bit* is associated with each page frame. This bit is set upon initial allocation of a page to the frame.

When a page fault occurs, the queue is circularly scanned from the pointer position. The pointer skips and resets the allocated page frames and replaces the very first unallocated page frame. When all frames are allocated, the front of the queue is replaced, as in the FIFO policy.

(6) *Random replacement* — This is a trivial algorithm which chooses any page for replacement randomly.

Example 4.9 Page tracing experiments and interpretation of results

Consider a paged virtual memory system with a two-level hierarchy: main memory M_1 and disk memory M_2. For clarity of illustration, assume a page size of four words. The number of page frames in M_1 is 3, labeled a, b and c; and the number of pages in M_2 is 10, identified by $0, 1, 2, ..., 9$. The ith page in M_2 consists of word addresses $4i$ to $4i + 3$ for all $i = 0, 1, 2, ..., 9$.

A certain program generates the following sequence of word addresses which are grouped (underlined) together if they belong to the same page. The sequence of page numbers so formed is the *page trace*:

Word trace:	0,1,2,3,	4,5,6,7,	8,	16,17,	9,10,11,	12,	28,29,30,	8,9,10,	4,5,	12,	4,5
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Page trace:	0	1	2	4	2	3	7	2	1	3	1

Page tracing experiments are described below for three page replacement policies: LRU, OPT, and FIFO, respectively. The successive pages loaded in the page frames (PFs) form the trace entries. Initially, all PFs are empty.

	PF	0	1	2	4	2	3	7	2	1	3	1	Hit Ratio
LRU	*a*	0	0	0	4	4	4	7	7	7	3	3	$\frac{3}{11}$
	b		1	1	1	1	3	3	3	1	1	1	
	c			2	2	2	2	2	2	2	2	2	
	Faults	*	*	*	*		*	*		*	*		
OPT	*a*	0	0	0	4	4	3	7	7	7	3	3	$\frac{4}{11}$
	b		1	1	1	1	1	1	1	1	1	1	
	c			2	2	2	2	2	2	2	2	2	
	Faults	*	*	*	*		*	*			*		
FIFO	*a*	0	0	0	4	4	4	4	2	2	2	2	$\frac{2}{11}$
	b		1	1	1	1	3	3	3	1	1	1	
	c			2	2	2	2	7	7	7	3	3	
	Faults	*	*	*	*		*	*	*	*	*		

The above results indicate the superiority of the OPT policy over the others. However, the OPT cannot be implemented in practice. The LRU policy performs better than the FIFO due to the locality of references. From these results, we realize that the LRU is generally better than the FIFO. However, exceptions still exist due to the dependence on program behavior. ■

Relative Performance The performance of a page replacement algorithm depends on the page trace (program behavior) encountered. The best policy is the OPT algorithm. However, the OPT replacement is not realizable because no one can predict the future page demand in a program.

The LRU algorithm is a popular policy and often results in a high hit ratio. The FIFO and random policies may perform badly because of violation of the program locality.

The circular FIFO policy attempts to approximate the LRU with a simple circular queue implementation. The LFU policy may perform between the LRU and the FIFO policies. However, there is no fixed superiority of any policy over the others because of the dependence on program behavior and run-time status of the page frames.

In general, the page fault rate is a monotonic decreasing function of the size of the resident set $R(t)$ at time t because more resident pages result in a higher hit ratio in the main memory.

Block Replacement Policies The relationship between the cache block frames and cache blocks is similar to that between page frames and pages on a disk. Therefore, those page replacement policies can be modified for *block replacement* when a cache miss occurs.

Add, *subtract*, *multiply*, and *divide* are the four primitive arithmetic operations. For fixed-point numbers, the add or subtract of two n-bit integers (or fractions) produces an n-bit result with at most one carry-out.

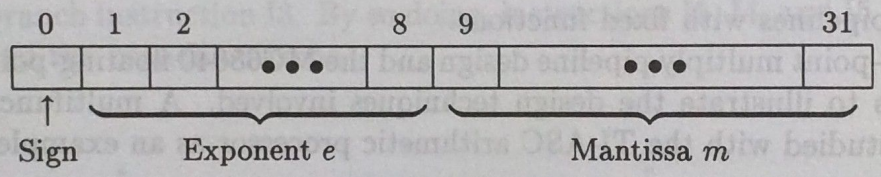
The multiplication of two n-bit numbers produces a $2n$-bit result which may require use of two memory words or two registers to hold the full-precision result.

The division of an n-bit number by another may create an arbitrarily long quotient and a remainder. Only an approximate result is expected in fixed-point division with rounding or truncation. However, one can always expand the precision by using a $2n$-bit dividend and a $2n$-bit divisor to yield a $2n$-bit quotient.

Floating-Point Numbers A floating-point number X is represented by a pair (m, e), where m is the *mantissa* (or *fraction*) and e is the *exponent* with an implied *base* (or *radix*). The algebraic value is represented as $X = m \times r^e$. The sign of X can be embedded in the mantissa.

Example 6.9 The IEEE 754 floating-point standard

A 32-bit floating-point number is specified in the IEEE 754 Standard as follows:



A binary base is assumed with $r = 2$. The 8-bit exponent e field uses an *excess-127* code. The dynamic range of e is $(-127, 128)$, internally represented as $(0, 255)$. The sign s and the 23-bit mantissa field m form a 25-bit sign-magnitude fraction, including an implicit or "hidden" 1 bit to the left of the binary point. Thus the complete mantissa actually represents the value $1.m$.

This hidden bit is not stored with the number. If $0 < e < 255$, then a nonzero normalized number represents the following algebraic value:

$$X = (-1)^s \times 2^{e-127} \times (1.m) \tag{6.15}$$

When $e = 255$ and $m \neq 0$, a *not-a-number* (NaN) is represented. NaNs can be caused by dividing a zero by a zero or taking the square root of a negative number, among many other nondeterminate cases. When $e = 255$ and $m = 0$, an infinite number $X = (-1)^s \infty$ is represented. Note that $+\infty$ and $-\infty$ are represented differently.

When $e = 0$ and $m \neq 0$, the number represented is $X = (-1)^s 2^{-126}(0.m)$. When $e = 0$ and $m = 0$, a zero is represented as $X = (-1)^s 0$. Again, $+0$ and -0 are possible.

The 64-bit (double-precision) floating point can be defined similarly using an excess-1023 code in the exponent field and a 52-bit mantissa field. A number which is nonzero, finite, non-NaN, and normalized, has the following value:

$$X = (-1)^s \times 2^{e-1023} \times (1.m) \tag{6.16}$$

Special rules are given in the standard to handle overflow or underflow conditions. Interested readers may check the published IEEE standards for details. ■

Floating-Point Operations The four primitive arithmetic operations are defined below for a pair of floating-point numbers represented by $X = (m_x, e_x)$ and $Y = (m_y, e_y)$. For clarity, we assume $e_x \leq e_y$ and base $r = 2$.

$$X + Y = (m_x \times 2^{e_x - e_y} + m_y) \times x^{e_y} \tag{6.17}$$

$$X - Y = (m_x \times 2^{e_x - e_y} - m_y) \times x^{e_y} \tag{6.18}$$

$$X \times Y = (m_x \times m_y) \times 2^{e_x + e_y} \tag{6.19}$$

$$X \div Y = (m_x \div m_y) \times 2^{e_x - e_y} \tag{6.20}$$

The above equations clearly identify the number of arithmetic operations involved in each floating-point function. These operations can be divided into two halves: One half is for exponent operations such as comparing their relative magnitudes or adding/subtracting them; the other half is for mantissa operations, including four types of fixed-point operations.

Floating-point units are ideal for pipelined implementation. The two halves of the operations demand almost twice as much hardware as that required in a fixed-point unit. Arithmetic shifting operations are needed for equalizing the two exponents before their mantissas can be added or subtracted.

Shifting a binary fraction m to the right k places corresponds to the weighting $m \times 2^{-k}$, and shifting k places to the left corresponds to $m \times 2^k$. In addition, normalization of a floating-point number also requires left shifts to be performed.

Elementary Functions Elementary functions include trigonometric, exponential, logarithmic, and other transcendental functions. Truncated polynomials or power series can be used to evaluate the elementary functions, such as $\sin x$, $\ln x$, e^x, $\cosh x$, $\tan^{-1} y$, $\sqrt{x}$, x^3, etc.

Some CORDIC (coordinate rotation digital computer) algorithms have been developed to calculate trigonometric functions and to convert between binary and mixed radix number systems. Interested readers may refer to the book by Hwang (1979) for details of computer arithmetic functions and their hardware implementation.

It should be noted that computer arithmetic can be implemented by hardwired random logic circuitry as well as by table lookup using ROMs or RAMs in memory. Frequently used constants and special function values can be easily generated by table lookup. Hashing can provide fast access to these tables.

6.4.2 Static Arithmetic Pipelines

Most of today's arithmetic pipelines are designed to perform fixed functions. These *arithmetic/logic units* (ALUs) perform fixed-point and floating-point operations sepa-

(d) Suppose nonatomic memory access is allowed in the above multiprocessor. For example, an invalidation does not reach all private caches at the same time. Prove that 011001 is possible even if all instructions were executed in program order but other processors did not observe them in program order.

Problem 5.8 The main memory of a computer is organized as 64 blocks, with a block size of eight words. The cache has eight block frames. In parts (a) through (d), show the mappings from the numbered blocks in main memory to the block frames in the cache. Draw all lines showing the mappings as clearly as possible.

(a) Show the direct mapping and the address bits that identify the tag field, the block number, and the word number.
(b) Show the fully associative mapping and the address bits that identify the tag field and the word number.
(c) Show the two-way set-associative mapping and the address bits that identify the tag field, the set number, and the word number.
(d) Show the sector mapping with four blocks per sector and the address bits that identify the sector number, the block number, and the word number.

Problem 5.9 Consider a cache (M_1) and memory (M_2) hierarchy with the following characteristics:

M_1: 16K words, 50 ns access time
M_2: 1M words, 400 ns access time

Assume eight-word cache blocks and a set size of 256 words with set-associative mapping.

(a) Show the mapping between M_2 and M_1.
(b) Calculate the effective memory-access time with a cache hit ratio of $h = 0.95$.

Problem 5.10 Consider a main memory consisting of four memory modules with 256 words per module. Assume 16 words in each cache block. The cache has a total capacity of 256 words. Set-associative mapping is used to allocate cache blocks to block frames. The cache is divided into four sets.

(a) Show the address assignment for all 1024 words in a four-way low-order interleaved organization of the main memory.
(b) How many blocks are there in the main memory? How many block frames are there in the cache?
(c) Explain the bit fields needed for addressing each word in the two-level memory system.
(d) Show the mapping from the blocks in the main memory to the sets in the cache and explain how to use the tag field to locate a block frame within each set.

Problem 5.11

(a) A uniprocessor system uses separate instruction and data caches with hit ratios h_i and h_d, respectively. The access time from the processor to either cache is c clock cycles, and the block transfer time between the caches and main memory is b clock cycles.

Among all memory references made by the CPU, f_i is the percentage of references to instructions. Among blocks replaced in the data cache, f_{dir} is the percentage of *dirty* blocks. (*Dirty* means that the cache copy is different from the memory copy.)

Assuming a write-back policy, determine the effective memory-access time in terms of h_i, h_d, c, b, f_i, and f_{dir} for this memory system.

(b) The processor memory system described in part (a) is used to construct a bus-based shared-memory multiprocessor. Assume that the hit ratio and access times remain the same as in part (a). However, the effective memory-access time will be different because every processor must now handle cache invalidation in addition to reads and writes.

Let f_{inv} be the fraction of data references that cause invalidation signals to be sent to other caches. The processor sending the invalidation signal requires i clock cycles to complete the invalidation operation. Other processors are not involved in the invalidation process. Assuming a write-back policy again, determine the effective memory-access time for this multiprocessor.

Problem 5.12 A computer system has a 128-byte cache. It uses four-way set-associative mapping with 8 bytes in each block. The physical address size is 32 bits, and the smallest addressable unit is 1 byte.

(a) Draw a diagram showing the organization of the cache and indicating how physical addresses are related to cache addresses.

(b) To what block frames of the cache can the address $000010AF_{16}$ be assigned?

(c) If the addresses $000010AF_{16}$ and $FFFF7Axy_{16}$ can be simultaneously assigned to the same cache set, what values can the address digits x and y have?

Problem 5.13 Consider a shared-memory multiprocessor system with p processors. Let m be the average number of global memory references per instruction execution on a typical processor.

Let t be the average access time to the shared memory and x be the MIPS rate of a uniprocessor using local memory. Consider the execution of n instructions on each processor of the multiprocessor.

(a) Determine the effective MIPS rate of the multiprocessor in terms of the parameters m, t, x, n, and p.

(b) Suppose a multiprocessor has $p = 32$ RISC processors, $m = 0.4$, and $t = 1\ \mu s$. What is the MIPS rate of each processor (i.e., $x = ?$) needed to achieve a multiprocessor performance of 56 MIPS effectively?

(c) Suppose $p = 32$ CISC processors with $x = 2$ MIPS each are used in the above multiprocessor system with $m = 1.6$ and $t = 1$ μs. What will be the effective MIPS rate?

Problem 5.14 Consider a RISC-based shared-memory multiprocessor with p processors, each having an off-chip instruction cache and data cache. The peak performance rating of each processor (assuming a 100% hit ratio in both caches) is x MIPS. You are required to derive a performance formula, taking into account cache misses, shared-memory accesses, and synchronization overhead.

Assume that on the average α percent of the instructions executed are for synchronization purpose, and the penalty for each synchronization operation is an additional t_s μs. The number of memory accesses per instruction is m. Among all memory references made by the CPU, f_i is the percentage of references to instructions. Assume that the instruction cache and data cache have hit ratios h_i and h_d, respectively, after a long period of program tracing on the machine. On cache misses, instructions and data are accessed from the shared memory with an average access time t_m μs.

(a) Derive an expression for approximating the effective MIPS rate of this multiprocessor in terms of $p, x, m, f_i, h_i, h_d, t_m, \alpha$, and t_s. Note that f_i, h_i, h_d, and α are all fractions and t_m and t_s are measured in μs. Ignore the cache-access time and other system overheads in your derivation.

(b) Suppose $m = 0.4, f_i = 0.5, h_i = 0.95, h_d = 0.7, \alpha = 0.05, x = 5, t_m = 0.5$ μs, and $t_s = 5$ μs. Determine the minimum number of processors needed in the above multiprocessor system in order to achieve an effective MIPS rate of 25.

(c) Suppose the total cost of all the caches and shared-memory is upper-bounded by \$25,000. The cache memory costs \$4.70/Kbyte, and the shared memory costs \$0.4/Kbyte. With $p = 16$ processors, each having an instruction cache of capacity $S_i = 32$ Kbytes and a data cache of capacity $S_d = 64$ Kbytes, what is the maximum shared-memory capacity C_m (in Mbytes) that can be acquired within the budget limit?

Problem 5.15 Consider the following three interleaved memory designs for a main memory system with 16 memory modules. Each module is assumed to have a capacity of 1 Mbyte. The machine is byte-addressable.

Design 1: 16-way interleaving with one memory bank.
Design 2: 8-way interleaving with two memory banks.
Design 3: 4-way interleaving with four memory banks.

(a) Specify the address formats for each of the above memory organizations.

(b) Determine the maximum memory bandwidth obtained if only one memory module fails in each of the above memory organizations.

(c) Comment on the relative merits of the three interleaved memory organizations.

Problem 5.16 Consider a memory system for the Cray 1 computer. There are $m = 16$

Until compiler technology is improved to increase the ILP, or multiphase clocking technology is further advanced, we cannot expect a significant increase in (m, n). Reasonable ranges are given in Fig. 6.34 for $1 \leq m \leq 4$ and $1 \leq n \leq 6$. When data dependence and branch penalty are considered, the ideal speedup curves will drop in all cases.

6.6 Bibliographic Notes and Exercises

A classic reference on pipeline architecture is [Kogge81]. [Smith89] surveyed instruction scheduling techniques. Branch strategies and performance models for pipeline processors are treated in [Lee84] and Smith and summarized in a research monograph by [Cragon92a]. Also, a book on pipeline processors is in preparation by [Cragon92b]. Pipeline scheduling theory started with the pioneer work reported in [Davidson71], [Davidson75] et al., [Shar72], and [Patel78]. [Sohi90] studied multiple-functional-unit pipelined processors. [Larson73] studied the optimal number of pipeline stages.

Superscalar and superpipelining techniques were treated in [Jouppi89] and Wall. Michael Johnson's book [Johnson91] also provides valuable insights on superscalar processor design.

Pipeline chaining and networking techniques were investigated by [Hwang88] and Xu. The paper by [Emma87] and Davidson described superpipeline clocking requirements. Compiler techniques for pipeline architecture were reported in [Gross83].

A survey of techniques for reducing branch penalties can be found in [Lilja88]. Instruction issue logic was studied by [Weiss84] and Smith. Dynamic instruction scheduling was pioneered by [Tomasulo67]. Recently, [Lam88] studied software pipelining techniques for VLIW architecture. [Acosta86], Kjelstrup, and Torng have examined the multiple-instruction issuing problem.

Computer arithmetic was treated in [Hwang78]. The CSA tree was reported in [Wallace64]. The details of the IBM 360/91 were reported in [Anderson67]. The TI-ASC was discussed by [Cragon89] and Watson. The CDC 6600 was reported in [Thornton70].

The IEEE floating-point standard can be ordered from [IEEE85]. MIPS R4000 was described in [Mirapuri92]. The DEC Alpha was described in [DEC92]. Information on the Motorola MC88110 can be found in [Diefendorff92] and Allen.

Exercises

Problem 6.1 Consider the execution of a program of 15,000 instructions by a linear pipeline processor with a clock rate of 25 MHz. Assume that the instruction pipeline has five stages and that one instruction is issued per clock cycle. The penalties due to branch instructions and out-of-sequence executions are ignored.

(a) Calculate the speedup factor in using this pipeline to execute the program as compared with the use of an equivalent nonpipelined processor with an equal amount of flow-through delay.

(b) What are the efficiency and throughput of this pipelined processor?

Problem 6.2 Study the DEC Alpha architecture in Example 6.13, find more information in the DEC Alpha handbook, and then answer the following questions with reasoning:

(a) Analyze the scalability of the Alpha processor implementation in terms of superscalar degree and superpipeline degree.
(b) Analyze the scalability of an Alpha-based multiprocessor system in terms of address space and multiprocessor support.

Problem 6.3 Study Section 6.5 and the paper by Jouppi and Wall (1989) and answer the following questions:

(a) Explain why a superpipelined processor performs less effectively than a superscalar processor of small degree.
(b) What is the duality of latency and parallel instruction issue under ideal circumstances?

Problem 6.4 Find the optimal number of pipeline stages k_0 given in Eq. 6.7 using the performance/cost ratio (PCR) given in Eq. 6.6.

Problem 6.5 Prove the lower bound and upper bound on the minimal average latency (MAL) specified in page 277.

Problem 6.6 Consider the following reservation table for a four-stage pipeline with a clock cycle $\tau = 20$ ns.

	1	2	3	4	5	6
S1	X					X
S2		X		X		
S3			X			
S4				X	X	

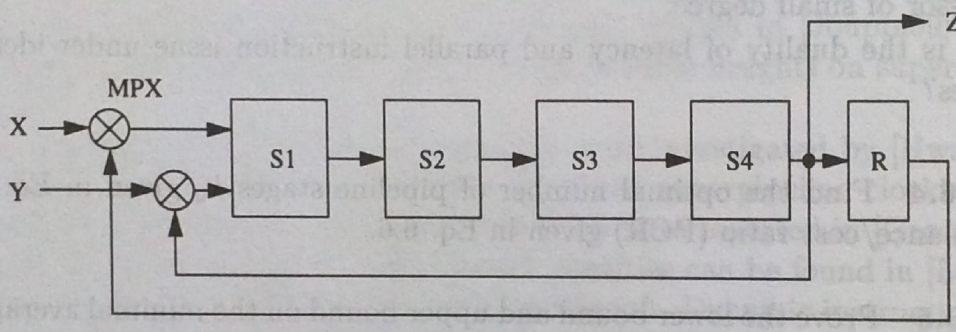
(a) What are the forbidden latencies and the initial collision vector?
(b) Draw the state transition diagram for scheduling the pipeline.
(c) Determine the MAL associated with the shortest greedy cycle.
(d) Determine the pipeline throughput corresponding to the MAL and given τ.
(e) Determine the lower bound on the MAL for this pipeline. Have you obtained the optimal latency from the above state diagram?

Problem 6.7 You are allowed to insert one noncompute delay stage into the pipeline in Problem 6.6 to make a latency of 1 permissible in the shortest greedy cycle. The

purpose is to yield a new reservation table leading to an optimal latency equal to the lower bound.

(a) Show the modified reservation table with five rows and seven columns.
(b) Draw the new state transition diagram for obtaining the optimal cycle.
(c) List all the simple cycles and greedy cycles from the state diagram.
(d) Prove that the new MAL equals the lower bound.
(e) What is the optimal throughput of this pipeline? Indicate the percentage of throughput improvement compared with that obtained in part (d) of Problem 6.6.

Problem 6.8 Consider an adder pipeline with four stages as shown below. The pipeline consists of input lines X and Y and output line Z. The pipeline has a register R at its output where the temporary result can be stored and fed back to S1 at a later point in time. The inputs X and Y are multiplexed with the outputs R and Z.



(a) Assume the elements of the vector A are fed into the pipeline through input X, one element per cycle. What is the minimum number of clock cycles required to compute the sum of an N-element vector A: $s = \sum_{I=1}^{N} A(I)$? In the absence of an operand, a value of 0 is input into the pipeline by default. Neglect the setup time for the pipeline.
(b) Let τ be the clock period of the pipelined adder. Consider an equivalent non-pipelined adder with a flow-through delay of 4τ. Find the actual speedup $S_4(64)$ and the efficiency $\eta_4(64)$ of using the above pipeline adder for $N = 64$.
(c) Find the maximum speedup $S_4(\infty)$ and the efficiency $\eta_4(\infty)$ when N tends to infinity.
(d) Find $N_{1/2}$, the minimum vector length required to achieve half of the maximum speedup.

Problem 6.9 Consider the following pipeline reservation table.

	1	2	3	4
S1	X			X
S2		X		
S3			X	

(a) What are the forbidden latencies?
(b) Draw the state transition diagram.
(c) List all the simple cycles and greedy cycles.
(d) Determine the optimal constant latency cycle and the minimal average latency.
(e) Let the pipeline clock period be $\tau = 20$ ns. Determine the throughput of this pipeline.

Problem 6.10 Consider the five-stage pipelined processor specified by the following reservation table:

	1	2	3	4	5	6
S1	X					X
S2		X			X	
S3			X			
S4				X		
S5		X				X

(a) List the set of forbidden latencies and the collision vector.
(b) Draw a state transition diagram showing all possible initial sequences (cycles) without causing a collision in the pipeline.
(c) List all the simple cycles from the state diagram.
(d) Identify the greedy cycles among the simple cycles.
(e) What is the minimum average latency (MAL) of this pipeline?
(f) What is the minimum allowed constant cycle in using this pipeline?
(g) What will be the maximum throughput of this pipeline?
(h) What will be the throughput if the minimum constant cycle is used?

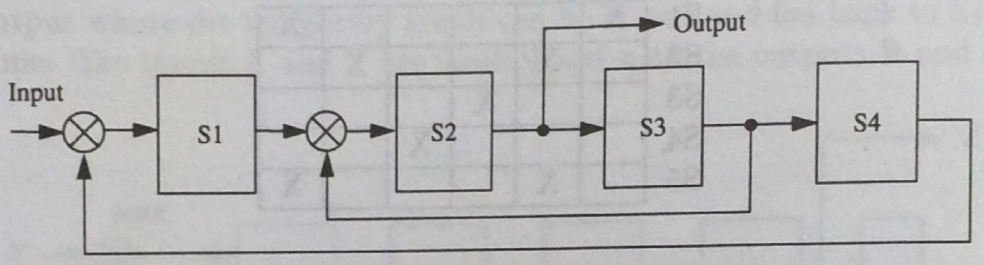
Problem 6.11 The following assembly code is to be executed in a three-stage pipelined processor with hazard detection and resolution in each stage. The stages are instruction fetch, operand fetch (one or more as required), and execution (including a write-back operation). Explain all possible hazards in the execution of the code.

Inc R0	/ R0 ← (R0) + 1 /
Mul ACC, R0	/ ACC ← (ACC) × (R0) /
Store R1, ACC	/ R1 ← (ACC) /
Add ACC, R0	/ ACC ← (ACC) + (R0) /
Store M, ACC	/ M ← (ACC) /

Problem 6.12 This problem compares the performance of a superpipelined superscalar processor of degree (m, n) with that of a base scalar processor of degree $(1, 1)$. Analyze the speedup expression $S(m, n)$ in Eq. 6.32 for the following limiting cases:

(a) Within the range $1 \leq m \leq 4$ and $1 \leq n \leq 6$, what is the optimal number of pipeline stages that would maximize the speedup $S(m, n)$?
(b) What are the practical limitations preventing the growth of the superscalar degree m?
(c) What are the practical limitations preventing the growth of the superpipeline degree n?

Problem 6.13 Consider the following pipelined processor with four stages. This pipeline has a total evaluation time of six clock cycles. All successor stages must be used after each clock cycle.



(a) Specify the reservation table for this pipeline with six columns and four rows.
(b) List the set of forbidden latencies between task initiations.
(c) Draw the state diagram which shows all possible latency cycles.
(d) List all greedy cycles from the state diagram.
(e) What is the value of the minimal average latency?
(f) What is the maximal throughput of this pipeline?

Problem 6.14 Three functional pipelines f_1, f_2, and f_3 are characterized by the following reservation tables. Using these three pipelines, a composite pipeline network is formed below:

f_1 :

	1	2	3	4
S1	X			
S2		X		
S3			X	X

f_2 :

	1	2	3	4
T1	X			X
T2		X		
T3			X	

f_3 :

	1	2	3	4
U1	X		X	
U2				X
U3		X		

Each task going through this composite pipeline uses the pipeline in the following order: f_1 first, f_2 and f_3 next, f_1 again, and then the output is obtained. The dual multiplexer selects a pair of inputs, (A,B) or (X,Y), and feeds them into the input of f_1. The use of the composite pipeline is described by the combined reservation table.