2 The hardware

I always loved that word, Boolean.
Claude Shannon[1]

Going through the layers

In the last chapter, we saw that it was possible to logically separate the design of the actual computer hardware – the electromagnetic relays, vacuum tubes, or transistors – from the software – the instructions that are executed by the hardware. Because of this key abstraction, we can either go down into the hardware layers and see how the basic arithmetic and logical operations are carried out by the hardware, or go up into the software levels and focus on how we tell the computer to perform complex tasks. Computer architect Danny Hillis says:

> This hierarchical structure of abstraction is our most important tool in understanding complex systems because it lets us focus on a single aspect of a problem at a time.[2]

We will also see the importance of "functional abstraction":

> Naming the two signals in computer logic 0 and 1 is an example of functional abstraction. It lets us manipulate information without worrying about the details of its underlying representation. Once we figure out how to accomplish a given function, we can put the mechanism inside a "black box," or a "building block" and stop thinking about it. The function embodied by the building block can be used over and over, without reference to the details of what's inside.[3]

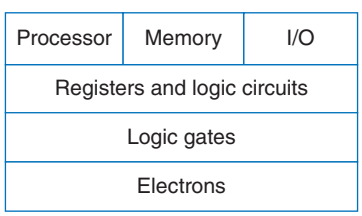


Fig. 2.1. A diagram showing the major abstraction layers of computer hardware.

In this chapter, like Strata Smith going down the mines, we'll travel downward through the hardware layers (Fig. 2.1) and see these principles in action.

George Boole and Claude Shannon

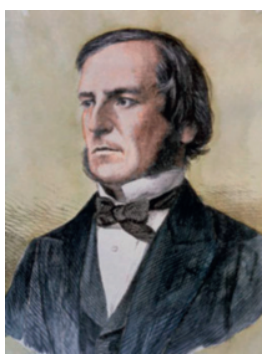
In the spring of 1936, Vannevar Bush was looking for a smart electrical engineering graduate who could assist visiting scientists in setting up their calculations on his Differential Analyzer at MIT. Claude Shannon (B.2.1), a

twenty-year-old graduate from the University of Michigan, applied and got the position. Although the Differential Analyzer was a mechanical machine with many spinning discs and rods, there was, as Shannon said later, also "a complicated control circuit with relays."[4] A relay is just a mechanical switch that can be opened or closed by an electromagnet, so that it is always in one of two states: either on or off, just like a light switch. Bush suggested that a study of the logical organization of these relays could be a good topic for a master's thesis. Shannon agreed, and drawing on his undergraduate studies in symbolic logic, he began trying to understand the best way to design complicated relay circuits with hundreds of relays. Commenting on the importance of symbolic logic in this endeavor, Shannon later said that "this branch of mathematics, now called Boolean algebra, was very closely related to what happens in a switching circuit."[5] Let's see how this comes about.

B.2.1. Claude Shannon (1916–2001) is often referred to as the father of information technology. He is credited with two groundbreaking ideas: the application of Boolean algebra to logical circuit design and digitization of information. In this photograph, he is holding a mechanical mouse that can learn from experience as it moves around the complicated maze.

George Boole (B.2.2) was a nineteenth-century, self-taught mathematician whose best-known work is a book called *An Investigation of the Laws of Thought*. In this book, Boole tried to reduce the logic of human thought to a series of mathematical operations in which decisions are predicated on determining whether any given logical statement is true or false. It was the Greek philosopher Aristotle who introduced what is now called propositional logic. This was a form of reasoning that enabled him to deduce new conclusions by combining true propositions in a "syllogism" of the form:

Every Greek is human.
Every human is mortal.
Therefore, every Greek is mortal.



B.2.2. George Boole (1815–64) invented the algebra of 0s and 1s by introducing the rules and notation for describing logic. He was a self-taught mathematician and at the age of thirty-four became a professor of mathematics at Queens College in Cork, Ireland. Boole was widely recognized for his work aiming to combine algebra and logic. De Morgan, the leading logician of the time wrote: "Boole's system of logic is but one of many proofs of genius and patience combined."[B1]

Boole devised a language for describing and manipulating such logical statements and for determining whether more complex statements are true or false. Equations involving such statements can be written down using the logical operations AND, OR, and NOT. For example, we can write an equation to express the obvious fact that if neither statement A nor statement B is true, then both statements A and B must be false:

```
NOT (A OR B) = (NOT A) AND (NOT B)
```

This equation is actually known as De Morgan's theorem, after Boole's colleague, Augustus De Morgan (B.2.3). It is a simple, (but as we shall see) powerful expression of Boolean logic. In this way, Boolean algebra allows much more complex logical statements to be analyzed.

Shannon realized that relays combined in circuits were equivalent to combining assertions in Boolean algebra. For example, if we connect two relays, A and B, in series and apply a voltage to the ends, current can only flow if both relays are closed (Fig. 2.2). If we take the closed state of a relay as corresponding to "true," then this simple connection of two relays corresponds to an AND operation: both relays must be closed – both true – for current to flow. Similarly, if we connect up the two relays in parallel, this circuit performs an OR operation, since current will flow if either relay A or B is closed or true (Fig. 2.3).

Shannon's master's thesis, "A Symbolic Analysis of Relay and Switching Circuits," showed how to build electrical circuits that were equivalent to

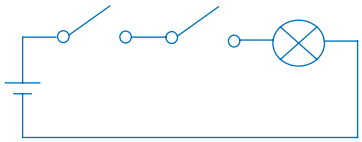


Fig. 2.2. Two relay switches in series serving as an AND gate. The current can only flow if both relays are closed.

expressions in Boolean algebra. Relays are switches that can be closed or open corresponding to a state being logical true or false, respectively. This means that any function that can be described as a correct logical statement can be implemented as a system of electrical switches. Toward the end of his thesis, Shannon points out that true and false could equally well be denoted by 1 and 0, respectively, so that the operation of the set of switches corresponds to an operation in binary arithmetic. He wrote:

> It is possible to perform complex mathematical operations by means of relay circuits. Numbers may be represented by the position of relays and stepping switches, and interconnections between sets of relays can be made to represent various mathematical operations.[6]

B.2.3. Augustus De Morgan (1806–71) is known for his pioneering work in logic, including the formulation of the theorem that bears his name.

As an example, Shannon showed how to design a relay circuit that could add two binary numbers. He also noted that a relay circuit can make comparisons and take alternative courses of actions depending on the result of the comparison. This was an important step forward, since desktop calculators that could add and subtract had been around for many years. Shannon's relay circuits could not only add and subtract, they could also make decisions.

Let us now look at how Shannon's insights about relay circuits and Boolean algebra transformed the way early computer builders went about designing their machines. Computer historian Stan Augarten says:

> Shannon's thesis not only helped transform circuit design from an art into a science, but its underlying message – that information can be treated like any other quantity and be subjected to the manipulation of a machine – had a profound effect on the first generation of computer pioneers.[7]

But before we take a deeper look at relay circuits and Boolean algebra, we should say a few words about binary arithmetic.

Binary arithmetic, bits, and bytes

Although some of the early computers – like the ENIAC – used the familiar decimal system for their numerical operations, it turns out to be much simpler to design computers using binary arithmetic. Binary operations are simple, with no need to memorize any multiplication tables. However, we pay a price for these easy binary operations by having to cope with longer numbers: a dozen is expressed as 12 in decimal notation but as 1100 in binary (B.2.4, Fig. 2.4).

Mathematics in our normal decimal system works on base 10. A number is written out in "positional notation," where each position to the left represents 10 to an increasing power. Thus when we write the number 4321 we understand this to mean:

$$4321 = (4 \times 10^3) + (3 \times 10^2) + (2 \times 10^1) + (1 \times 10^0)$$

Here 10^0 is just 1, 10^1 is 10, 10^2 is 100, and so on; the numbers of powers of ten in each field are specified by digits running from 0 to 9. In the binary system, we use base 2 instead of base 10, and we specify numbers in powers of two and use only the two digits, 0 and 1. Thus the binary number 1101 means:

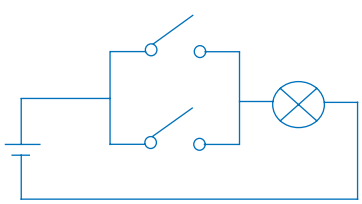


Fig. 2.3. Two relay switches in parallel equivalent to an OR gate. The current can flow if at least one of the relays is closed.

B.2.4. Gottfried Wilhelm Leibniz (1646–1716) was a German mathematician and philosopher who is credited with the discovery of binary numbers. Leibniz described his ideas in a book titled *The Dyadic System of Numbers*, published in 1679. His motivation was to develop a system of notation and rules for describing philosophical reasoning. Leibniz was fascinated by binary numbers and believed that the sequences of 0s and 1s revealed the mystery of creation of the universe and that everything should be expressed by binary numbers. In 1697 he wrote a letter to the Duke of Brunswick suggesting a design for a celebration of binary numbers to be minted in a silver coin. The Latin text at the top of the coin reads "One is sufficient to produce out of nothing everything." On the left of the table there is an example of binary addition and on the right, an example of multiplication. At the bottom the text in Latin says the "Image of Creation."

Fig. 2.4. A silver coin capturing the concept of binary numbers.

$$1101 = (1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 13 \text{ in decimal}$$

We can add and multiply binary numbers in a similar but much simpler way as in the decimal system. When adding in the decimal system, we automatically align the powers of ten in the numbers and perform a "carry" to the next power when needed, for example:

$$\begin{array}{r} 47 \\ +85 \\ \hline 132 \end{array}$$

For binary addition we have similar sum and carry operations. Adding the decimal numbers 13 and 22 using binary notation leads to:

$$\begin{array}{r} 1101 \\ +\ 10110 \\ \hline 100011 \end{array}$$

We have just used the basic rules of binary addition:

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 0 &= 1 \\ 1 + 1 &= 0 \text{ plus a carry } 1 \end{aligned}$$

Let's take a look at multiplication in both the decimal and binary systems by multiplying 37 by 5 in both decimal and binary notation:
In decimal:

$$\begin{array}{r} 37 \\ \times\ 5 \\ \hline 185 \end{array}$$

In binary:

$$\begin{array}{rl} 100101 & \\ \times 101 & \\ \hline 100101 & \\ \underline{10010100} & \text{(shifted two places to the left)} \\ 10111001 & \end{array}$$

As this example shows, all we need to perform binary multiplication is an implementation of these basic rules of binary addition including the carry and shift operations.

We can perform a binary addition physically by using a set of strips of plastic with compartments, rather like ice cube trays, and small pebbles to specify the binary numbers. An empty compartment corresponds to the binary digit 0; a compartment containing a pebble represents the binary digit 1. We can lay out the two strips with the numbers to be added plus a strip underneath them to hold the answer (Fig. 2.5).

The abstract mathematical problem of adding 1101 to 10110 has now been turned into a set of real world rules for moving the pebbles. Remarkably, with these simple rules we can now add two numbers of any size (Fig. 2.6). What this shows is that the basic operations of addition and multiplication can be reduced to a very simple set of rules that are easy to implement in a variety of technologies – from pebbles to relays to silicon chips. This is an example of functional abstraction.

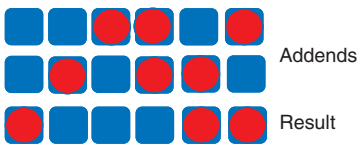


Fig. 2.5. Addition of binary numbers as illustrated by pebbles.

The word *bit* was used by Shannon as a contraction of the phrase "binary digit" in his groundbreaking paper "A Mathematical Theory of Communication." In this paper, Shannon laid the foundations for the new field of information theory. Modern computers rarely work with single bits but with a larger grouping of eight bits, called a "byte," or with groupings of multiple bytes called "words." The ability to represent all types of information by numbers is one of the groundbreaking discoveries of the twentieth century. This discovery forms the foundation of our digital universe. Even the messages that we have sent out into space are encoded by numbers (Fig. 2.7).


Fig. 2.6. A Russian abacus, used in almost all shops across the Soviet Union well into the 1990s. It is fast, reliable, and requires no electricity or batteries. It was common practice in shops that the results calculated by an electronic till were double-checked on the abacus.

Universal building blocks

With this introduction, we can now explain how a small set of basic logic building blocks can be combined to produce any logical operation. The logic blocks for AND and OR are usually called logic gates. These gates can be regarded just as "black boxes," which take two inputs and give one output depending entirely on the inputs, without regard to the technology used to implement the logic gate. Using 1s and 0s as inputs to these gates, their operation can be summarized in the form of "truth tables." The truth table for the AND gate is shown in Figure 2.8, together with its standard pictorial symbol. This table reflects the fundamental property of an AND gate, namely, that the output of A AND B is 1 only if both input A is 1 and input B is 1. Any other combination of inputs gives 0 for the output. Similarly, the truth table for the OR

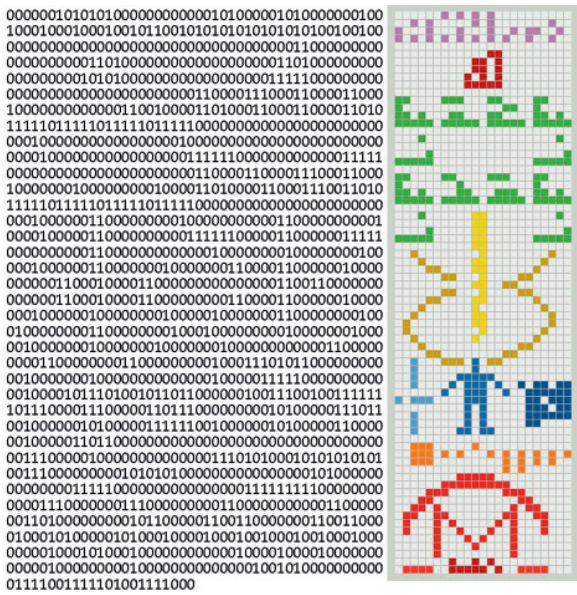


Fig. 2.7. This photograph shows the binary message that was sent out to space by the Arecibo radio telescope in November 1974. Using numbers to represent information is expected to be a universally understandable way to communicate with alien civilizations. The message shown here contains 1,679 binary digits arranged in seventy-three rows and twenty-three columns. The first ten rows represent numbers from 1 to 10 in binary format. The following rows are numbers that represent our genetic basis: the atomic numbers of chemical elements that form DNA, followed by the chemical formulas of the bases in DNA, the shape of the double helix, and the number of nucleotides in DNA. These are followed by the figure and height of an average man, the population of Earth, the position of the planet in our solar system, and the dimensions of the radio antenna broadcasting this message.

gates can be written as in Figure 2.9, where we also show the usual pictorial symbol for an OR gate. For the output of the OR gate to be 1, either or both of the inputs A and B have to be 1.

For a complete set of logic gates from which we can build any logical operation, it turns out that we need to supplement these AND and OR gates by another, very simple gate, the NOT, or invert, operation (see Fig. 2.10). This is a black box that inverts the input signal. If the input is 1, the NOT gate outputs a 0; if the input is a 0, NOT outputs a 1. The truth table for the NOT gate is shown in with the usual symbol for NOT. De Morgan's theorem allows one to play around with these operators and find complicated equivalences between combinations of operators, such as writing an OR gate in terms of AND and NOT gates. Another example is joining the OR and NOT gates to construct a NOR gate (see Fig. 2.11).

Functional abstraction allows us to implement these logic building blocks in a range of technologies – from electromagnetic relays to vacuum tubes to transistors. Of course, different types of gates may be easier or harder to make in any given technology. The set of AND, OR, and NOT gates are sufficient to

Fig. 2.8. Truth table for AND Gate.

A	B	A AND B
0	0	0
0	1	0
1	0	0
1	1	1

Fig. 2.9. Truth table for OR Gate.

A	B	A OR B
0	0	0
0	1	1
1	0	1
1	1	1

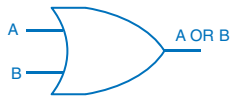


Fig. 2.10. The NOT Gate.

A	NOT A
0	1
1	0

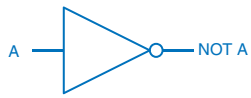
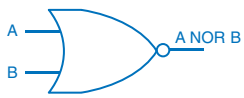


Fig. 2.11. The NOR Gate.

A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



construct any logical operation with any number of inputs and outputs. There are also other possible choices for such a "complete set" of gates.

Let's see how this all works by building a "bit-wise adder" black box from our basic AND, OR, and NOT building blocks. The bit-wise adder has two inputs – the bits A and B to be added – and two outputs – the result R and the carry C, if there is one (Fig. 2.12).

We need to combine AND, OR, and NOT gates appropriately to produce this logical behavior. First we note that the table for the carry bit is identical to that of the AND gate. We can write:

```
Carry C: A AND B
```

The table for the result R is almost the same as that for the OR gate except for the result for the 1 + 1 input when the output must be inverted. The result R is therefore A OR B unless A AND B is 1. After some thought we can write this as:

```
Result R: (A OR B) AND NOT (A AND B)
```

This leads to the implementation of the bit-wise adder shown in Figure 2.13.

This circuit is actually called a "half-adder" because although it correctly produces the carry value, it does not include a possible carry as an input to the device, as well as A and B. Using the half-adder of Figure 2.13, we can easily create an implementation for a "full adder" (Fig. 2.14). Adders for words with any number of bits can now be created by chaining these full adders together.

These examples illustrate two key principles of computer design. The first principle is that we rely on a "hierarchical design" process by which complex objects are built up from simpler objects. Logic gates are our fundamental and

A	B	R	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

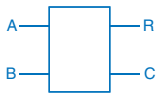


Fig. 2.12. Truth table for bit-wise adder.

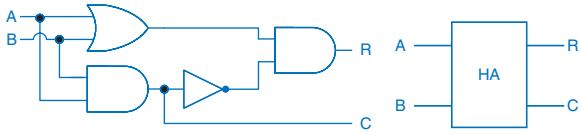


Fig. 2.13. A half-adder made by combining four logic gates.

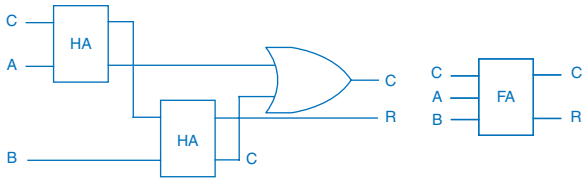


Fig. 2.14. A full adder made by composing two half-adders with an OR gate.

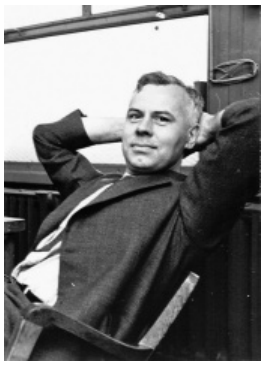
Fig. 2.15. The first binary adder, consisting of two battery cells, wires, two telephone relays, two light bulbs, pieces of wire, and a switch made from a tobacco tin.

universal building blocks. These objects can then be combined to build a bit-wise half-adder, which can then be used to build a full bit-wise adder, which then can be used to build whole-word adders and so on. The second principle is that of "functional abstraction." We have seen how the early computers used electromagnetic relays to implement logic gates and logical operations (B.2.5, Fig. 2.15).

The slow relays were soon replaced by the faster but unreliable vacuum tubes, which in turn gave way to transistors and now to integrated circuits in silicon. The important point is that the logical design of an adder is independent of its implementation. In his delightful book *The Pattern in the Stone*, the computer architect Danny Hillis shows how you can make mechanical implementations of logic gates (Fig. 2.16).

The memory hierarchy

In order to be able to do something useful, computers need a mechanism for storing numbers. A useful memory not only needs to store results of intermediate calculations on some sort of digital scratch pad, but also needs to be



B.2.5. George Stibitz (1904–95). As is often the case in science, some of Claude Shannon's ideas about relay circuits had been discovered independently around the same time. George Stibitz, a physicist at Bell Telephone Laboratories, was a member of a group of mathematicians whose job was to design relay-switching equipment for telephone exchanges. Stibitz also saw "the similarity between circuit paths through relays and the binary notation for numbers."[B2] Over one weekend in 1937, Stibitz wired up some relays to give the binary digits for the sum of two one-digit binary numbers. His output was two lightbulbs, which lit up according to the result of the binary addition. He then designed more complicated circuits that could subtract, multiply, and divide. With a Bell Labs engineer named Samuel Williams, Stibitz went on to build a machine with about four hundred relays that could handle arithmetic on complex numbers.

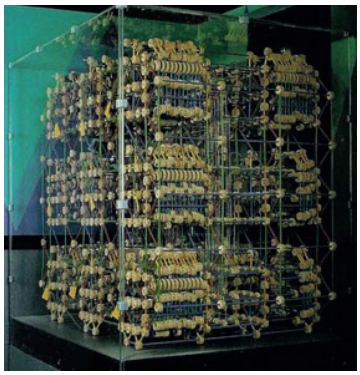


Fig. 2.16. We think of computers in terms of today's technologies, which use electronics and integrated circuits. Computers can be constructed from mechanical devices. This machine was built by a group of students at MIT using Tinker Toy spools and fishing line and can play tic-tac-toe.

Fig. 2.17. Pigeonholes are a useful analogy for a computer's memory.

able to alter what we have stored. We can think of computer memory as being like an array of mailboxes or pigeonholes (Fig. 2.17). Each box can store a data value that can either be retrieved or replaced with a new value. Memory registers are just a set of electronic boxes that can store a pattern of bits. As with logic gates, a wide variety of technologies have been used to implement computer memory. From an abstract point of view, the specific technology does not matter – in practice it matters a great deal, for reasons of reliability, speed of access, and cost!

In the logic-gate circuits discussed in the preceding text, the output states are completely determined by the inputs and the connections between the gates. Such a circuit is called a "combinational circuit." It is also possible to construct another type of circuit – a "sequential circuit" – for which the output of a device depends on the previous history of its inputs. A counter circuit is an example of a sequential device where the current count number stored is the sum of the number of pulses it has received. The elemental sequential digital circuit is designed to be stable in one of two states. These "bistable elements" are usually called "flip-flops" – since they flip between the two states in response to an input. The basic flip-flop circuit is important because it is used as a memory cell to store the state of a bit. Register memories are constructed by connecting a series of flip-flops in a row and are typically used for the intermediate storage needed during arithmetic operations. Another type of sequential circuit is an oscillator or clock that changes state at regular time intervals. Clocks are needed to synchronize the change of state of flip-flop circuits.

The simplest bistable circuit is the set-reset or RS flip-flop (Fig. 2.18). The state of the flip-flop is marked as Q and is interpreted as the state 1 if the voltage at Q is high or as 0 if the voltage is low. The complement of Q, $\overline{Q}$, is also available as a second output. There are also two terminals that allow the flip-flop to be initialized. The state Q can be set to 1 by applying a voltage pulse on the "set" input S. A signal on the "reset" input R resets Q to 0. Figure 2.18 shows an RS flip-flop made out of NOR gates together with the corresponding truth tables. An input signal S = 1 sets $\overline{Q}$ = 0 and if the input R = 0 then both inputs to the top NOR gate are zero. Thus a signal on the set line S and no signal on R gives Q = 1. This makes both inputs to the lower NOR gate 1. The other elements of the truth table can be filled in by similar reasoning. Note that an input state with both R = 1 and S = 1 is logically inconsistent and must be avoided in the operation of the flip-flop. So far, this RS flip-flop is still a combinational circuit because the state Q depends only on the inputs to R and S. We can make this a sequential RS flip-flop by adding a clock signal and a couple of additional gates (Fig. 2.19). The response of the clocked RS flip-flop at time t+1, Q(t+1), now depends on the inputs and the state of the flip-flop at time t, Q(t). A clock pulse must be present for the flip-flop to respond to its input states. There are many other different types of flip-flops and it is these bistable devices that are connected together to make registers, counters, and other sequential logic circuits.

Nowadays computers make use of a whole hierarchy of memory storage, built from a variety of technologies (Fig. 2.20). The earliest machines had only registers for storing intermediate results, but it soon became apparent that computers needed an additional quantity of memory that was less intimately linked to the central processing unit (CPU). This additional memory is called

S	R	Q	$\bar{Q}$
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	*	*

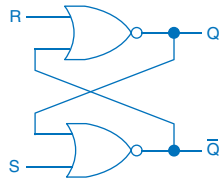


Fig. 2.18. Truth table and schema of a flip-flop circuit built from NOR gates. Inputs S = 1 and R = 1 are not allowed, the outputs marked by * are indeterminate.

(a)

R	S	Q	Q(t+1)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	*
1	1	1	*

(b)

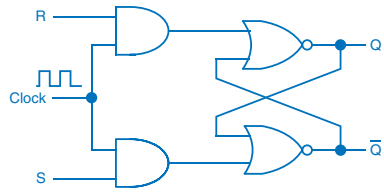


Fig. 2.19. Truth table and schema of clocked RS flip-flop.

the "main memory" of the computer and can be used to store results from registers as well as storing data needed by the registers for the different stages of the calculation. Between the registers and main memory, modern computers now incorporate several levels of memory that can be accessed more quickly than the main memory. These fast-access levels constitute the "cache memory" that is used to store the most frequently used data in order to avoid time delays that would be incurred in retrieving data from the slower memory.

Finally, since main memory is expensive, computer engineers introduced "secondary memory." This uses cheaper and slower technologies but allows data to be transferred to the main memory as and when required. Initially, data for this secondary memory was recorded on punched cards or paper tape and fed in manually to the machine by computer operators. The use of cards for secondary memory was superseded by magnetic tapes rather than much more expensive magnetic core memory. Magnetic tapes holding computer data became so common that many movies showing computers in operation showed images of spinning tape drives as an icon for a computer. Much clever engineering has been devoted to making tape drives extremely reliable and fast. However, there is one problem that no amount of engineering can solve:

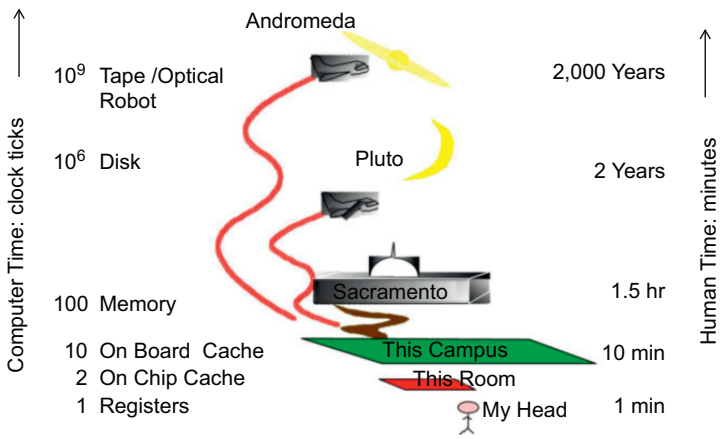


Fig. 2.20. How long does it take to get the data? This figure shows an analogy suggested by Jim Gray to illustrate the different data access times and the importance of memory hierarchy in computers. On the left, the access time is given in CPU clock ticks. For a typical 1 gigahertz clock this is one nanosecond. To relate these times to human timescales, on the right we translate a clock tick to one minute. The drawing in the middle illustrates how far we could have traveled during the time to retrieve data from the different elements of the computer memory hierarchy.

magnetic tapes are fundamentally sequential in the way they store and access data on the tape. Accessing some data stored halfway along the tape requires running through all the other data on the tape until that point is reached. This is fine if we only want to read long streams of information, but it is not very efficient if we want to access small amounts of nonsequentially stored bits of information. For this reason, disks and solid state semiconductor memory technologies that allow "random access" directly to any piece of data stored on the device are usually preferred, with the use of magnetic tapes restricted to providing archival backup storage for large data sets.

The fetch-execute cycle

We have now seen that computer hardware consists of many different components, which can all be implemented in a variety of ways. How do we coordinate and orchestrate the work of all these devices? The word *orchestrate* is an appropriate analogy. In an orchestra, there is a conductor who makes sure that everybody plays at the right time and in the right order. In the computer, an electronic clock performs the function of the conductor. Without the presence of a clock signal, memory circuits will not operate reliably. The clock signal also determines when the logic gates are allowed to switch.

In his draft report on the EDVAC, another key idea that von Neumann introduced was the "fetch-execute cycle." This relies on an electronic clock to generate a heartbeat that drives the computer through its series of operations. For simplicity and reliability, von Neumann chose the simplest possible control cycle to be coordinated by the central control unit:

- Fetch the next instruction from memory and bring it to the control unit;
- Execute the instruction using data retrieved from memory or already present;
- Send results back for storage in memory; and
- Repeat the fetch-execute cycle.

Von Neumann chose the approach of just getting one instruction at a time because he feared that any other approach would make the computer too hard to build and program reliably. Alan Perlis, one of the early programming pioneers, once said, "sometimes I think the only universal in the computing field is the fetch-execute cycle."[8]

The processor or CPU is the place where the instructions are executed and the data are manipulated. The main functions of the processor are to fetch the instructions from the main memory, decode the instructions, fetch the data on which the instruction's mathematical or logical operation will be performed, execute the instructions, and store the results. These main functions have not changed since the early processor designs. On a logical level, a simple processor (see Fig. 2.21) consists of a bank of registers, an arithmetical logical unit (ALU), and control unit (CU).

The CU fetches instructions from the memory, decodes them and generates the sequence of control signals that are required for completing the instructions. The ALU performs the arithmetical and logical operations. For an execution of each instruction, various components need to be connected by switches

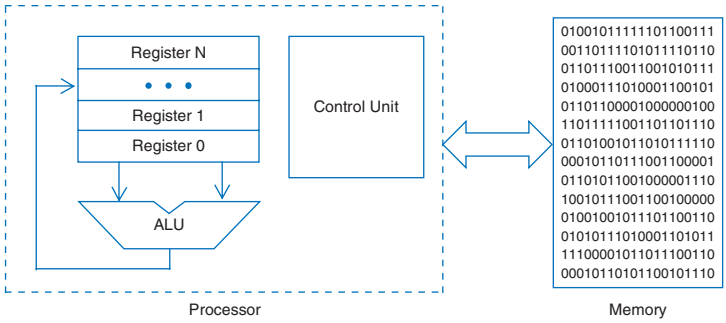


Fig. 2.21. A programmer's view of a processor.

that set the paths directing the flow of electrons. The bank of registers is for storing the instructions and the intermediate results of operations.

The exact choice of instructions that the hardware is built to execute defines the hardware-software interface. This is the "instruction set" of the machine. In the next chapter we will go up the hierarchy from the hardware layers and examine the software of the machine.

Key concepts

- Hierarchical design and functional abstraction
- Boolean algebra and switching circuits
- Binary arithmetic
- Bits, bytes, and words
- Logic gates and truth tables
- Combinational and sequential logic circuits
- Flip-flops and clocks
- The memory hierarchy
- The fetch-execute cycle

Some early history

The Manchester Baby and the Cambridge EDSAC computers

In his draft report on the design for the EDVAC, von Neumann analyzed the technical options for implementing the memory of the machine at length and concluded that the ability to construct large memories was likely to be a critical limiting factor. The first "stored program" computer to actually run a program was the University of Manchester's "Baby" computer. This was a cut-down design of their more ambitious "Mark 1" and was built primarily to test the idea of hardware architect, Freddie Williams, to use a cathode ray tube – like the screens of early televisions – as a device for computer memory. In June 1948, the Baby ran a program written by co-architect Tom Kilburn to find the highest factor of 2^{18} (Fig. 2.22). The program was just a sequence of binary numbers that formed the instructions for the computer to execute. The output appeared on the cathode ray tube and, as Williams recounted, it took some time to make the system work:

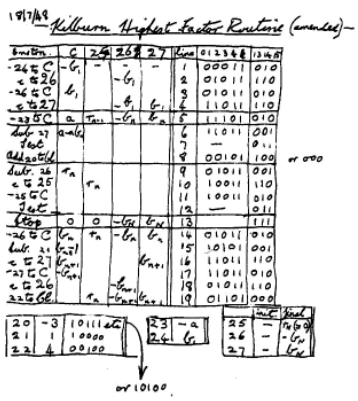


Fig. 2.22. Kilburn's highest factor routine from July 1948. The program ran for fifty-two minutes, and executed about 2.1 million instructions and made more than 3.5 million memory accesses.

> When first built, a program was laboriously inserted and the start switch pressed. Immediately the spot on the display tube entered a mad dance. In early trials it was a dance of death leading to no useful result and, what was even worse, without yielding any clue as to what was wrong. But one day it stopped and there, shining brightly in the expected place, was the expected answer.[9]

The success of the Manchester Baby experiment led to the construction of the full-scale Manchester Mark 1 machine. This became the prototype for the Ferranti Mark I, the world's first "commercially available general-purpose computer"[10] in February 1951, just a month before Eckert and Mauchly delivered their first UNIVAC computer in the United States.

B.2.6. Maurice Wilkes (1913–2010) seen here checking a mercury delay-line memory. He was a major figure in the history of British computing and at the University of Cambridge he led the team that designed and built the first fully operational stored-program computer.

While the Manchester Baby showed that a stored program computer was feasible, it was the EDSAC, built by Maurice Wilkes (B.2.6) and his team in Cambridge that was really the "first complete and fully operational regular electronic digital stored-program computer."[11] The computer used mercury delay-lines for storage as reflected in Wilkes's choice of the name EDSAC – Electronic Delay-Storage Automatic Calculator. The development of a suitable computer memory technology was one of the major problems for the early computer designers. It was mainly because of storage difficulties that EDVAC-inspired computers in the United States were delayed and lagged behind the Manchester and Cambridge developments. Wilkes chose to use mercury delay-lines for EDSAC because he knew that such delay-lines had played an important role in the development of radar systems during the war. Wilkes had a working prototype by February 1947, just six months after he had attended the Moore School Lectures. Working with a very limited budget had forced Wilkes to make some compromises in the design: "There was to be no attempt to fully exploit the technology. Provided it would run and do programs that was enough."[12]

Wilkes said later:

> It resembled the EDVAC in that it was a serial machine using mercury tanks, but that was all. When I was at the Moore School the EDVAC design did not exist, except maybe in Eckert's head.[13]

His visit to the Moore School had had a huge impact on Wilkes and he left with an enduring respect for Ekert and Mauchly and said "They remain my idols."[14] He also spent time with John Mauchly after the lectures and acknowledged this action as a "wonderful, wonderful piece of generosity."[15]

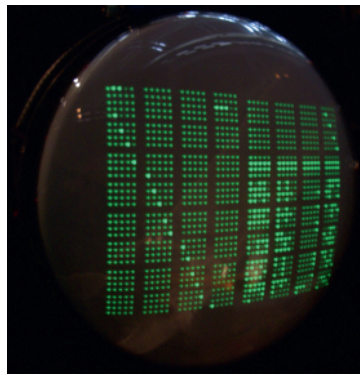


Fig. 2.23. A bit pattern on a cathode ray tube used in "Williams" memory.

Computer memory technologies

Williams and Kilburn and their team designing the Manchester Baby developed an internal memory using cathode ray tubes, the same technology as was then used in radar screens and televisions. In these so-called Williams Tubes, the electron guns could make charge spots on the screen that corresponded to binary 1 and 0 values. Since the charge would dissipate after a fraction of a second, the screen needed to be refreshed in order to retain the record of the bits. The Baby had four Williams Tubes: one to provide storage for 32-by-32-bit words; a second to hold a 32-bit "register," in which the intermediate results of a calculation could be stored temporarily; and a third to hold the current program instruction and its address in memory. The fourth tube, without the storage electronics of the other three, was used as the output device and could display the bit pattern of any selected storage tube (Fig. 2.23). Williams Tubes were used as storage in the commercial version of the Baby, the Ferranti Mark 1, as well as in such U.S. computers as the Princeton IAS and the IBM 701.

B.2.7. Austrian-born astrophysicist Tommy Gold (1920–2004) did important work in many scientific and academic fields. His knowledge of mercury delay-line storage from his experience with radar during the war was very helpful to Wilkes. Gold made major contributions to astronomy and cosmology as well as to radar technology. Gold was first to propose the now generally accepted theory that pulsars are rotating neutron stars.

Another early memory storage technology, originally suggested by Eckert, was based on the idea of a mercury delay-line – a thin tube filled with mercury that stores electronic pulses in much the same way as a hiker in a canyon can "store" an echo. A pulse represented binary 1; no pulse, binary 0. The pulses bounced from end to end and could be created, detected, and reenergized by electronic components attached to the tube. The EDSAC, built by Wilkes and his team in Cambridge, U.K., used mercury delay-lines. Wilkes had the good fortune to recruit a remarkable research physicist at Cambridge, Tommy Gold (B.2.7), who had been working on mercury delay-lines for radar during the war. Gold's knowledge and experience were invaluable in constructing large, five-foot "tanks" of mercury that were long enough to store enough pulses but were also precision engineered to an accuracy of a thousandth of an inch. Each mercury-filled tube could store 576 binary digits and the main store consisted of thirty-two such tubes, with additional tubes acting as central registers within the processor.

Mercury delay-lines and Williams Tubes were widely used until the early 1950s, when Jay Forrester (B.2.8), working at MIT, invented the magnetic core memory (Fig. 2.24). This device consisted of small

magnetizable rings located at the intersections of a grid of wires. Magnetization in the north direction could represent binary 1, and this could be flipped to a south magnetization, representing binary 0, by changing the current in the wire. By having a rectangular grid of wires and locating the cores at the intersections, it was possible to access each core individually. This allowed genuine random access of the individual memory locations – as opposed to having to go through the bits in sequence to get to the desired bit, as would be the case if the bits are stored on a magnetic tape. Forrester's technology was first tested in the construction of the Memory Test Computer at MIT in 1952. Compared to memory based on Williams Tubes, magnetic core memory proved much faster and far more reliable.

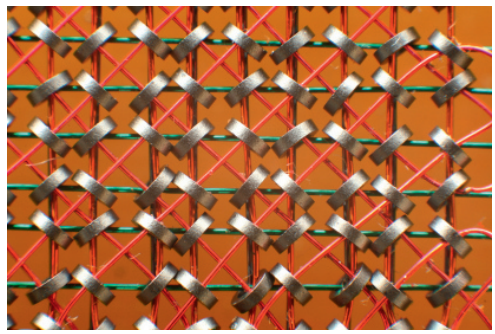


Fig. 2.24. Forrester's Magnetic Core Memory.

The first device to provide almost random access to data was not Forrester's magnetic core memory. It was a spinning drum with a magnetizable surface that allowed fast access to information stored in magnetized bands on the drum. This was invented in 1948 by Andrew Booth of Birkbeck College in England (B.2.9). Booth had made a visit to Princeton and had seen the progress von Neumann's team was making toward building the IAS stored program computer. Booth's prototype magnetic drum device was only two inches in diameter and could store ten bits per inch (Fig. 2.25). He followed up this prototype with larger drums that featured thirty-two magnetized bands, each divided into thirty-two words of thirty-two bits. A read/write head simply read off the values as the drum spun. Booth's drum memory was soon taken up by others and was adopted as secondary memory for Williams and Kilburn's scaled up version of the Baby, the Manchester Mark 1 machine. Magnetic drum memory was widely used for secondary memory in the 1950s and 1960s, until it was gradually superseded by magnetic disks.

B.2.8. Jay Forrester holding a frame of core memory from the Whirlwind computer. He invented magnetic core memory while working at MIT in the early 1950s. His invention proved much faster and more reliable than the earlier Williams Tubes or delay-line memory technologies.

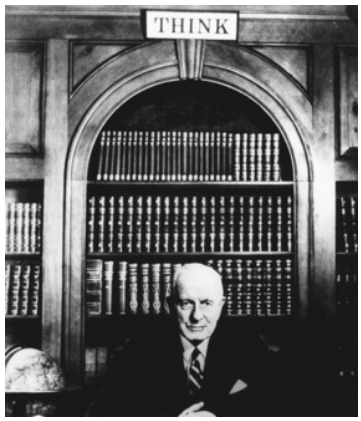
The first hard disk was introduced by IBM in 1956 and hard disks soon became ubiquitous. Hard disk drives consist of a number of flat circular disks, called platters, mounted on a spindle (Fig. 2.26). The platters are coated with a thin film of magnetic material, and changes in the direction of magnetization record the required pattern of binary digits. The engineering of these drives is impressive. The platters can rotate at

Fig. 2.25. Andrew Booth's magnetic drum memory.

B.2.9. Andrew Booth (1918–2009), together with his assistant and future wife, Kathleen Britten, developed magnetic drum storage. He also invented a fast multiplication algorithm that is used in modern Intel microprocessors.

Fig. 2.26. Hard disk drive from IBM.

speeds of up to 15,000 rpm and the read and write heads operate within tens of nanometers of the surface. Originally introduced for IBM's mainframe computers, hard disks are now small enough to be incorporated in PCs, laptops, and even iPods. The company referred to hard disk drives as "direct access storage devices" or DASDs – rather than use the term *computer memory*. This was reportedly because Tom Watson Sr., the legendary first head of IBM, feared that an anthropomorphic term like *memory* might exacerbate people's fear and distrust of computers (B.2.10).



B.2.10. Tom Watson Sr. (1874–1956) had built IBM to be the dominant company in punched card tabulating machines that offered businesses and governments the ability to process huge amounts of data. IBM was also known for its highly effective salesmen dressed in ties and dark suits; for the company motto "THINK"; and the prohibition of any alcohol on IBM property. Watson is often credited with saying "I think there is a world market for maybe five computers," but there is no evidence that he actually said this. It was his son, Tom Watson Jr. (1914–93), who drove the company's move into electronic computers.