This chapter introduces many concepts essential to understanding SITDAC programming and the remaining chapters of this book. Chapter 1 explained that bit-serial SIMD computers share many properties with SITDAC associative computers. As a result, they also share many programming techniques. However, since SIMD computers are not widely known, this chapter will discuss several basic concepts that facilitate the programming of associative and SIMD computers. The first concept is SIMD parallelism; section 2.1 describes the execution of conditional statements. Section 2.2, on flow of control, describes more complex conditional-execution sequences. The classroom model described in section 2.3 describes a design aid based on a familiar situation: Many associative-programming situations can be directly mapped onto a teacher as the sequential control and students as individual associative cells.

Memory allocation or, more accurately, cell allocation, which is very simple and efficient in SITDAC programming, is discussed in section 2.4. Section 2.5 explains massive data-parallel searching using the classroom model and shows that associative searching is an alternative to pointers for accessing data. Section 2.6 establishes the special relationship of tabular data arrays in SITDAC programming. A SITDAC computer does not have to be bit serial. However, many SIMD computers are. Consequently, section 2.7 describes bit-serial arithmetic and shows that it is an approach already known by everyone.

Section 2.8 introduces the most important basic concept in this book. It describes how the responder hardware identified in Chapter 1 as distinguishing an associative computer from associative and SIMD processors is used to develop powerful associative searching routines. Section 2.9 describes how the step/find hardware is used for iteration.

Section 2.10 is on corner turning. Corner turning is simply the data reorganization required to convert conventional sequential data files into a format
suitable for bit-serial computation. Bucket-brigade communication (section 2.11) describes a learning aid that facilitates the understanding of grid-interconnection network communications. Section 2.12 describes the rationale for mapping multidimensional data onto multidimensional memories. Section 2.13 describes how data can be duplicated in parallel arrays for little additional overhead, a process that may allow considerably more parallelism. Section 2.14 reviews this chapter.

2.1. SIMD Parallelism

This section describes how statements are executed conditionally in a SIMD environment. The SITDAC model for associative computing assumes a basic SIMD flow of control. That is, a single instruction stream is always broadcast to all of the parallel processors. Each processor has its own mask bit which it can set to 1 (execute or TRUE) or 0 (ignore or FALSE) based on the results of local computation. The mask bits are used to determine which processors execute the instruction stream and which processors ignore it. The instruction stream may also contain global set commands, which are executed by the processors regardless of the state of the mask bit.

This situation can be likened to a TV station broadcasting programs to all of the homes in an area. The TV on–off switch corresponds to the mask bit. Each person in a home (which represents the information content of a cell) determines whether or not to turn on the switch and watch the TV. There are other broadcast signals, such as a severe-weather siren, that are heard by everyone regardless of the position of the TV on–off switch and that demand that everyone turn on the TV if it is not already on and pay attention to the forthcoming message.

The conditional execution of a parallel if statement illustrates the use of the mask bit. In general, all parts of an if statement must be broadcast in the instruction stream since there is a high probability that both true and false results are present in the parallel data. So, for example, as illustrated in Fig. 2-1, a typical if-statement instruction sequence would be

- Set all mask bits.
- Broadcast code to calculate the if condition.
- Set the individual cell mask bit to
  TRUE if the local condition is true.
  FALSE otherwise.
- Broadcast code for the TRUE or then portion of the if statement.
- Complement the mask bits; i.e.,
  if the condition was true, set the mask bit to FALSE.
  if the condition was false, set the mask bit to TRUE.
- Broadcast code for the FALSE or else portion of the if statement.