I would now like to take a look at a subject which is extremely interesting, but almost entirely academic in nature. This is the subject of the energetics of computing. We want to address the question: how much energy must be used in carrying out a computation? This doesn’t sound all that academic. After all, a feature of most modern machines is that their energy consumption when they run very fast is quite considerable, and one of the limitations of the fastest machines is the speed at which we can drain off the heat generated in their components, such as transistors, during operation. The reason I have described our subject as "academic" is because we are actually going to ask another of our fundamental questions: what is the minimum energy required to carry out a computation?

To introduce these more physical aspects of our subject I will return to the field covered in the last chapter, namely the theory of information. It is possible to treat this subject from a strictly physical viewpoint, and it is this that will make the link with the energy of computation.

5.1: The Physics of Information

To begin with, I would like to try to give you an understanding of the physical definition of the information content of a message. That physics should get involved in this area is hardly surprising. Remember, Shannon was initially interested in sending messages down real wires, and we cannot send messages of any kind without some interference from the physical world. I am going to illustrate things by concentrating on a particular, very basic physical model of a message being sent.

I want you to visualize the message coming in as a sequence of boxes; each of which contains a single atom. In each box the atom can be in one of two places, on the left or the right side. If it’s on the left, that counts as a 0 bit, if it’s on the right, it’s a 1. So the stream of boxes comes past me, and by looking to see where each atom is I can work out the corresponding bit (Fig. 5.1):
To see how this model can help us understand information, we have to look at the physics of jiggling atoms around. This requires us to consider the physics of gases, so I will begin by taking a few things I need from that. Let us begin by supposing we have a gas, containing $N$ atoms (or molecules), occupying a volume $V_1$. We will take this gas to be an exceptionally simple one; each atom, or molecule, within it (we take the terms to be interchangeable here) is essentially free — there are no forces of attraction or repulsion between each constituent (this is actually a good approximation at moderately low pressures). I am now going to shrink the gas, pushing against its volume with a piston, compressing it to volume $V_2$. I do all this isothermally: that is, I immerse the whole system in a thermal "bath" at a fixed temperature $T$, so that the temperature of my apparatus remains constant. Isn’t it wonderful that this has anything to do with what we’re talking about? I’m going to show you how. First we want to know how much work, $W$, it takes to compress the gas (see Fig. 5.2):

![Fig. 5.2 Gas Compression](image)

Now a standard result in mechanics has it that if a force $F$ moves through a
small distance $\delta x$, the work\textsuperscript{1} done $\delta W$ is:

$$\delta W = F \delta x$$  \hspace{1cm} (5.1)

If the pressure of the gas is $p$, and the cross-sectional area of the piston is $A$, we can rewrite this using $F = pA$ and letting the volume change of the gas $\delta V = A \delta x$ so that:

$$\delta W = p \delta V.$$ \hspace{1cm} (5.2)

Now we draw on a standard result from gas theory. For an ideal gas at pressure $p$, volume $V$ and temperature $T$, we have the relation:

$$pV = NkT$$ \hspace{1cm} (5.3)

where $N$ is the number of molecules in the gas and $k$ is Boltzmann’s constant (approximately $1.381 \times 10^{-23}$ J K$^{-1}$). As $T$ is constant — our isothermal assumption — we can perform a simple integration to find $W$:

$$W = \int_{V_1}^{V_2} \frac{NkT}{V} \, dV = NkT \log \frac{V_2}{V_1}.$$ \hspace{1cm} (5.4)

(Here, $\log x = \log_e x$.) Since $V_2$ is smaller than $V_1$, this quantity is negative, and this is just a result of the convention that work done on a gas, rather than by it, has a minus sign. Now, ordinarily when we compress a gas, we heat it up. This is a result of its constituent atoms speeding up and gaining kinetic energy. However, in our case, if we examine the molecules of the gas before and after compression, we find no difference. There are the same number, and they are jiggling about no more or less energetically than they were before. There is no difference between the two at the molecular level. So where did the work go? We put some in to compress the gas, and conservation of energy says it had to go somewhere. In fact, it was converted into internal gas heat, but was promptly

\textsuperscript{1}Another one of those awkward words, like "information". Note that, with this definition, a force must move through a distance to perform work; so it does not take any of this kind of "work" to hold up a suitcase — only to lift it! [RPF]
drained off into the thermal bath, keeping the gas at the same temperature. This is actually what we mean by isothermal compression: we do the compression slowly, ensuring that at all times the gas and the surrounding bath are in thermal equilibrium.

From the viewpoint of thermodynamics, what we have effected is a "change of state", from a gas occupying volume $V_1$ to one occupying volume $V_2$. In the process, the total energy of the gas, $U$, which is the sum of the energies of its constituent parts, remains unchanged. The natural thermodynamical quantities with which such changes of state are discussed are the free energy $F$ and the entropy $S$, which are related by:

$$ F = U - TS. $$ (5.5)

The concept of free energy was invented to enable us to discuss the differences between two states even though there might be no actual mechanical differences between them. To get a better feel for its meaning, look at how expression (5.5) relates small variations at constant temperature:

$$ \delta F = \delta U - T \delta S. $$ (5.6)

For the change under consideration, the total gas energy remains constant, so $\delta U=0$ and $\delta F = -T \delta S$. $\delta F$ is just the "missing" heat energy siphoned off into the heat bath, $NkT \log(V_1/V_2)$, and we use this to write (5.6) as an entropy change:

$$ \Delta S = Nk \log \frac{V_2}{V_1}. $$ (5.7)

Note that as we are dealing with a finite change here, we have replaced the infinitesimal $\delta$ with a finite $\Delta$.

Entropy is a rather bizarre and counter-intuitive quantity, and I am never sure whether to focus on it or on the free energy! For those who know a little thermodynamics, the general equation $\delta S = -\delta F/T$ is a variant of the standard formula $\delta S = \delta Q/T$ for the infinitesimal change in entropy resulting from a thermodynamically reversible change of state where, at each stage, an amount
of heat $\delta Q$ enters or leaves the system at absolute temperature $T$. For an irreversible process, the equality is replaced by an inequality, ensuring that the entropy of an isolated system can only remain constant or increase — this is the Second Law of Thermodynamics. I'll say a little more about entropy in a moment.

Now we take a bit of a leap, and it is not obvious that we can do this, but we can. We consider the case where our gas contains only one molecule. That is, we put $N=1$ into our formulae. Now it's difficult to get a feeling for concepts like temperature, pressure and volume, never mind free energy and entropy, when you only have one molecule! However, these concepts make sense as long as we consider them to be time averaged, smoothing out the irregularities of this one particle as it bounces back and forth. Indeed, our formulae actually work with $N=1$, as long as there is this hidden smoothing. The situation is more fun, too!

Let us suppose that we are halving the volume occupied by the molecule: $V_2 = V_1/2$. We then find that the free energy and the entropy of the particle change by:

$$+ kT \log 2 \text{ and } -k \log 2$$

(5.8)

respectively. What does this mean? Pictorially, the situation has changed from:

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>V_1</td>
</tr>
</tbody>
</table>

_to:

<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>V_2</td>
</tr>
</tbody>
</table>

\[ V_2 = V_1 / 2 \]

The physical state of the molecule before and after the compression appears to
be the same — its actual (kinetic) energy has not changed, for example — yet for some reason we have a change in these quantities $F$ and $S$. What has happened, and this is very subtle, is that my  \textit{knowledge of the possible locations of the molecule has changed}. In the initial state, it could be hiding anywhere in volume $V_1$; after the compression, it must be somewhere within $V_2$. In other words, there are fewer places it can be in.

This concept of "knowledge" is extremely important, and central to the concept of entropy, so I will dwell on it awhile. It arises from the deeply statistical nature of thermodynamics. When doing the mathematics of vast numbers of particles that make up gases, we cannot practically follow the paths and momenta of every molecule in the gas, so we are forced to turn to probability theory. Concepts such as temperature and pressure of a gas are essentially defined to be statistical averages. We assign certain physical properties to each molecule, assume particular distributions for these molecules, and calculate the average by a weighting process: so many molecules will move this fast, corresponding to one temperature; so many will move that fast, giving another temperature; and we just average over everything. The entropy of a gas is defined statistically, indeed this is its core definition, but in a different way to quantities such as temperature and energy. Unlike these, it is not a macroscopic property that arises from a sum of microscopic properties. Rather, it is directly related to the \textit{probability that the gas be in the configuration in which it is found}. By "configuration" I mean a particular arrangement, or cluster of arrangements, of positions and momenta for each of the $N$ constituent molecules (or, if you want to be fancy, a particular point or region in "phase space"). The existence of such a probability should not come as too much of a surprise: if you look at any given gas it is far less likely at the outset that you will find all the molecules moving in the same direction or paired up and dancing than you will find them shooting all over the place at random. Entropy quantifies this notion. Loosely speaking, if the probability of a particular gas configuration is $W$, we have:

$$S = k \log W.$$  \hspace{2cm} (5.9)

The bigger $W$, the bigger the entropy, and, like all probabilities, the $W$'s add, so we can straightforwardly calculate the chances of being in some range of configurations. The gas with molecules going all one way has a $W$ much less than that of the one with a more random — or more disordered — structure, and hence has a lower entropy. What has all this got to do with our knowledge of
a system? Simply, the less we know about the configuration of a gas, the more states it could be in, and the greater the overall $W$ — and the greater the entropy. This gives us a nice intuitive feel of what is happening when we compress a gas into a smaller volume. Working isothermally, the momenta of the molecules within the container remains the same ($\delta U = 0$), but each molecule has access to fewer possible spatial positions. The gas has therefore adopted a configuration with smaller $W$, and its entropy has decreased. As an aside, the Second Law of Thermodynamics tells us that in any isolated system:

$$\delta S = k \frac{\delta W}{W} \geq 0,$$

(5.10)

i.e. the entropy never decreases. The fact that the entropy of our compressed gas has dropped is a reminder that the system is not isolated — we have been draining heat into a heat bath. The heat flow into the bath increases its entropy, preserving the Second Law. Generally speaking, the less information we have about a state, the higher the entropy.

As the definition of entropy is essentially statistical, it is perfectly all right to define it for a gas with a single molecule, such as the one we have been considering, although there are a few subtleties (which we will avoid). You can see that if we compress the volume by a factor of 2, then we halve the number of spatial positions, and hence the number of configurations that the molecule can occupy. Before, it could be in either half of the box: now, it can only be in one half. You should be able to see in our probabilistic picture how this leads to a decrease in entropy by an amount:

$$\delta S = k \log 2$$

(5.11)

This is the same as we obtained with our work and free energy considerations.

We can now return to the topic of information and see where all this weird physics fits in. Recall the atomic tape with which we opened this section, in which the position of atoms in boxes tells us the binary bits in the message. Now if the message is a typical one, for some of these bits we will have no prior knowledge, whereas for others we will — either because we know them in advance, or because we can work them out from correlations with earlier bits that we have examined. We will define the information in the message to be proportional to the amount of free energy required to reset the entire tape to zero. By "reset to zero", we mean compress each cell of the tape to ensure that
its constituent atom is in the "zero" position.

Straightaway, we note what seems to be an obvious problem with this definition, namely, that it introduces an unnatural asymmetry between 0 and 1. If an atom is already in the zero part of the compartment, then surely the reset operation amounts to doing nothing, which costs no free energy. Yet if it is in the one position in the compartment, we have to do work to move it into the zero position! This doesn't seem to make sense. One would expect to be able to introduce an alternative definition of information for which the tape is reset to one — but then we would only seem to get the same answer if the message contained an equal number of ones and zeroes! But there is a subtlety here. Only if we do not know which side of the compartment the atom is in do we expend free energy. It is only in this circumstance that the phase space for the atom is halved, and the entropy increases. If we know the atom's position, then we expend no energy in resetting, irrespective of where the atom starts out. In other words, as one would hope, the information in the message is contained in the surprise bits. Understanding why this is so is worth dwelling on, as it involves a style of argument often seen in the reversible computing world. It seems a bit counter-intuitive to claim that the energy required to reset a one to a zero is no more than leaving a zero alone — in other words, nothing.

To clear this point up, I first have to stress the idealized nature of the set-up we are considering. Although I have talked freely about atoms in "boxes", these boxes are not real boxes made of cardboard and strung together, with mass and kinetic and potential energy. Moreover, when I talk about "energy", I certainly don't mean that of the tape! We are only interested in the content of the message, which is specified by the positions of the atoms. Let us suppose we have a message bit that we know is a one — the atom is on the right hand side — so we have the following picture:

![Diagram of a box with a dot inside]

We can show that to reset this to zero costs no energy in several ways. One pretty abstract way is to first slip in a little partition to keep the atom in place. All I have to do now is turn the box over. The end result is that we now have a zero on the right hand side (Fig. 5.3):
This is abstract because it might seem odd to be able to insert pistons and turn boxes without expending energy. In the real world, of course, you can't — but we are dealing with abstractions here and, as I have said, we are not interested in the kinetic energy or weight of the "boxes". Given our assumptions, it is possible to do so, although the downside is that we would have to take an eternity to do it! (We will return to this sort of argument in §5.2.) Another way, perhaps a little less abstract, would be to introduce two pistons, one on each side of the box, and push the atom over with one, while drawing the other out (Fig. 5.4):

Now the bombardment on the left is equal to that on the right, and any work put in at one end will be taken out at the other, and so is recovered. One could even join the pistons by an up-and-over rod, and you should be able to see that the tiniest touch on one piston will send the whole thing coasting over to its final position. So, if you do it slowly enough — "infinitesimal in the limit" — no work is done in resetting. Clearing, or resetting, the tape is what occurs when we don't know what compartment the atom is in. Then we must perform a compression, and this will take free energy, as we discussed earlier, as we are
lessening our ignorance of the atom's position.

Another way of looking at these ideas is due to Bennett, who suggests using a message tape as *fuel*, and relates the information in the tape to its fuel value — that is, to the amount of energy we can get from it. His idea, which is quite subtle, goes as follows. We suppose we have a machine, in contact with some kind of heat bath, which takes in tapes at one end, and spits them out at the other. We assume to begin with, that the tape the machine eats is blank, i.e. all of its atoms are in the zero state. We will show how such a tape can be used to provide us with useful work, which we can use to power our machine.

What we do is incorporate a piston into the system. As each cell comes in, we bring the piston into it, up to the halfway position in each box (Fig. 5.5):

![Fig. 5.5 An Information-driven Engine](image)

We now let the heat bath warm the cell up. This will cause the atom in the cell to jiggle against the piston, isothermally pushing it outwards as in Figure 5.6:

![Fig. 5.6 Work Generation Mechanism in the Engine](image)
This is just the opposite process to the compression of a gas we considered at the beginning of this section. The net result is that work is done on the piston which we can subsequently extract: in other words, we can get our tape to do work for us. You should be able to see that for a tape of \( n \) bits this work is equal to \( nkT \log_2 \), the free energy, where \( T \) is the temperature of the heat bath. An important consequence of our procedure is that the tape that the machine spits out has been \textit{randomized}: after the piston has been pushed out, the atom that did the pushing can be anywhere in that cell, and we have no way of knowing where, short of performing a measurement.

We now generalize the argument by assuming that our piston is maneuverable. This allows us to extract work from tapes which have a 1 in them. If we get a 1, we switch the piston to the other side of the cell, bring it up to the edge of the 1 half, and proceed as before. Again we get \( kT \log_2 \) of useful work given out, and again the tape that emerges from the machine is randomized. What is crucial here is that we \textit{know} what bit is about to enter the machine. Only then can we ready the piston to ensure that it does work for us. Obviously, if we left the piston in the 0 position, and we got a 1 in, we would actually have to do work to shift the atom into the 0 cell, and when the atom expands back into the full cell we would get that work back: that is, no useful work would be done. Clearly, a \textit{random tape has zero fuel value}. If we do not know what bit is coming in next, we do not know how to set our piston. So we would leave it in one position, and just push it in and hope, push it in and hope, boom, boom, boom. Sometimes we would get lucky, and find an atom pushing our piston out again, giving us work; but equally likely, for a truly random message, we have to do work on the atom. The net result is zero work to power our machine.

Clearly, Bennett’s tape machine seems to do the opposite to our reset process. He uses a message tape to extract work, ending up with a random tape: we took a random tape and did work on it, to end up with a tape of standard zeroes. This inverse relationship is reflected in the definition of information within Bennett’s framework. Suppose we have a tape with \( N \) bits. We define the information, \( I \), in the tape by the formula:

\[
\text{Fuel value of tape} = (N-I)kT \log 2.
\]  \hspace{1cm} (5.12)

From this we see that a tape giving us a full fuel-load — that is, \( kT \log 2 \) per bit — carries zero information. This is what we would expect since such a tape must have completely predictable contents. There is a nice physical symmetry
between these two approaches. If we run a message tape through the machine, we will be able to extract a certain energy \( E \) from it: this energy \( E \) will be precisely what we need to reset the newly randomized tape to its original form. It is, of course, up to you which picture you prefer to adopt when thinking about these things. I opt for the erasure picture partly because I do not like having to subtract from \( N \) all the time to get my information!

You might like to contemplate some problems on Dr. Bennett's machine.

**Problem 5.1:** Suppose we have two tapes: an \( N \)-bit random tape, and an exact copy. It can be shown that the fuel value of the two tapes combined is \( N k T \log 2 \). See if you can design a machine that will be able to extract this amount of energy from the two tapes. (Hint: you have to expand one tape "relative" to the other.)

**Problem 5.2:** We have a tape in which three bits are repeated in succession, say 110110110110... For a \( 3N \)-bit tape, what is the fuel value? How do you get it out?

### 5.1.1: Maxwell's Demon and the Thermodynamics of Measurement

Those of you who wish to take your study of the physics of information further could do no better than check out many of the references to a nineteenth century paradox discovered by the great Scottish physicist James Clerk Maxwell. *Maxwell's Demon*, as it is known, resulted in a controversy that raged among physicists for a century, and the matter has only recently been resolved. In fact, it was contemplation of Maxwell's demon that partly led workers such as Charles Bennett and Rolf Landauer to their conclusions about reversible computing, the energy of computation, and clarified the link between information and entropy. Importantly, such research has also shed light on the role of *measurement* in all this. I will not go into the matter in great detail here, but supply you with enough tidbits to at least arouse your interest. A full discussion of the demon and of the attempts to understand it can be found in *Maxwell's Demon: Entropy, Information, Computing*, by H.S. Leff and A.F. Rex (Adam Hilger, 1990).

With Maxwell, we will imagine that we have a small demon sitting on a partitioned box, each half of which is filled by a gas of molecules with a random distribution of positions and velocities (Fig. 5.7):
The demon has a very simple task. Set into the partition is a flap, which he can open and shut at will. He looks in one half of the box (say, the left) and waits until he sees a fast-moving molecule approaching the flap. When he does, he opens the flap momentarily, letting the molecule through into the right side, and then shuts the flap again. Similarly, if the demon sees a slow-moving molecule approaching from the right side of the flap, he lets that through into the side the fast one came from. After a period of such activity, our little friend will have separated the fast- and slow-moving molecules into the two compartments. In other words, he will have separated the hot from the cold, and hence created a temperature difference between the two sides of the box. This means that the entropy of the system has decreased, in clear violation of the Second Law!

This seeming paradox, as I have said, caused tremendous controversy among physicists. The Second Law of Thermodynamics is a well-established principle in physics, and if Maxwell's demon appears to be able to violate it, there is probably something fishy about him. Since Maxwell came up with his idea in 1867, many people have tried to spot the flaw in his argument. Somehow, somewhere, in the process of looking for molecules of a given type and letting them through the flap, there had to be some entropy generated.

Until recently, it was generally accepted that this entropy arose as a result of the demon's measurement of the position of the molecules. This did not seem unreasonable. For example, one way in which the demon could detect fast-moving molecules would be to shine a demonic torch at them; but such a
process would involve dispersing at least one photon, which would cost energy. More generally, before looking at a particular molecule, the demon could not know whether it was moving left or right. Upon observing it, however this was done, his uncertainty, and hence entropy, would have reduced by half, surely accompanied by the corresponding generation of entropy in the environment.

In fact, and surprisingly, Bennett has shown that Maxwell's demon can actually make its measurements with zero energy expenditure, providing it follows certain rules for recording and erasing whatever information it obtains. The demon must be in a standard state of some kind before measurement, which we will call $S$: this is the state of uncertainty. After it measures the direction of motion of a molecule, it enters one of two other states — say $L$ for "left-moving", or $R$ for "right-moving". It overwrites the $S$ with whichever is appropriate. Bennett has demonstrated that this procedure can be performed for no energy cost. The cost comes in the next step, which is the erasure of the $L$ or $R$ to reset the demon in the $S$ state in preparation for the next measurement. This realization, that it is the erasure of information, and not measurement, that is the source of entropy generation in the computational process, was a major breakthrough in the study of reversible computation.

5.1.2: Energy and Shannon's Theorem

Before leaving physics and information, I would like to return to something we studied in the previous chapter, namely, the limits on sending information down a channel. It will come as no surprise to you that we can revisit Shannon's Theorem with our physical tools too! Let us combine our study of the physics of information with our earlier work on errors. An interesting question is: How does the occurrence of an error in a message affect its information content? Let's start off with a message with all its $M$ bits perfectly known, containing information $N$, and suppose that we want to send it somewhere. We're going to send it through a noisy channel: the effect of this is that, in transit, each bit of the message has a probability $q$ of coming through wrong. Let us ask a familiar question: what is the minimum number of bits we have to send to get the information in the $M$ bits across? We will have to code up the message, and in keeping with our earlier look at this question, we'll say the coded message has length $M_c$. This is the number of bits we actually send. Now we have said that to clear the tape, assuming we know nothing about its contents, we need to expend the following amount of free energy:


\[ M_C kT \log 2. \quad (5.13) \]

However, some of this energy is taken up in clearing errors. On average, using our earlier derivations, this amount will be:

\[ M_C kT \log 2 [-q \log_2 q - (1-q) \log_2 (1-q)] = [1 - f(q)] M_C kT \log 2. \quad (5.14) \]

This energy we consider to be wasted. This leaves us with the free energy:

\[ M_C kT \log 2 - [1 - f(q)] M_C kT \log 2 = f(q) M_C kT \log 2 \]  

(5.15)

to expend in clearing the message. By conservation of energy, then, and using our relationship between free energy and information, the greatest amount of information I can send through this channel will be:

\[ M_C [q \log_2 (1/q) + (1-q) \log_2 (1/(1-q))] \]

(5.16)

You can see how this kind of physical argument now leads us on to Shannon’s result.

5.2: Reversible Computation and the Thermodynamics of Computing

It has always been assumed that any computational step required energy\(^2\). The first guess, and one that was a common belief for years, was that there was a minimum amount of energy required for each logical step taken by a machine. From what we have looked at so far, you should be able to appreciate the argument. The idea is that every logical state of a device must correspond to some physical state of the device, and whenever the device had to choose between 0 and 1 for its output — such as a transistor in an AND gate — there would be a compression of the available phase-space of the object from two

options to one, halving the phase-space volume. Therefore, the argument went, a minimum free energy of $kT \log 2$ would be required per logical step\(^3\). There have been other suggestions. One focused on the reliability of the computational step. The probability of an error, say $q$, was involved and the minimum energy was supposed to be $kT \log q$. However, recently this question has been straightened out. The energy required per step is less than $kT \log q$, less than $kT \log 2$, in fact less than any other number you might want to set — provided you carry out the computation carefully and slowly enough. Ideally, the computation can actually be done with no minimal loss of energy. Perhaps a good analogy is with friction. In practice, there is always friction, and if you take a look at a typical real-world engine you will see heat energy dissipated all over the place as various moving parts rub against one another. This loss of energy is ordinarily large. However, physicists are very fond of studying certain types of idealized engines, so-called Carnot heat engines in which heat energy is converted into work and back again, for which it is possible to calculate a certain maximum efficiency of operation. Such engines operate over a reversible closed cycle: that is, they start off in a particular state and, after one cycle of operation, return to it. The Second Law ensures that this cannot be done for zero energy cost but it is theoretically possible to operate such machines in such a way as to achieve the maximum efficiency, making the losses due to friction, for example, as small as possible. Unfortunately, they have to be run infinitesimally slowly to do this! You might, for example, want to drain heat from the engine into a surrounding reservoir to keep everything at thermal equilibrium, but if you operate the machine too quickly you will not be able to do this smoothly and will lose heat to parts of the engine that will simply dissipate it. But the point is that, in principle, such engines could be made, and physicists have learned much about thermodynamics from studying them. The crucial requirement is reversibility. Now it turns out that a similar idea works in computers. If your computer is reversible, and I’ll say what I mean by that in a moment, then the energy loss could be made as small as you want, provided you work with care and slowly — as a rule, infinitesimally slowly. Just as with Carnot’s engines, if you work too fast, you will dissipate energy. Now you can see why I think of this as an academic subject. You might even think the question is a bit dopey — after all, as I’ve said, modern transistors dissipate something like $10^8 kT$ per switch — but as with our discussion of the limits of what is computable, such questions are of interest. When we come to design the Ultimate Computers of the far future, which might have "transistors" that are

\(^3\) This is actually a lower limit far beneath anything practically realizable at present. Conventional transistors dissipate on the order of $10^8 kT$ per step! [RPF]
atom-sized, we will want to know how the fundamental physical laws will limit us. When you get down to that sort of scale, you really have to ask about the energies involved in computation, and the answer is that there is no reason why you shouldn't operate below $kT$. We shall look later at problems of more immediacy, such as how to reduce the energy dissipation of modern computers, involving present-day transistors.

5.2.1: Reversible Computers

Let me return to the matter of "reversible computing". Consider the following special kind of computation, which we draw as a black box with a set of input and output lines (Fig. 5.8):

![Fig. 5.8 A Reversible Computation](image)

Suppose that for every input line there is one, and only one, output, and that this is determined by the input. (In the most trivial case, the signals simply propagate through the box unchanged.) In such circumstances, the output carries no more information than the input — if we know the input, we can calculate the output and, moreover, the computation is "reversible". This is in sharp contrast to a conventional logic gate, such as an AND (Fig. 5.9):

![Fig. 5.9 The AND Gate](image)

In this case we have two lines going in but only one coming out. If the output is found to be zero, then any one of three possible states could have led to it. I have irretrievably lost information about the input so the AND gate is irreversible. So too, is the OR gate (but not the NOT!). In other words, the phase space of the inputs has shrunk to that of the output, with an unavoidable
decrease in entropy. This must be compensated by heat generation somewhere. The mistake everyone was making about energy dissipation in computers was based on the assumption that logical steps were necessarily like AND and OR — irreversible. What Bennett and others showed was that this is not necessarily the case. The fact that there is no gain in information in our abstract "computation" above is the first clue that maybe there's no loss of entropy involved in a reversible computation. This is actually correct: reversible computers are rather like Carnot engines, where the reversible ones are the most efficient. It will turn out that the only entropy loss resulting from operating our abstract machine comes in resetting it for its next operation.

We can consider a "higher" kind of computer which is reversible in a more direct sense: it gives as its output the actual result of a computation plus the original input. That is, it appends the input data to the output data printed on its tape (say). This is the most direct way of making a computation reversible. We will later show that, in principle, such a calculation can be performed for zero energy cost. The only cost is incurred in resetting the machine to restart, and the nice thing is that this does not depend on the complexity of the computation itself but only on the number of bits in the answer. You might have billions of components whirring away in the machine, but if the answer you get out is just one bit, then $kT \log 2$ is all the energy you need to run things.

We actually studied some reversible gates earlier in the course. NOT is one, as I've said. A more complicated example we looked at was Fredkin's CONTROLLED CONTROLLED NOT gate (Fig. 5.10):

![CCN Gate Diagram]

in which the lines $A$ and $B$ act as control lines, leaving $C$ as it is unless both are one, in which case $C$ becomes NOT $C$. This is reversible in the sense that we
can regain our input data by running the output through another CCN gate (see section 2.3).

I would now like to take a look in more detail at some reversible computations and demonstrate the absence of a minimum energy requirement. I'll start with a computation that you might not ordinarily think of as a computation: the act of copying (recall our discussion of Turing copying machines, §3.5). This seems like a dumb sort of computation, as you’re not getting anywhere, but it is a useful introduction to some of the ideas underlying issues of energy dissipation. It's not at all obvious that you can copy information down from one place to another without expending at least some energy, even in principle. Having said this, it is easy to suggest why it might not cost any energy. We can consider a set of data and its copy as two messages on tape, both identical. Either we know what the original message is, or we don't. In the first case, no free energy is expended in clearing the tape, and none need be for the copy tape: we just turn it over when necessary, as we discussed previously. In the second case, clearing the tape will cost free energy, but not for the copy: knowing what the first tape says, we can use this information to clear the copy by turning bits over again. Simply, there is no more information in the (data plus copy) set than is in just the single data set. Clearing the system should not, therefore, require more free energy in the first case than the second. This is a common type of argument in the reversible computing world.

5.2.2: The Copy Computation

Let us make these ideas a little more concrete. In a moment, I will examine a copying machine found in Nature, namely the RNA molecule found in living cells. But first, I will take a look at two rather artificial examples of copying machines. Our discussion follows Bennett.

We start with a very general copy process. We will have an original object, which we'll call the model, which can somehow hold a zero or one. It's some kind of bistable physical device. We want another object, which we'll call the copier, which can also hold a zero or one. An example of a bistable device would be one which could be modeled by the following potential well (Fig. 5.11):
I will give one possible physical realization of this shortly. What this rather abstract diagram means is that some part of the device, which we will represent by a dot, can be in either of two stable states — here, in the left or the right trough, meaning one or zero, say. The curve displays the potential energy of the dot according to its position in the device. The troughs are the minima of this energy, and are favored by the dot: they are of equal depth, and are hence equally likely to be occupied at the outset. A useful way to think of this operation is to have the dot as a ball, and the curve an actual shape constraining it. Putting energy into the ball makes it move up and down the sides of its trough; enough energy and the ball will go over the hill and into the next trough — equivalent to our model changing its bit-state. The height of the hill, the amount of energy needed for the transition to occur, is called the barrier potential. In actual operation, we would want the typical thermal fluctuations of whatever it is the dot represents to be much less than this, to keep the device stable. Another way of visualizing this is to imagine the dot to be in a box separated into two halves by a partition. The barrier potential will be the energy required to get the dot from one half into the other.

We suppose both model and copier to be modeled by such a potential, and the model to be in some state. This can be random — we need not know what it is, but for sake of illustration let’s say it is as shown in Figure 5.12 (where we have used an $X$ for the model’s dot):

Fig. 5.12 Initial State of the Model
How does the copier start out? It must be in some standard state. It cannot be in a random state, because copying will involve getting it into a definite state, and to do this we must do work (compressing, if we use the box and partition analogy). Alternatively, you can use phase space considerations, comparing the number of possible model-copier options before copying (four, if the model is randomly set) and after (just two): this would be a logically irreversible step. Let’s say the copier starts out in the state opposite to the model (Fig. 5.13):

![Fig. 5.13 Initial State of the Copier](image)

Clearly, copying is going to involve somehow getting the dot from one trough to the other. To do this, we need to be able to manipulate the potential curve; we have to make the other trough energetically more favorable to the dot. We shall assume that there are two parameters associated with the copier that we can adjust: the barrier height, and the relative depths of the troughs. Furthermore, we assume that the depths of the troughs can be altered by some force of interaction between the copier and the model. (Don’t worry if this is all horribly confusing and abstract! All will become clear.) We’ll call this a "tilt" force, since it tilts the graph. We will combine these two operations to move the copier dot, but we will combine them in such a way — and this is important — that there will always be a unique minimum accessible to the dot at all times.

What we do is this. We start with the model some way away from the copier. Even at a distance it will exert a slight tilt force on the copier. We take this force to have the consequence of increasing the depth of whichever trough of the copier corresponds to that occupied in the model. The copier potential will hence be slightly distorted at the outset, as shown overleaf:
The first step in the copy process involves gently lowering the copier's potential barrier. This removes the obstacle to the dot switching positions: it can now wander over to the other bit state. What will make it do this? This is where the "tilt" from the model comes in. In step two, we slowly bring the model up closer to the copier, and in the process its tilt force increases. This gradually distorts the copier's potential even more, lowering the energy of the appropriate trough as shown in Figure 5.15:

The dot now slides smoothly down the potential curve, occupying the new, energetically more favorable trough. In step three, we replace the potential barrier to secure the dot in its new position, and finally, step four, we take the
model away, restoring the copier’s potential to its normal state (Fig. 5.16):

![Figure 5.16 Final State of the System](image)

That is the basic idea of this copy machine. It’s possible to play around with it further. For example, for appropriate physical systems, we can envisage bringing the model up to the copier in step one in such a way as that the tilt force lowers the state the dot is already in so that the dot is held steady while we lower the potential barrier, if this is a concern. The model is then moved over to the other side to provide the new tilt. This is one variation, but it does not significantly alter the basic idea.

The crucial thing about this process is that it needs to be carried out slowly and carefully. There are no jumps or sudden changes. The easiest way to get the dot from one trough to the other would be to bring the model up rapidly to bias the troughs in the desired way, then to rip away the potential barrier. The dot would then slosh over into its new trough, but the whole process, while nice and quick, would invariably involve dissipation in a real system. However, if the procedure is graceful enough, the lowering of the barrier, the tilting of the trough and the copying can be done for nothing. This is basically because the physical quantities that contribute to the energy dissipation — such as the kinetic energy of the dot moving to its new state, the work done in raising and lowering the barrier — are negligible under such circumstances. You should be able to see, incidentally, that this procedure will
work even if we don’t know what state the model is in.

When Bennett discovered all this, no one knew it could be done, although much of the preliminary groundwork had been carried out by his IBM colleague, Landauer, as far back as 1961. There was a lot of prejudice around that had to be argued against. I see nothing wrong with his arguments. I was asked by Carver Mead at CalTech to look into the energy consumption of computers, so I looked at all this stuff and gradually concluded that there was no minimum energy. This was something of a surprise to me! Bennett’s result was four years old by then but there were still people fighting over it. Also it’s nice to work this sort of thing out for yourself: as I said in Chapter One — OK, you’re not the first, but at least you understand it!

5.2.3: A Physical Implementation

Let me return to the preceding example and give you something that is essentially a physical realization of it. It is also fun to think about! We need some kind of bistable physical device, and here it is: two compass needles — just two magnetic dipoles on pivots. One end is North and the other South, and as we all know North attracts South and vice versa; otherwise we have repulsion. Now suppose that both the model and the copier are made up of such a pair. To make the analysis easier, we insist that the each member of a pair is linked to the other, in such a way that both members must point in the same direction. This means that we can analyze each system in terms of just one variable, the angle $\phi$ the needles make with the horizontal. So we have the allowed and disallowed situations shown below:

![Diagram showing Allowed Angular Configuration](image)

Fig. 5.17(a) Allowed Angular Configuration
The disallowed case would, in any case, clearly be unstable. Now, not all alignments of the needles within a pair have the same potential energy. This is obvious by comparing the states shown in Figure 5.18:

\[
\text{(horizontal)} \quad S \longrightarrow N \quad S \longrightarrow N \quad \text{with (vertical)} \quad \uparrow \uparrow \quad \downarrow \downarrow \\
\text{S S}
\]

**Fig. 5.18 Stable and Unstable States**

The first is evidently quite stable, with the tip of one needle attracting the base of the other. The second, with both arrows vertical, is quite unstable: the North poles will repel, and the needles will seek to occupy the first state or its mirror image. We can actually calculate the potential energy for a state with angle \( \phi \). It is approximately (close enough for us) given by:

\[
\text{Potential energy} = \sin^2 \phi 
\]  

(5.17)

This potential energy function looks like the graph of Figure 5.19:

**Fig. 5.19 Potential Energy as a function of \( \phi \)**
Note how similar this is to our abstract potential well. The minima are at $\phi = 0$ and $\phi = \pi$, corresponding to the stable "horizontal" states, whilst the maxima correspond to the vertical states at $\pi/2$ and $3\pi/2$. (Remember that the graph wraps around at $\phi = 0$ and $2\pi$.) The system is clearly bistable and we can see that once the needles are in one of the two minima, energy would have to be expended to push them to the other.

To manipulate the barrier in this case, we introduce a vertical magnetic field $B$. It can be shown that this adds a term:

$$-B \sin \phi$$  \hspace{1cm} (5.18)

to the potential energy. As we increase $B$, the effect is to lower the barrier between the $0$ and $\pi$ states as shown in Figure 5.20:

\begin{center}
\begin{tikzpicture}
\draw[very thick,->] (0,0) -- (4,0) node[right] {$\phi$};
\draw[very thick,->] (0,0) -- (0,4) node[above] {$\phi$};
\draw[very thick] (0,0) -- (3,0);
\draw[very thick] (0,3) -- (3,3);
\draw[very thick] (0,0) .. controls (1,1) and (2,2) .. (3,3);
\draw[very thick] (0,3) .. controls (1,2) and (2,1) .. (3,0);
\draw[very thick,->] (0,0) -- (1,1) node[below] {$B$};
\end{tikzpicture}
\end{center}

\textbf{Fig. 5.20 Barrier Manipulation in the Dipole Copier}

(You can play with numbers to gauge the exact effect of this.) The tilt force, as before, results from bringing the model closer to the copier; this time, we can see what it is about the model that causes this force — it is the magnetic field from the data bit. The force is perpendicular to $B$, and in the direction of the needles in the model. If we call it $b$, then it contributes:
\[ -b \cos \phi \] (5.19)

to the potential energy. This clearly removes the symmetry about \(\pi/2\) and \(3\pi/2\) and represents a tilting. We can now see how the copying process works. We start with the copier in a standard state, which we take to be the \(\phi = 0\) state (→→). We gently turn up the field \(B\) — or alternatively slowly move the copier from a region of weak \(B\) to one of high \(B\) — until the barrier is removed. At this stage, the dipole is vertical (Fig. 5.21):

![Fig. 5.21 Initial (Unstable) Copier State](image)

Now we bring in the model. This has already been slightly perturbing the copier pair, but not enough to have a noticeable effect so far. Now, as it gets closer, its field biases the copier needles to flip over — but not suddenly! — into a new state. (This is if a new state is appropriate: if the standard state and the model state coincide, the needles will simply return to their original position.) The model is removed, the copier taken out of the field \(B\) to restore the barrier, and the copying is finished.

Once again, you can check that this copying method will work if we do not know what the model state is. It is not difficult to see that, if performed slowly, it will cost no energy — no current, no nothing. My previous discussion was to show you the principles; this specific example is probably easier to understand.

5.2.4: A Living Computer

The foregoing example of two dipoles has a certain physical basis, but is undeniably artificial. However, here's a copying process that really is found in
Nature and is one that involves thermodynamical, rather than mechanical, forces. It occurs as one of the steps in the synthesis of proteins in a living cell. Now you probably know what proteins are — long, twisted molecular chains of amino acids (such as tryptophan or alanine) — and you may know how central they are both to the structure and functioning of living things. However, a proper understanding of the complex business that is their manufacture would require an understanding of biochemistry lying way beyond this course! I can’t make up for that here, so I’ll just try to give you enough background to let you see how the copying "machine" I have in mind behaves.

A living creature typically contains a huge number of different types of protein, each uniquely defined by some combination of specific amino acids. If the cell is to manufacture these molecules, then clearly a set of "design rules" for each protein-type must be available somewhere. This information actually resides in the DNA (Deoxyribonucleic Acid) molecule, the famous "double-helix" structure which resides in the cell nucleus. DNA comprises a double chain, each strand of which is made up of alternating phosphate and pentose sugar groups. To each sugar group is attached one of four bases, A (adenine), T (thymine), C (cytosine) and G (guanine) (a base-sugar-phosphate group is called a nucleotide). It is a certain sequence of bases that provides the code for protein synthesis.

We can break down the synthesis of proteins into two stages. The first stage, and it is only this which interests us, requires the formation of another, linear, strand of sugar phosphates with bases attached, called messenger RNA (or m-RNA). The code on the DNA is copied onto the RNA strand base by base (according to a strict matching rule, which I shall come on to), and the m-RNA, once completed, leaves the nucleus and travels elsewhere to assist in the making of the protein. The machine that does the copying is an enzyme called RNA polymerase. What happens is this. The DNA and enzyme are floating around in a crazy biological soup which contains, among other things, lots of triphosphates — such as ATP (adenosine triphosphate), CTP, GTP and UTP (U is another base, Uracil). These are essentially nucleotides with two extra phosphates attached. The polymerase attaches itself to whichever part of one of the DNA strands it is to duplicate and then moves along it, building its RNA copy base by base by reacting the growing RNA strand with one of the four nucleotides present in the soup. (A crucial proviso here is that RNA is built out of the four bases A, G,

---

4 For a discussion of this topic in the literature, see C. Bennett, Int. J. Theor. Phys 21, pp. 905-940 [1982]. [RPF]
C and U (not T), and that the RNA strand must be complementary to that on the DNA; the complementarity relationships are that As on the DNA must match with Us on the RNA, Ts with As, Cs with Gs and Gs with Cs. The nucleotides are provided in the triphosphate form, and during the addition process two of the phosphates are released back into the soup, still bound together (as a pyrophosphate). The nucleotide chosen must be the correct one, that is, complementary to the base on the DNA strand that is being copied. For example: suppose the enzyme, traveling along the DNA strand, hits a C base. Now at this particular stage of its operation a bonding between the polymerase and a GTP molecule from the soup happens to be more energetically favorable than one between it and UTP or ATP: complementarity is actually enforced by energy considerations. Preferentially, then, it will pick up a GTP molecule. It releases a pyrophosphate back out into the soup, moves along the DNA and looks for the next complementary nucleotide.

Schematically, we have the following picture (Fig. 5.22):

![Diagram showing the process of RNA synthesis](image)

RNA (N bases) + (G-S-P-P-P) → RNA (N+1 bases) + P\(_i\) (pyrophosphate)

**Fig. 5.22 Formation of m-RNA**

Now the role of enzymes in biochemical processes is as catalysts: they influence the rate at which reactions occur, but not the direction in which they proceed. Chemical reactions are reversible, and it would be just as possible for
the polymerase reaction to go the other way — that is, for the enzyme to undo the m-RNA chain it is working on. In such an event, it would extract a pyrophosphate from the surrounding soup, attach it to a base on the m-RNA, and then release the whole lot back into the environment as one of our triphosphates. The enzyme could just move along the wrong way, eat a G, move along, eat a C, move along... undoing everything. Which way the reaction goes depends on the relative concentrations of pyrophosphates and triphosphates in the soup. If there is a lot of ATP, GTP, and so on, but not much free pyrophosphate, then the rate at which the enzyme can run the reaction backwards is lowered, because it can’t find much pyrophosphate with which to pull off the m-RNA nucleotides. On the other hand, if there is an excess of free pyrophosphates over triphosphates, the reaction will tend to run the wrong way, and we’ll be uncopying and ruining our copy.

We can actually interpret these relative concentrations in terms of the number of possible states available to our system at any given computational point. If there are plenty of triphosphates around, then there are plenty of forward-moving, and comparatively few backward-moving, states available: the RNA polymerase will tend to enter the former state, in the process lowering its entropy. The difference in free energies, measured by the differing concentrations, determines the way it goes. If we get the concentrations just right, the copier will oscillate forever, and we will never get around to making copies. In an actual cell, the pyrophosphate concentration is kept low by hydrolysis, ensuring that only the copying process occurs, not its inverse. The whole RNA polymerase system is not particularly efficient as far as energy use goes: it dissipates about $100kT$ per bit. Less could be wasted if the enzyme moved a little more slowly (and of course, the reaction rate does vary with concentration gradient), but there has to be a certain speed for the sake of life! Still, $100kT$ per bit is considerably more efficient than the $10^8kT$ thrown away by a typical transistor!

To reiterate: The lesson of this section is that there is no absolute minimum amount of energy required to copy. There is a limit, however, if you want to copy at a certain speed.

---

5Bennett has nicely christened machines like this "Brownian computers" to capture the manner in which their behavior is essentially random but in which they nevertheless progress due to some weak direction of drift imposed on their operation. [PPE]
5.3: Computation: Energy Cost versus Speed

The question of speed is important and I would like to write down a formula for the amount of free energy it takes to run a computation in a finite time. This at least makes our discussion a bit more practical. There is little room for reversible computing in the computer world at the moment, although one can foresee applications that are a little more immediately useful than getting from $10^8 kT$ to under $kT$. (You can actually get to 2 or 3 $kT$ irreversibly, but you can't get under this.) For example, we can look at the problem of errors arising in parallel processing architectures where we might have thousands of processors working together. The question of error correction through coding in such a situation has arisen and is unsolved. It occurs to me that maybe the devices in the machine could all be made reversible, and then we could notice the errors as we go. What would be the cost of such reversible devices? Maybe these things will find application soon. That would make this discussion more practical to you and since computing is engineering you might value this! In any case, I shouldn't make any more apologies for my wild academic interest in the far future.

An example we gave of reversible computing was that of the chemical process of copying DNA. This involved a machine (if you like) that progressed in fits and starts, going forward a bit, then backwards, but more one than the other because of some driving force, and so ended up doing some computation (in this case, copying). We can take this as a model for more general considerations and will use this "Brownian" concept to derive a formula for the energy dissipation in such processes. This will not be a general formula for energy dissipation during computation but it should show you how we go about calculating these things. However, we will precede this discussion by first giving the general formula\(^6\), and then what follows can be viewed as illustration.

Let us suppose we have a reversible computer. Ordinarily, the free energy expended in running it reversibly will be zero, when the process is infinitesimally slow, but let us suppose that we are actually driving it forward at a rate $r$. In other words, at any given stage, it is $r$ times more likely to make a forward calculational step than a backwards one. Then, the general result is that minimum energy that must be expended per computational step in the process is:

\[^6\text{This rule is pretty general, but there will be exceptions, requiring slight corrections. We will discuss a "ballistic" computer in 8.5.5. [RPF]}\]
Note that the smaller $r$ is, the lower the energy.

Let us illustrate this rule by looking at a Brownian-type computer. Imagine we have a system, or device, in a particular state, which has a particular energy associated with it. It can go forwards or backwards into a new state, each transition corresponding either to doing a computation (forward) or undoing it (backward). We can model this situation using the energy level diagram of Figure 5.23:

![Energy Level Diagram](image)

**Fig. 5.23 The General Transition**

We assume our computer to be sitting in one of the two states, with energy $E_1$ or $E_2$. These energies will not generally be equal. Now our device can go from $E_1$ to $E_2$, a forward step — the idea is that the energies are lower in the direction of computation — or from $E_2$ to $E_1$, a backward step. The energies of the two states might be equal, but one of them could be effectively lowered by the imposition of an external driving force. We have introduced into this diagram the "activation energy" $A$, which is the energy that must be supplied to the system to cause a transition of any kind. We will focus on the effects of thermal fluctuations which will, quite randomly, cause the computer to move between states, whenever the energy of these fluctuations exceeds $A$. Such fluctuations can make the device go either way, and we can calculate the rate at which it goes in either direction. These will not be equal. Roughly, the chance of the system going into the state with energy $E_1$ is the chance that by accident it acquires enough energy to get past the barrier (that is, $A$) and into $E_i$. Clearly, the energy needed to get from $E_1$ to $E_2$, a forward step, is $(A-E_1)$, while to get from $E_2$ to $E_1$ it is $(A-E_2)$. It is a standard result in statistical mechanics that the
probability of a transition from one state to another differing in (positive) energy \( \delta E \) is:

\[
C \exp(-\delta E/kT),
\]

(5.21)

where \( C \) is a factor that carries information about the thermal fluctuations in the environment. This can be calculated through a phase-space (entropy-type) analysis, examining the probabilities of ensemble transitions between states. However, we are interested in the transition rates between states and this is describable by a similar formula. We simply have to insert another factor, say \( X \), giving us:

\[
\text{forward rate} = CX \exp[-(A-E_1)/kT] \tag{5.22}
\]

and

\[
\text{backward rate} = CX \exp[-(A-E_2)/kT]. \tag{5.23}
\]

The factor \( X \) depends on a variety of molecular properties of the particular substance (the mean free path, the speed, and so on), but the property that interests us is that it does not depend on \( E \) (consider the transition rates for the case \( E_1 = E_2 \)). We can therefore write for the ratio of the forward to backward rates:

\[
\exp[(E_1-E_2)/kT]. \tag{5.24}
\]

This depends only on the energy difference between successive states. This gives us some insight into the rate at which our computation (= reaction) proceeds, and the energy difference between each step required to drive it. The bigger the energy difference \( E_1 - E_2 \), the quicker the machine hops from \( E_1 \) to \( E_2 \), and the faster the computation.

We can tie this result into our earlier general formula by setting the above rate equal to \( r \). We then have, for the energy expended per step:

\[
kT \log r = E_1 - E_2 \tag{5.25}
\]
which makes sense.

Let me give you one more illustration of driving a computer in a particular direction. This time we will look at computational states that do not differ in their energy, but in their availability. That is, our computer selects which state of a certain kind to go into next, not on the basis of the energy of the state, but on the number of equivalent states of that kind available for it to go into. We have an example of such a situation in our DNA copier. A calculational step there involved the RNA enzyme attaching bases to the RNA chain and liberating pyrophosphates into the surroundings. The inverse step involved taking up phosphates from the solution and breaking off bases. Each step is energetically equivalent but one can be favored over the other, depending on the relative concentrations of chemicals in the soup. Suppose there is a dearth of phosphates but a wealth of bases available. Then, the number of (forward) states of the system in which a base is attached to the RNA strand and a phosphate is released — and we consider all such states equivalent — exceeds the number of states in which a phosphate is grabbed and a base released (again, all such states we take to be the same). So we can envisage a computer designed so that it proceeds by diffusion, in the sense that it is more likely to move into a state with greater, rather than lower, availability. Schematically, we have the situation shown in Figure 5.24:

![Diagram](image)

**Fig. 5.24 The Availability of States**

where \( n_i \) is the number of available states. It is possible to show in this situation (although it takes a little thought) that the ratio of the forward rate to the backward rate is:

\[
r = \frac{n_2}{n_1}. \tag{5.26}
\]

If you recall, we defined the entropy of a configuration of a system to be:
\[ S = k \log W \]  \hspace{1cm} (5.27)

where \( W \) is the probability of finding the system in that configuration. Hence we may write:

\[ kT \log r = kT (\log n_2 - \log n_1) = (S_2 - S_1)T \]  \hspace{1cm} (5.28)

(with various constant factors canceling to leave the equality). In other words, for this process the energy loss per step is equal to the entropy generated in that step, up to the usual temperature factor. Again, this makes sense.

So we can see that our general formula reduces to the specific formulae we have obtained in these instances. An interesting question that arises is: in a real world situation, can we minimize the energy taken per computational step? We know that if we have an effectively reversible computer, the chances of forward and backward movement are equal, and we have no energy loss. The price we pay for this is that a computation will take an infinite time. We will never know when we're finished. So as we've said, to get it going we want to give things a tug, lower the energies of successive steps, make them more available, or whatever. Let us suppose that we have the forward rate, \( f \), just a little bigger than the backward rate, \( b \), so the computation just goes. We write:

\[ f = b + \Theta \]  \hspace{1cm} (5.29)

where \( \Theta \) is small. Our general formula now gives:

\[ \text{energy per step} = kT \log [1+(\Theta/b)] = kT\Theta/b = kT(f-b)/b \]  \hspace{1cm} (5.30)

for small \( \Theta \). We can provide a nice physical interpretation of this expression, although at the cost of mathematical inaccuracy. We replace the formula above by one that is nearly equal to it:

\[ \text{energy per step} = kT \frac{(f-b)}{(f+b)/2}. \]  \hspace{1cm} (5.31)

This differs from the original formula by terms of order \( \Theta^2 \). Now the numerator of this fraction is the speed at which we go forward and do the calculation. It is a bit like a velocity, in that it represents the rate at which the computer drifts through its calculation, measured in steps per second. The denominator is the average rate of transition — it is a measure of the degree to which our computer
is oscillating back and forth. We can interpret this roughly as the fastest speed at which you could possibly go, backwards or forwards, which would be the speed found if the computer underwent a series of steps in one direction with no reverses: it is the greatest possible drift. So we can write, approximately:

$$\text{energy loss per step} = kT \frac{v_{\text{drift}}}{v_{\text{max}}}$$  \hspace{1cm} (5.32)

Alternatively, we can emphasize time as our variable and write:

$$\text{energy loss/step} = kT \frac{\text{minimum time taken/step}}{\text{time/step actually taken}}$$  \hspace{1cm} (5.33)

Let us now take a look at more general issues in reversible computing.

5.4: The General Reversible Computer

We have repeatedly stated that, if a computation is to be reversible, then we have to store a lot of information that we would ordinarily lose or throw away in order that we can subsequently use it to undo something. The logic gates of such a machine give us not just the answer to the logical calculation we want, but a whole lot of extra bits. A simple illustration of this for a realistic gate is a simple adder built out of reversible gates. In §2.3 I set as a problem for you, the construction of a full three-bit adder from reversible gates — specifically, using CN and CCN gates (or alternatively, just CCN gates, out of which all others can be built). An easier example, the simple two-bit adder, is built as follows:

![Diagram of a two-bit adder]

The C-input is kept set at zero (the full three-bit adder requires the addition of a fourth input line, kept set at zero). As well as the sum and carry of A+B, we
find this gate feeds the A-line through. We can see that this bit is necessary if we are to be able to reconstruct the input \((A, B)\) from the output. If you look at the three-bit adder, you will find two spare bits at the output. Generally, then, we will always need a certain amount of junk to remind us of the history of the logical operation. We can summarize the main constraint on reversible gates as follows: it is obvious that, when running a computer forward, there must be no ambiguity in the forward step — if you have a "goto", you have to know where to go to. With a reversible machine, there cannot be any ambiguity in backward steps either. You should never have a situation where you do not know where to go back to. It is this latter feature that makes reversible computing radically different from ordinary, irreversible computing.

We can, following Bennett, consider the most general computational process, and also answer a criticism leveled at advocates of reversible computing. Let us suppose we have a system of (reversible) logic units tied together, and we put into it some input data. We also have to feed in a set of "standard" zeroes, the bits that are kept set at particular values to control the reversible gates. (If we want a "standard" one instead of a zero, we can just NOT one of the zeroes: this is reversible, of course!) The logic unit will do its business — dup, dup, dup — and at the end we will find an output — the answer we want plus a pile of garbage bits, forming the history tape. This is shown in Figure 5.26 below:

![Diagram](attachment:image.png)

**Fig. 5.26 The General Reversible Computation**
Now this picture makes it look like you start up with a blank tape (or a preset one) and end up with a lot of chaos. Not surprisingly, everyone said that was where the entropy was going: "This randomization of zeroes is (in Bennett’s picture) fueling the running of your machine. How can keeping this data make your computation practically reversible? It’s rather like claiming that you can make an irreversible heat engine reversible by keeping the water that all the heat has gone into, rather than throwing it away. If you don’t throw the water away, sure you have all the information you need about the history of the system, but that hardly means the engine is going to be able to run backwards, reversing the motions of water molecules!" In the thermodynamic case, that would indeed be silly. But it isn’t so for computing. By adding one more tape to the system, and feeding the results through another machine, we can bypass this difficulty (Fig. 5.27):

![Diagram](image)

**Fig. 5.27 A Zero Entropy Loss Reversible Computer**

Let us try to make sense of this! The new logic unit that we have added is the reverse of the original (hence we have labeled it $M^{-1}$) and is also reversible. $M^{-1}$
is such that if we feed the output of $M$ through this, it undoes all the work on it and feeds us back the original inputs to $M$. The new tape is a cleared register which we will use to copy the answer to our computation. We begin as before, feeding into $M$ the input data and the standard bits for control. $M$ gives us an output and a history tape (marked garbage in the diagram). The history tape we feed directly into $M^{-1}$. We also feed the data output tape in. However, before we do this we make a copy of it onto the cleared register. We have shown this schematically as a fanout, but this actually represents a copy process (which is, of course, a reversible operation).

The reverse machine $M^{-1}$ now undoes all the work done by $M$, producing as its output the standard bits and the input data. At the end of the whole process we are left with the answer to the computation, plus an exact copy of the inputs we started with. So our grand machine has done a calculation for no entropy loss (ideally — in practice we would have to drive the system a little as discussed) and reversible computing really can save us work. Of course, there will be an energy loss when we wipe our tapes clean to do another calculation.

Reversible computing is quite a strange concept for those used to thinking in classical Boolean terms, so let me suggest a few problems for you to work on to help you become more comfortable with the ideas.

**Problem 5.3:** Suppose a reversible computer is carrying out a calculation and it needs to execute a subroutine. So it gets sent off to some other place to execute a compact set of instructions. Now these instructions must be reversible, as are the basic computing elements, and so there is a chance that once we are into our subroutine we might find ourself running backward. It might even happen that we get back to the start of the routine — and then have to re-enter the main body of the program where we left it! The question is: Given that this same subroutine might be used several times throughout the computation, how does the machine know where to return to when this reverse happens? You might like to think about this. Somehow you have to have a number of memory stacks to keep track of where you have to go to find the subroutine, but also where to go back to should you reverse. This is your first problem in reversible computing — how to handle subroutines.

**Problem 5.4:** A related problem concerns how to get "if" clauses to work. What if, after having followed an "if... then..." command, the machine starts to reverse? How can the machine get back to the original condition that dictated which way the "if" branched? Of course, a set of initial conditions can result in a single "if" output ("if $x = 2, 3, 4$ or $6.159$ let $F = d"$), so this condition may not
be uniquely specified. Here is a nice way to analyze things. Simply bring in a new variable at each branch, and assign a unique value to this variable for each choice at a branch point. You might like to work this through in detail.

**Problem 5.5:** A simple question to ask about a general reversible computer is: How big a history tape do we need? The gates we have considered so far have had the number of outputs equal to the number of inputs. Is this always necessary for reversibility? As far as I know, this question hasn’t even been asked by theorists. See if you can work it out. Certainly the minimum has something to do with the number of possible inputs that the output could represent, and we’ll apparently need a number of bits to keep track of that (on top of the actual outputted answer). So the questions are: firstly, what is the minimum number of bits needed to keep a gate reversible in principle, and secondly, could we actually accomplish it?

**5.5: The Billiard Ball Computer**

To give you a demonstration of a reversible computer that can actually do calculations, I am now going to discuss an ingeniously simple machine invented by Fredkin, Toffoli and others. In this device, the movement of billiard balls on a plane is used to simulate the movement of electronic signals (bits) through logic gates. We fire balls into the machine to represent the input, and the distribution of balls coming out gives us our output. The balls all move diagonally across a planar grid and obey the laws of idealized classical mechanics (i.e zero friction and perfectly elastic collisions). To introduce you to the basic idea, examine the following diagram (Fig. 5.28):

![Diagram of a basic two-ball collision computation](image)

*Fig. 5.28 The Basic Two-ball Collision Computation*
This illustrates how a two-ball collision realizes a two-input four-output logic function. The data to this gate is represented by the presence of a ball at a particular position (1) or its absence (0). For example, the gate has two input channels, $A$ and $B$. If we fire a ball in at $A$, then the input at $A$ is binary 1. If there is no ball, it is zero. Similarly with $B$. If we find a ball coming out at point $X$, this means output $X$ is 1, and so on. There are four possible input states, and for each we use basic mechanics to calculate the configuration of balls coming out of the device. There are four possible outputs, two corresponding to one input ball being absent and the other going straight through, and two corresponding to a collision.

Let us suppose there is no ball at $A$. If there is a ball at $B$, it will continue on through the "machine", coming out at $X$. We can see that we will only get a ball at $X$ if there is no ball at $A$ and one ball at $B$. In logic terms, $X$ is 1 if and only if $B$ is 1 and $A$ is 0, so we have:

$$X = B \text{ AND } \text{NOT } A$$  \hspace{1cm} (5.34)

Similarly, we find that:

$$Y = A \text{ AND } \text{NOT } B$$  \hspace{1cm} (5.35)

Output $W$ is a little trickier. We will find a ball there only if there is a ball at both $A$ and $B$. Likewise for output $Z$. Hence, both $W$ and $Z$ realize the same AND function:

$$W, Z = A \text{ AND } B$$  \hspace{1cm} (5.36)

Let us summarize this with some fancy notation (Fig. 5.29):

![Logical Structure of the Basic Collision Computation](image)

**Fig. 5.29 Logical Structure of the Basic Collision Computation**
This is the fundamental collision of this billiard ball computer and you can see how neatly the logic element drops out of it. We can build other logic functions besides AND with this gate. For example, we can use it to make a FANOUT. If we set \( A = 1 \) (the billiard equivalent of a control line set to "on") and take our output from \( W \) and \( Z \), then clearly this has the effect of branching our \( B \) input: a ball at \( B \) produces one at each of \( W \) and \( Z \); no ball at \( B \) leaves both outputs blank. You can also make a CN gate with this unit (try it). However, by itself, the basic collision gate will not make enough elements to build a whole computer — we'd be stuck with pairs of balls going along two lines, and we could never change anything! How do we reroute balls? We have to introduce two fundamental mechanical devices. The first, which you would never invent if you were a logician, as it seems a damn silly thing to do, I'll call a collision gate; in this device, two balls go in, but four come out (Fig. 5.30):

![Fig. 5.30 The Collision Gate](image)

This is a sort of all-in "double-FANOUT" process, which we achieve by letting the two incoming balls collide with two stationary ones. (You might find it an interesting exercise to consider the energy and momentum properties of this gate.) The second and more important device is a redirection gate. This is just a mirror to reflect a ball. It can be oriented any way you wish, although we restrict ourselves to four possibilities (Fig. 5.31):

![Fig. 5.31 Four Redirection Gates](image)
Mirrors enable us to do a lot of things. For example, we can use mirrors to construct a "crossover" device (Fig. 5.32):

![Crossover Device Diagram]

Fig. 5.32 A Crossover Device

Incidentally, this device tells us something important about the balls, namely, that they are indistinguishable. We do not tell them apart, and are interested only in their presence or absence. The above crossover device actually switches the incoming balls, but as we can’t tell them apart, it looks as if they just pass each other by. Note that if one ball is missing, the other just sails right through.

To show you the sorts of thing you can build with these basic structures, I will first give you a unit that acts as a switch (Fig. 5.33):

![Switching Device Diagram]

Fig. 5.33 A Switching Device
This is a sort of offset crossover. Note that, irrespective of whether or not there is a B input, the lower right output is always the same as A. This is a "debris" bit, corresponding to the control line fed through the gate. Of course, we are used to such outputs by now.

A question that arises in the context of this chapter is obviously: "OK, show me how to make reversible gates with all these mirrors and balls." Specifically, can we build, say, a CN gate? The answer is that we can, and a CCN gate too if we like. However, it is more enlightening to build a Fredkin, or controlled exchange gate. This is because it is possible to build everything we could want, just out of Fredkin gates! I’ll remind you of what such a gate is (Fig. 5.34):

![Diagram of the Fredkin Gate]

Fig. 5.34 The Fredkin Gate

Line A goes through unchanged. This is true of B and C also, if A=0; but if A=1, B and C switch. I won’t leave building a Fredkin gate as an exercise. It is constructed from four switching devices of the kind depicted in Figure 5.33, put together with considerable ingenuity as shown in Figure 5.35:

![Diagram of the Fredkin Gate Realized by Billiard Ball Gates]

Fig. 5.35 The Fredkin Gate Realized by Billiard Ball Gates
Obviously, there is no point in making a computer like this except for fun. However, it does show how profoundly simple the basic structure of a machine can be.

Now anybody who is familiar with bouncing balls knows that if there's a slight error, it is rapidly magnified. Suppose you have a ball on a table and you drop another onto it from above, right in the middle. You might think: "Oh, it'll go straight down, then straight up, and so on." Everybody has an intuition about this but if you played with balls as a baby you know that you can't bounce one ball on another. It doesn't work! What happens is that as soon as the ball bounces ever so slightly wrongly, the next bounce is further out and the ball comes down slightly more cock-eyed. When it comes down next time, it is further out still and hits the lower ball in an even more glancing fashion. Next time, the balls will probably miss altogether.

The reasons for this are not hard to fathom. Although at the macroscopic level, balls seem stable and solid, at the microscopic level, they are a seething mass of jiggling molecules. Thermal oscillations, statistical mechanical fluctuations and whatnot, all contribute corrections to the naïve collision of ideal balls. In fact, even the tiniest effects of quantum mechanics get in the way. According to the Uncertainty Principle, we cannot know both the precise location and momentum of a ball, so we cannot drop one perfectly straight. Suppose we have two ideal 1cm balls, and we drop one onto the other from a height of 10cm. How many bounces can we get away with before, according to quantum mechanics, things have to go awry? We can actually calculate this and the answer is about seventeen bounces. Of course, in reality the disturbances from more classical phenomena are far more significant and we would never get anywhere near this quantum limit. Don't forget, even your hand will be shaking from Brownian motion!

So surely the billiard ball machine idea is nonsense? We may not be dropping balls from a height, but we are colliding them and we would therefore expect errors to accrue just as inevitably. So how can we claim to have a physically implementable reversible computer? After all, all you have to do is give me an error per collision, and I will tell you how long you have before the machine falls apart. $10^{-3}$? Five minutes. $10^{-6}$? OK, ten minutes. It looks completely hopeless. In order to get this system to work, we have to find some way to keep straightening out the balls. Perhaps we could put them in troughs, guiding them in some way. But if you put a ball in a trough it'll slosh back and forth, getting worse and worse, unless there are losses — absorption, resistance, dissipation. Even if we design our troughs to cope with these difficulties,
inevitably energy will be lost because of friction in the trough. We would have to pull the balls through to drive the machine. Now if you drive it just a little, you can find that the energy required to drive it is a minimum of the ratio:

\[
\frac{kT \text{ time to make collision}}{\text{speed at which it happens}}.
\]  

(5.37)

This expression has not been analyzed in any great detail for the billiard ball machine.

5.6: Quantum Computation

The billiard ball computer operates chiefly according to the laws of classical mechanics. However, inspired by the questions it brings up, people have asked me (and others have thought about this too?): "What would the situation be if our computer is operating according to the laws of quantum mechanics?" Suppose we wanted to make extremely small computers, say the size of a few atoms. Then we would have to use the laws of quantum mechanics, not classical mechanics. Wouldn’t the Uncertainty Principle screw things up? Not necessarily. I will wind up this chapter by briefly considering what may become the computers of the future — quantum computers.

We are asking yet another question about absolute limitations! This time, it is: "How small can you make a computer?" This is one area where, I think, I’ve made a contribution. Unlike an airplane, it turns out that we can make it pretty much as small as we want. There will be engineering details about wires, and we will have to find a way of magnifying outputs and whatnot, but we are here discussing questions of principle, not practicality. We cannot get any smaller than atoms because we will always need something to write on,

\[7\text{Notably the physicist Paul Benioff (see, for example, "Quantum Mechanical Models of Turing Machines that Dissipate No Energy", Phys. Rev. Lett. 48, pp. 1581-1585 [1982]). [Editors]}

\[8\text{It is interesting to note that most computer theorists treat wire as idealized thin string that doesn’t take up any room. However, real computer engineers frequently discover that they just can’t get enough wires in! (We’ll return to this in Chapter Seven.) [RPF]}

\[9\text{I am not allowing for the possibility that some smart soul will build a computer out of more fundamental particles! [RPF]}

but all we actually need are bits which communicate. An atom, or a nucleus will do since they are natural "spin systems", i.e., they have measurable physical attributes that we can put numbers to and we can consider each different number to represent a state. We can make magnets the size of atoms. (It'll put some chemists out of a job, but that's progress). But the point is that there are no further limitations on size imposed by quantum mechanics, over and above those due to statistical and classical mechanics.

I won't go into too much detail here: I will return to the subject, and all its lovely math — in the next chapter. For now, I'll just give you the gist of the ideas. Let us begin with some idealized quantum mechanical system (anything very small) and suppose that it can be in one of two states — say "up", which might correspond to an excited state, and "down", corresponding to a de-excited state. Alternatively, the two states might refer to the spin of the quantum system (spin is a crude classical analogy). We can actually allow it to be in other states as well, but for our purposes it just needs at least two states to represent a binary number: up is one, down is zero. I'll call this quantum mechanical system an atom, so that you can get a grip on its basic nature, but bear in mind that it could be something more complex, or even something simpler, like an electron (which has two spin states). Now the idea is that we build our computing device out of such atoms by stringing them together in a particular way. We start with part or all of the system — a string of atoms in one or other of their two states — representing a number, our input. We then let the whole system evolve over time according to the laws of quantum theory, interacting with itself — the atoms change states, the ones and zeroes move around — until at some point we have a bunch of atoms somewhere which will be in certain states, and these will represent our answer.

We could set the machine running with a single input bit — say firing an atom into the system — and design things such that the machine itself tells us when the calculation is complete, say by firing an atom out of the system. Nothing would be trustable until the output bit was one. You would measure this bit, then change it to zero and freeze the answer for examination. Putting the information in and out is not, incidentally, a particularly quantum mechanical process — it is a matter of amplification. Interestingly, as a rule one cannot predict the time the computer will take to complete its calculation. It turns out to be ballistic, like Fredkin's, but at the end you only get a wave packet for the arrival of the answer. We test to see whether or not the answer is in the machine or not. For the simple machine I have designed (see next chapter), there exist several quantum mechanical "amplitudes" (certain physical properties of the system) which, upon measurement, tell us how far through the calculation we
have gone, but ultimately we have to wait for the machine to let us know it's finished.

So, in 2050, or before, we may have computers that we can't even see! I will return to these strange beasts in the next chapter.