7 Large and Fast: Exploiting Memory Hierarchy

While *caches, translation lookaside buffers,* and *virtual memory* appear different, they can be understood by examining how they deal with the four questions: (1) Where can a block be placed? (2) How is a block found? (3) What block is replaced on a miss? (4) How are writes handled? *(page 512)*

The challenge of memory hierarchies is that every change that improves the *miss* rate can also negatively affect performance: Increasing size decreases *capacity misses* but may also increase *access time;* increasing *associativity* decreases miss rate due to *conflict misses* but may also increase access time; and increasing *block size* may decrease miss rate yet also increase *miss penalty. (page 514)*

8 Interfacing Processors and Peripherals

Different bus characteristics allow the creation of buses optimized for a wide range of demands. In general, higher cost systems use wider and faster buses that are *synchronous.* In contrast, low-cost systems favor buses that are narrower, do not require intelligence among the devices, and are *asynchronous* so that low-speed devices can interface inexpensively. *(page 563)*

The performance of an I/O system, whether measured by bandwidth or latency, depends on all the elements in the path between the device and memory, including the operating system that generates the I/O commands. *(page 579)*

9 Parallel Processors

A key characteristic of programs for parallel machines is the frequency of *synchronization* and *communication.* Large-scale parallel machines have *distributed physical memory;* the higher bandwidth and lower overhead of local memory reward programs utilizing *locality. (page 634)*

APPENDICES

A Assemblers, Linkers, and the SPIM Simulator

Assembly language is a programming language. Its principal difference from high-level languages such as BASIC, Pascal, and C is that it provides only a few, simple types of data and control flow. Assembly language programs do not specify the type of value held in a variable, leaving the programmer to apply the appropriate operation. *(page A-13)*

B The Basics of Logic Design

C Mapping Control to Hardware

Independent of whether the control is represented as a finite state diagram or as a microprogram, the translation to hardware is similar: Each state or microinstruction asserts a set of control outputs and specifies how to choose the next state; next state function may be encoded in a finite state machine or with an explicit sequencer; control logic may be either ROMs or PLAs. *(page C-27)*

D Introducing C to Pascal Programmers

E Another Approach to Instruction Set Architecture: VAX

The differing goals for VAX and MIPS led to different architectures. The VAX goals, simple compilers and code density, led to powerful addressing modes, powerful instructions, and efficient instruction encoding. The MIPS goals were high performance via pipelining, ease of hardware implementation, and compatibility with optimizing compilers. These goals led to simple instructions, simple addressing modes, fixed-length instruction formats, and many registers. *(page E-4)*

Orthogonality is key to the VAX architecture; the opcode is independent of the addressing modes, which are independent of the data types and even the number of unique operands. Thus a few hundred operations expand to hundreds of thousands of instructions when accounting for the data types, operand counts, and addressing modes. *(page E-23)*