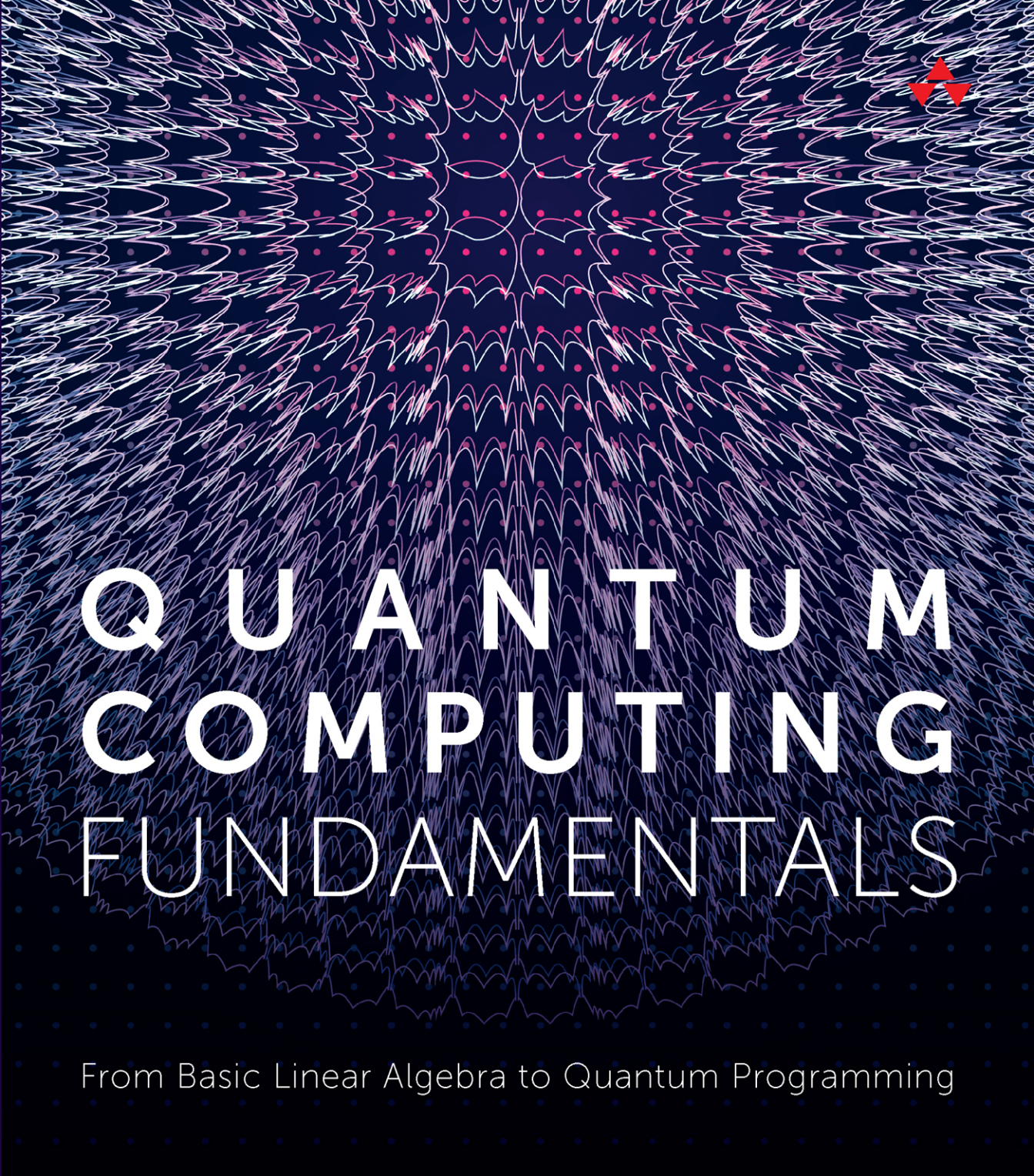

QUANTUM COMPUTING FUNDAMENTALS

From Basic Linear Algebra to Quantum Programming

DR. CHUCK EASTTOM

Quantum Computing Fundamentals

4.5 Logic Gates

Logic gates play a critical role in computer science and are a central issue in quantum computing. In traditional computers, logic gates are implemented usually via transistors, or some similar object. They perform a basic logical operation on input to produce output. Keep in mind that a classical computer can only see 1s and 0s. Thus, binary numbers and binary operations are ultimately all the computer works with. You may ultimately be performing trigonometry or calculus, but the computer has to break this down into binary math steps.

Before proceeding into logic gates, a brief discussion of basic binary operations is in order. For many readers, this will be a review, but in case it is not, this is essential to understanding classical computer logic gates. The binary number system was developed in its modern form by Gottfried Leibniz. He and Isaac Newton independently discovered calculus. The three operations of interest are AND, OR, and XOR operations.

4.5.1 AND

To perform the AND operation, you take two binary numbers and compare them one place at a time. If both numbers have a 1 in both places, then the resultant number is a 1. Otherwise, the resultant number is a 0, as you see here:

```
1 1 0 0
1 0 0 1

_____________
1 0 0 0
```

4.5.2 OR

The OR operation tests to determine if there is a 1 in either or both numbers in a given place. If so, then the resultant number is 1. Otherwise, the resultant number is 0, as you see here:

```
1 1 0 0
1 1 0 1

_____________
1 1 0 1
```

4.5.3 XOR

The XOR operation tests if there is a 1 in a number in a given place, but *not* in both numbers at that place. If it is in one number but not the other, then the resultant number is 1. Otherwise, the resultant number is 0, as you see here:

```
1 1 0 1
1 0 0 1

_____________
0 1 0 0
```

The term XOR means "exclusively OR" rather than and/or. XORing has a very interesting property in that it is reversible. If you XOR the resultant number with the second number, you get back the first number, and if you XOR the resultant number with the first number, you get the second number:

```
0 1 0 0
1 0 0 1

_____________
1 1 0 1
```

In the 1930s, an engineer at NEC named Akira Nakashima introduced a switching circuit theory using two-valued Boolean algebra. This is often considered the beginning of modern logic gates. There are standards for symbols used in logic gates: first, the ANSI/IEEE Std 91-1984, then the revision, ANSI/IEEE Std 91a-1991.

4.5.4 Application of Logic Gates

The AND gate implements a truth table that uses the binary AND operation. It may be helpful to first consider the ANSI/IEEE standard diagram for an AND gate, shown in Figure 4.8.

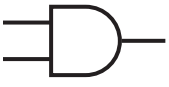

FIGURE 4.8 The AND gate

Two inputs lead to a single output. This is done with a simple truth table, much like the one shown in Table 4.2.

TABLE 4.2 Truth Table for AND

Input A	Input B	Output
1	1	1
1	0	0
0	0	0
0	1	0

Thus, what the AND gate does is take in two bits as input and perform the binary AND operation on them. It then ends out the output.

The OR gate is quite similar. The ANSI/IEEE symbol is shown in Figure 4.9.

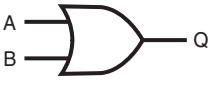

FIGURE 4.9 The OR gate

The two inputs lead to a single output. This is accomplished using a truth table, much like the one you saw in the previous table, only with the binary OR operation instead of AND (see Table 4.3).

TABLE 4.3 Truth Table for OR

Input A	Input B	Output
1	1	1
1	0	1
0	0	0
0	1	1

Of course, there is also an XOR gate for the "exclusive or" operation. This is sometimes referred to as an EOR gate or an EXOR gate. Figure 4.10 shows the ANSI/IEEE symbol for the XOR gate.

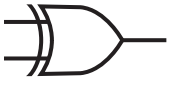

FIGURE 4.10 The XOR gate

The two inputs lead to a single output. This is accomplished using a truth table much like the ones you saw in the previous two tables, only with the binary XOR operation being used rather than AND or OR (see Table 4.4).

TABLE 4.4 Truth Table for XOR

Input A	Input B	Output
1	1	0
1	0	1
0	0	0
0	1	1

These three are simple logic gates, based on the basic three binary operations; however, there are many variations. A very common gate used is the NAND gate, which is a NOT-AND gate. Basically, it outputs false, only when all of the inputs are 1 (true). There are many systems using NAND gates. Figure 4.11 shows the ANSI/IEEE symbol for the NAND gate.

FIGURE 4.11 The NAND gate

The truth table for this one essentially says that if it is NOT an AND value (i.e., not both values a 1), then output 0 (see Table 4.5).

TABLE 4.5 Truth Table for NAND

Input A	Input B	Output
1	1	0
1	0	1
0	0	1
0	1	1

The NAND gate is very important because any Boolean function can be implemented using some combination of NAND gates. Another gate that has this property is the NOT-OR (or NOR) gate. Figure 4.12 shows the ANSI/IEEE symbol for a NOR gate.

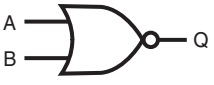

FIGURE 4.12 The NOR gate

The truth table for this one essentially says that if it is NOT an OR value (i.e., neither value is a 1), then output 0 (see Table 4.6).

TABLE 4.6 Truth Table for NOR

Input A	Input B	Output
1	1	0
1	0	0
0	0	1
0	1	0

Because either NOR gates or NAND gates can be used to create any Boolean function, they are called *universal gates*.

You might be wondering how explicitly 1s and 0s are implemented in circuitry. A common method is for a high voltage value to be a 1, and low to be a 0. So, you can see how ultimately mathematical operations are reduced to electricity flowing through logic gates and binary operations being performed on them.

The Hadamard gate is one of the more common logic gates utilized by a quantum computer. We will discuss these gates and others in more detail in later chapters. However, a brief introduction here provides a contrast to classical logic gates. The Hadamard gate acts on a single qubit and is a

one-qubit version of the quantum Fourier transform. It is often denoted by the Hadamard matrix shown in Equation 4.1.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

EQUATION 4.1 The Hadamard Matrix

Hadamard matrices are named after the mathematician Jacques Hadamard. They are square matrices whose rows are orthogonal. They are used in quantum computing to represent a logical gate, the Hadamard gate.

4.6 Computer Architecture

Computer architecture is how we design and describe computer systems. Instruction set architecture is a substantial subtopic of computer architecture. While human programs deal with programming languages such as C++ and Java, the machine has instruction sets that are used for low-level actions. If you have ever looked at assembly language, you have seen something quite close to the actual computer instructions on the microprocessors. Assembly gets its name from the fact that software named an assembler translates the assembly code into machine code.

There are several subtopics in the field of computer architecture. Instruction Set Architecture (ISA) is the model of the architecture that is realized in things like the central processing unit. The ISA defines data types, registers, addressing, virtual memory, and other fundamental features of the computer. There are many different ways to classify architecture. Two common ways are the CISC (complex instruction set) and RISC (reduced instruction set). CISC processors have a large number of specialized instruction, whereas RISC implements a smaller set of instructions. This, of course, brings us to the question, what is an instruction? An instruction is a simple statement of what the processor should do. It often involves moving small pieces of data into and out of registers in the CPU. For example, in basic arithmetic, the contents of two registers are used and the result is then stored in a register. There are also control flow operations such as branching to another location (including conditionally branching) and calling some other block of code.

Assembly code is how one directly programs code for a CPU. The code tends to be substantially longer than other programming languages such as Java, Python, C, etc. This is because the programmer must issue every step of a command. To show the difference, "Hello, World!" in C is written like this:

```
printf("Hello, World!");
```

In Java, the program is written like this:

```
System.out.println("Hello, World!");
```

In assembly code, however, it is written like this:

```
          global    _start

          section   .text
_start:   mov       rax, 1          ; system call for write
          mov       rdi, 1              ; file handle 1 is stdout
          mov       rsi, message  ; address of string to output
          mov       rdx, 13           ; number of bytes
          syscall                         ; invoke system to write
          mov       rax, 60
          xor       rdi, rdi                ; exit code 0
          syscall                           ; invoke system  to exit
          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

The reason for this apparent complexity is that the programmer has to literally move data onto and off of specific CPU registries. Programming assembly gives one a really good understanding of CPU architecture.

Microarchitecture deals with how a processor is actually organized. This is the computer engineering of the processing chip, and it generally leads to diagrams such as the one shown in Figure 4.13.

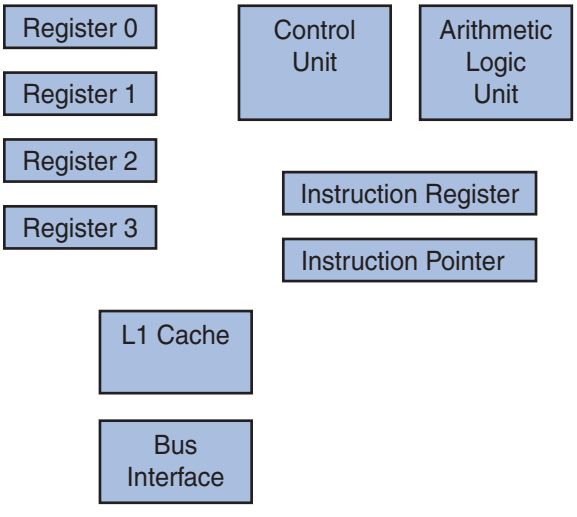

FIGURE 4.13 The CPU architecture

The image in Figure 4.13 is a very simplified CPU architecture, but it is useful for getting the general idea of microarchitecture. The idea is to design the various processing pathways. In Figure 4.13, we see just the general overview of the CPU. Memory is mapped out similarly, as are the various components of a motherboard, including the bus. The bus is the communication pathway for data transfer inside of a computer.