T_1	T_2
read(A)	
$A := A - 50$	
write(A)	
read(B)	
$B := B + 50$	
write(B)	
commit	
	read(A)
	$temp := A * 0.1$
	$A := A - temp$
	write(A)
	read(B)
	$B := B + temp$
	write(B)
	commit

Figure 17.2 Schedule 1—a serial schedule in which T_1 is followed by T_2.

Suppose the current values of accounts A and B are \$1000 and \$2000, respectively. Suppose also that the two transactions are executed one at a time in the order T_1 followed by T_2. This execution sequence appears in Figure 17.2. In the figure, the sequence of instruction steps is in chronological order from top to bottom, with instructions of T_1 appearing in the left column and instructions of T_2 appearing in the right column. The final values of accounts A and B, after the execution in Figure 17.2 takes place, are \$855 and \$2145, respectively. Thus, the total amount of money in accounts A and B—that is, the sum $A + B$—is preserved after the execution of both transactions.

Similarly, if the transactions are executed one at a time in the order T_2 followed by T_1, then the corresponding execution sequence is that of Figure 17.3. Again, as expected, the sum $A + B$ is preserved, and the final values of accounts A and B are \$850 and \$2150, respectively.

The execution sequences just described are called **schedules**. They represent the chronological order in which instructions are executed in the system. Clearly, a schedule for a set of transactions must consist of all instructions of those transactions and they must preserve the order in which the instructions appear in each individual transaction. For example, in transaction T_1, the instruction write(A) must appear before the instruction read(B), in any valid schedule. Note that we include in our schedules the **commit** operation to indicate that the transaction has entered the committed state. In the following discussion, we shall refer to the first execution sequence (T_1 followed by T_2) as schedule 1, and to the second execution sequence (T_2 followed by T_1) as schedule 2.