sequential, implicitly ordered operations versus the unrestricted compute model is especially important in FPGA designs and is a topic that we will continue to address throughout this book.

A subtle point is that it is commonplace to refer to some components of our Platform FPGA system as "hardware" when, in fact, these components are written like software. That is, the component is a specification of how the device is to be configured. The real, physical hardware is the FPGA device itself. Thus, in place of the conventional definition of hardware given earlier, we will distinguish hardware and software components of our system based on their model of execution. Earlier we said that software is characterized by a sequential execution model and that the hardware execution model is distinctly nonsequential. In other words, software will refer to specifications intended to be executed by a processor, whereas hardware will refer to specifications intended to be executed on the fabric of an FPGA and, on occasion, in a broader sense, the physical components of the computing system. To make this important distinction more concrete, consider the following computation:

```
R0+R1+R2 → Acc
```

Assume that all four names are registers. The computation implemented in the sequential model is illustrated in Figure 1.3. It shows the traditional fetch-execute machine. The dashed box highlights on the main mechanism. The ALU circuit is time-multiplexed to perform each addition sequentially. A substantial part of the circuit is used to decode and control the time steps. This is commonly referred to as the von Neumann stored-program compute model. We will continue to refer to it as the sequential execution model. In contrast, a nonsequential execution model can take a wide variety of forms, but is clearly distinguished from sequential execution by the absence of a general-purpose controller that sequences operations and the explicit time-multiplexing of named, fixed circuits. A nonsequential execution implementation of this computation is illustrated in Figure 1.4. This is generally known as a *data-flow model* of execution, as it is the direct implementation of a data-flow graph. (We use the broader sense of the term throughout this book; the data-flow architectures studied in the 1980s and 1990s incorporate more detail than this.)

In terms of how these two computing models work, consider their operation over time. In Figure 1.5, time advances from top to bottom. Each "slot" delineated by the dotted lines indicates what happens during one clock cycle. It is important to note that the speed of the two computing models cannot be compared by counting the number of cycles — the clock frequencies for each

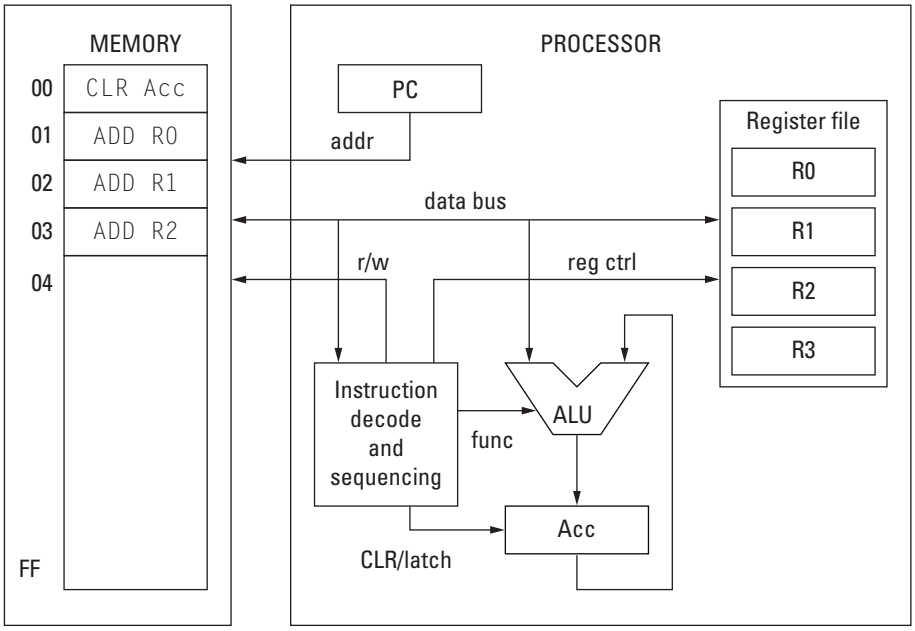

Figure 1.3. A sequential model.

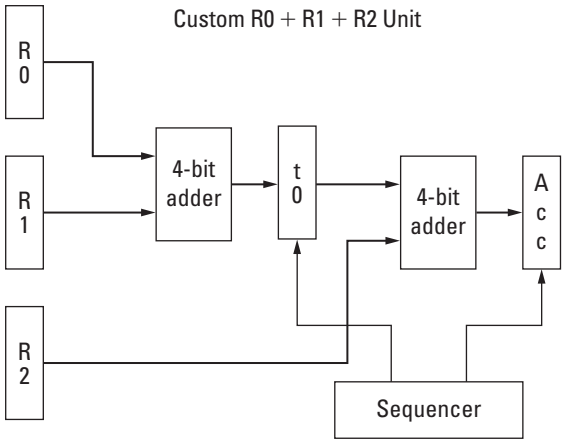

Figure 1.4. A nonsequential model.

are different. A typical FPGA clock period might be 5× to 10× longer than a processor implemented in the same technology. Also, modern processors typically operate on multiple instructions simultaneously, which has become crucial to their performance.

With this understanding, we can say that software runs on a processor where a *processor* is hardware that implements the

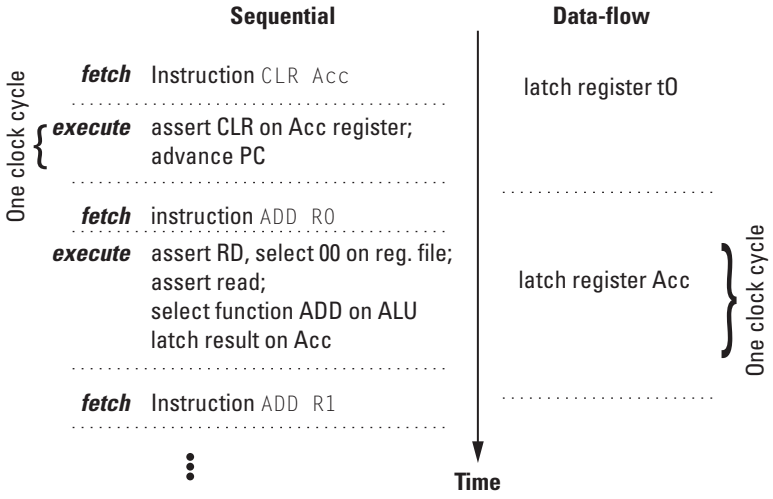

Figure 1.5. A cycle-by-cycle operation of sequential and data-flow models.

sequential execution model. Note that the terms Central Processing Unit (CPU) and microprocessor are common synonyms for processor. Hardware will be the specification that we use to configure the FPGA and does not use the sequential execution model.

1.2. Design Challenges

Now that we have a better understanding of what an embedded system is, it is worthwhile to understand a little bit about how embedded systems projects work. This section discusses the life cycle of a typical project, describes typical measures of embedded system success, and closes with a summary of costs.

1.2.1. Design Life Cycle

There are many books on how to manage an engineering project. Our goal in this section is simply to highlight a couple of terms and describe the design life cycle of a project to provide context for remaining sections. This life cycle, illustrated in Figure 1.6, is sometimes referred to as the "waterfall model." It is meant to suggest that going upstream (i.e., returning to a previous stage because of a bug or mistake in the design) is considerably more challenging than going downstream.

Whether initiated by a marketing department or by the end user, the first stage is called *requirements*. As the name suggests, the step is to figure out what the product is required to do. Often this will start as a very broad, abstract statement, and it is the developer's job to establish a concrete list of user measurable