In this example of a lowpass filter (the amplitude of frequencies above a certain threshold are attenuated), the complexity of digital sampling and a microprocessor appears unjustified. The power of DSP comes when much more complex analog transformations are performed that would require excessively complex analog circuit topologies. Some examples of applications in which DSPs are used include modems, cellular telephones, and radar. While sensitive analog circuits may degrade or fall out of calibration over time, digital instructions and sequences maintain their integrity indefinitely. Major manufacturers of DSPs include Analog Devices, Motorola, and Texas Instruments. Many books have been written on DSP algorithms and techniques, which are extremely diverse and challenging topics.

DSP algorithms are characterized by repetitive multiplication and addition operations carried out on the sampled data set. Multiply and addition operations are also known as *multiply and accumulate*, or MAC, operations in DSP parlance. These calculations involve the sampled data as well as coefficients that, along with the specific operations, define the transformation being performed. For DSP to be practical, it must be performed in real time, because the signals cannot be paused while waiting for the microprocessor to finish its previous operation. For DSP to be economical, this throughput must be achieved at an acceptable cost. A general-purpose microprocessor can be used to perform DSP functions, but in most cases, the solution will not be economical. This is because the microprocessor is designed to execute general programs for which there is less emphasis on specific types of calculations. A DSP is designed specifically to rapidly execute multiply and accumulate operations, and it contains additional hardware to efficiently fetch sequential operands from tables in memory. Not all of the features discussed below are implemented by all DSPs, but they are presented to provide an understanding of the overall set of characteristics that differentiates a DSP from a generic microprocessor.

At their core, DSPs contain one or more ALUs that are capable of multiplication and addition in a single cycle. This rapid calculation capability ensures that throughput can be maintained as long as operands are fed to the ALUs. DSPs are manufactured with a variety of ALU capabilities ranging from 16-bit integer to IEEE floating-point support. As with a generic microprocessor, the number of ALUs influences how many simultaneous operations can be carried out at a given time. To keep the ALUs supplied with operands, DSPs contain hardware structures called *address generators* that automatically calculate the addresses of the next operands to be used in a calculation. Sampled data is stored in a memory array, and algorithmic coefficients are stored in a separate array. Depending on the algorithm, the array entries may not be accessed sequentially. On a generic microprocessor, the software would have to add an arbitrary offset value to an index register each time a new operand was desired. Additionally, as a result of fixed array sizes, the pointer eventually wraps around from the end to the beginning, thereby requiring additional instructions to check for the wrap condition. This index register overhead slows the computation process. Address generators offload this overhead to hardware by associating additional registers with the index registers. These registers define the increment to be applied to an index register following a load or store operation and also define the start and end addresses of the memory array. Therefore, software is able to execute load/store and calculation operations without spending time on routine pointer arithmetic.

The specialized ALU and address generation hardware within the DSP core place a high demand on memory to maintain a steady flow of instructions and data. DSPs commonly implement a Harvard memory architecture in which separate buses connect to program and data memory. Most DSPs contain separate program and data memory structures integrated onto the same chip as the DSP core for minimal access latency to small repetitive DSP algorithm kernels. Program memory may be implemented as ROM or RAM, depending on whether an external interface is available from which to load programs. These on-chip memories may be as small as several kilobytes each for less expensive DSPs or hundreds of kilobytes for more powerful products. To mitigate the complexity of a Harvard architecture on the overall system design, most DSPs contain a unified external memory bus for con-

nection to external ROM and RAM. A DSP application can boot from external ROM, then load its kernel into on-chip program memory and perform the majority of its execution without fetching additional instructions from external memory.

## 7.8 PERFORMANCE METRICS

Evaluating the throughput potential of a microprocessor or a complete computer is not as simple as finding out how fast the microprocessor's clock runs. System performance varies widely according to the applications being used and the computing hardware on which they run. Applications vary widely in their memory and I/O usage, both being properties whose performance is directly tied to the hardware architecture. We can consider three general sections of a computer and how each influences the speed at which an application is executed: the microprocessor, the memory system, and the I/O resources.

The usable address space of a microprocessor is an important consideration, because applications vary in their memory needs. Some embedded applications can fit into several kilobytes of memory, making an 8-bit computer with 64 kB or less of address space quite adequate. More complex embedded applications start to look like applications that run on desktop computers. If large data arrays are called for, or if a multitasking system is envisioned whereby multiple tasks each contain megabytes of program memory and high-level data structures, a 32-bit microprocessor with hundreds of megabytes of usable address space may be necessary. At the very high end, microprocessors have transitioned to 64-bit architectures with gigabytes of directly addressable memory. A microprocessor's address space can always be expanded externally by banking methods, but banking comes at a penalty of increased time to switch banks and the complexity of making an application aware of the banking scheme.

Any basic type of application can run on almost any microprocessor. The question is how fast and efficiently a particular microprocessor is able to handle the application. The instruction set is an important attribute that should be considered when designing a computer system. If a floating-point intensive application is envisioned, it should probably be run on a microprocessor that contains an FPU, and the number of floating-point execution units and their execution latencies is an important attribute to investigate. An integer-only microprocessor could most likely run the floating-point application by performing software emulation of floating-point operations, but its performance would probably be rather dismal. For smaller-scale computers and applications, these types of questions are still valid. If an application needs to perform frequent bit manipulations for testing and setting various flags, a microprocessor that directly supports bit manipulation may be better suited than a generic architecture with only logical AND/OR type instructions.

Once a suitable instruction set has been identified, a microprocessor's ability to actually fetch and execute the instructions can become an important part of system performance. On smaller systems, there are few variables in instruction fetch and execution: each instruction is fetched and executed sequentially. Superscalar microprocessors, however, must include effective instruction analysis logic to properly utilize all the extra logic that has been put onto the chip and that you are paying for. If the multiple execution units cannot be kept busy enough of the time, your application will not enjoy the benchmark performance claims of the manufacturer. Vendors of high-performance microprocessors devote much time to instruction profiling and analysis of instruction sequences. Their results improve performance on most applications, but there are always a few niche applications that have uncommon properties that can cause certain microprocessors to fall off in performance. It pays to keep in mind that common industry benchmarks of microprocessor performance do not always tell the whole story. These tests have been around for a long time, and microprocessor designers have