additions. If each addition requires Q nanoseconds and each multiplication requires $2 \times Q$ nanoseconds, multiplying two matrices will require a total of:

$$T_{total} = 2 \times Q \times N^3 + Q \times (N^3 - N^2)$$

As an alternative to precise analysis, engineers use a weighted average. That is, instead of calculating the exact number of times each instruction is executed, an approximate percentage is used. For example, suppose a graphics program is run on many input data sets, the number of floating point operations is counted to obtain the list in Figure 19.1.

Instruction Type	Count	Percentage
Add	8513508	72
Subtract	1537162	13
Multiply	1064188	9
Divide	709458	6

Figure 19.1 Example of instruction counts for a graphics application run on many input values. The third column shows the relative percentage of each instruction type.

Once a set of instruction counts has been obtained, the performance of hardware can be assessed by using a weighted average. When the graphics application is run on the hardware described above, we expect the average time for each floating point instruction to be:

$$T_{avg'} = .72 \times Q + .13 \times Q + .09 \times 2\,Q + .06 \times 2\,Q = 1.16\,Q \ \textit{ns per instruction}$$

As the example shows, a weighted average can differ significantly from a uniform average. In this case, the weighted average is 23% less than the average in equation (19.1) that was obtained using uniform instruction weights†.

19.5 Instruction Mix

Although it provides a more accurate measurement of performance, the weighted average example above only applies to one specific application, and only assesses floating point performance. Can we give a more general assessment? One approach has become popular: use a large set of programs to obtain relative weights for each type of instruction, and then use the relative weights to assess the performance of a given architecture. That is, instead of focusing on floating point, keep a counter for each instruc-

†Equation 19.1 can be found on page 313.