The Fundamental Physical Limits of Computation

What constraints govern the physical process of computing? Is a minimum amount of energy required, for example, per logic step? There seems to be no minimum, but some other questions are open

by Charles H. Bennett and Rolf Landauer

A computation, whether it is performed by electronic machinery, on an abacus or in a biological system such as the brain, is a physical process. It is subject to the same questions that apply to other physical processes: How much energy must be expended to perform a particular computation? How long must it take? How large must the computing device be? In other words, what are the physical limits of the process of computation?

So far it has been easier to ask these questions than to answer them. To the extent that we have found limits, they are terribly far away from the real limits of modern technology. We cannot profess, therefore, to be guiding the technologist or the engineer. What we are doing is really more fundamental. We are looking for general laws that must govern all information processing, no matter how it is accomplished. Any limits we find must be based solely on fundamental physical principles, not on whatever technology we may currently be using.

There are precedents for this kind of fundamental examination. In the 1940's Claude E. Shannon of the Bell Telephone Laboratories found there are limits on the amount of information that can be transmitted through a noisy channel; these limits apply no matter how the message is encoded into a signal. Shannon's work represents the birth of modern information science. Earlier, in the mid- and late 19th century, physicists attempting to determine the fundamental limits on the efficiency of steam engines had created the science of thermodynamics. In about 1960 one of us (Landauer) and John Swanson at IBM began attempting to apply the same type of analysis to the process of computing. Since the mid-1970's a growing number of other workers at other institutions have entered this field.

In our analysis of the physical limits of computation we use the term "information" in the technical sense of information theory. In this sense information is destroyed whenever two previously distinct situations become indistinguishable. In physical systems without friction, information can never be destroyed; whenever information is destroyed, some amount of energy must be dissipated (converted into heat). As an example, imagine two easily distinguishable physical situations, such as a rubber ball held either one meter or two meters off the ground. If the ball is dropped, it will bounce. If there is no friction and the ball is perfectly elastic, an observer will always be able to tell what state the ball started out in (that is, what its initial height was) because a ball dropped from two meters will bounce higher than a ball dropped from one meter.

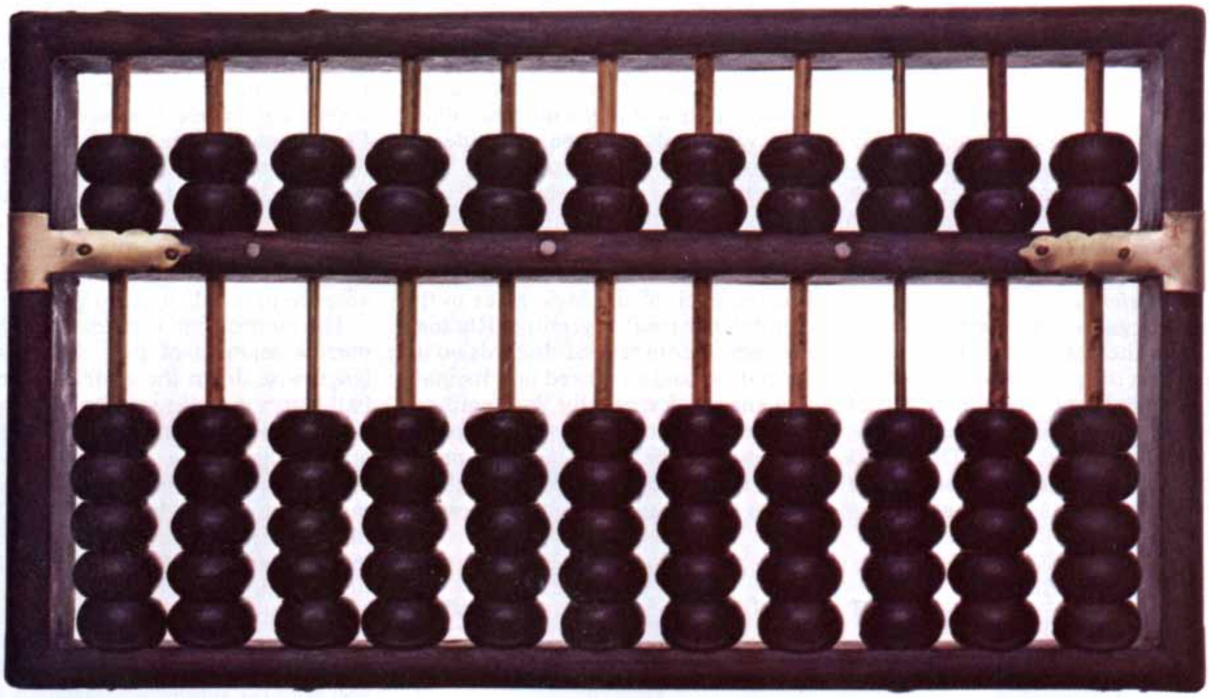
If there is friction, however, the ball will dissipate a small amount of energy with each bounce, until it eventually stops bouncing and comes to rest on the ground. It will then be impossible to determine what the ball's initial state was; a ball dropped from two meters will be identical with a ball dropped from one meter. Information will have been lost as a result of energy dissipation.

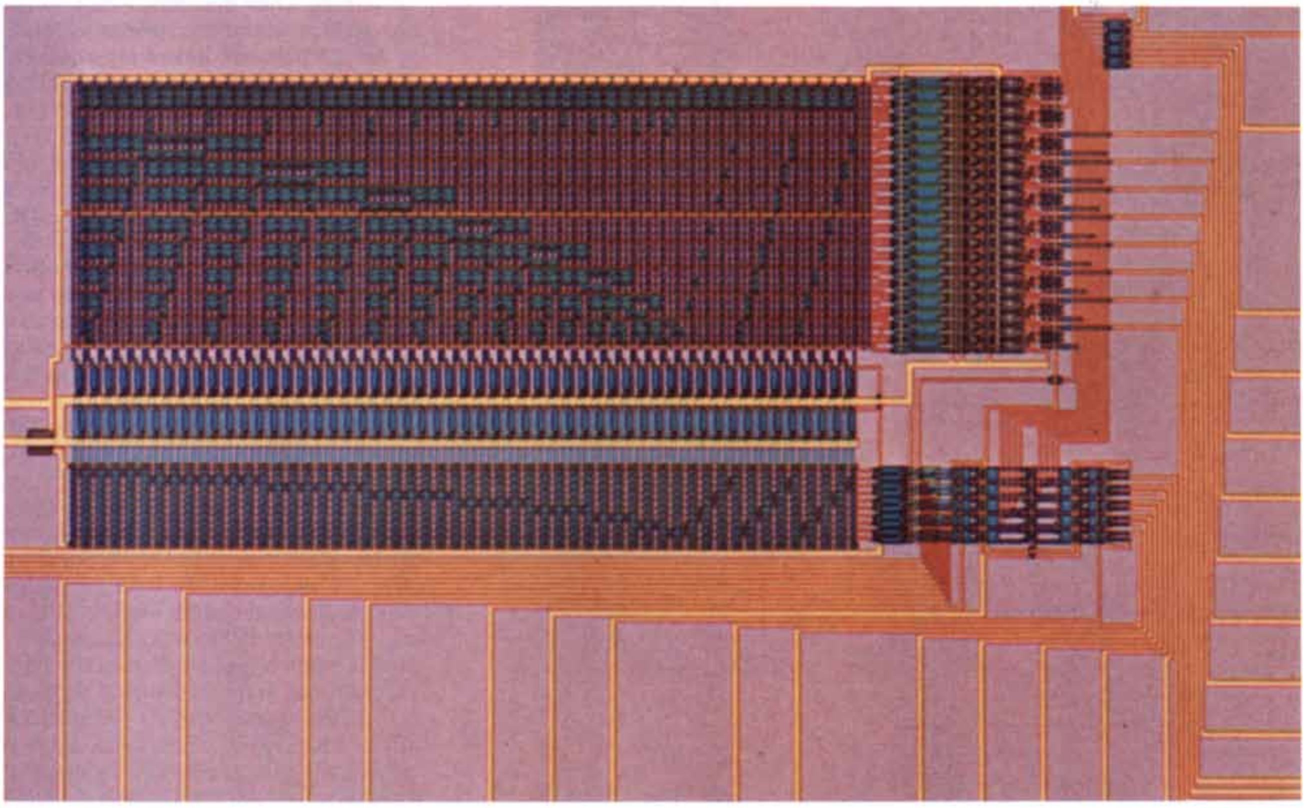
Here is another example of information destruction: the expression $2 + 2$ contains more information than the expression $= 4$. If all we know is that we have added two numbers to yield 4, then we do not know whether we have added $1 + 3$, $2 + 2$, $0 + 4$ or some other pair of numbers. Since the output is implicit in the input, no computation ever generates information.

In fact, computation as it is currently carried out depends on many operations that destroy information. The so-called *and* gate is a device with two input lines, each of which may be set at 1 or 0, and one output, whose value depends on the value of the inputs. If both inputs are 1, the output will be 1. If one of the inputs is 0 or if both are 0, the output will also be 0. Any time the gate's output is a 0 we lose information, because we do not know which of three possible states the input lines were in (0 and 1, 1 and 0, or 0 and 0). In fact, any logic gate that has more input than output lines inevitably discards information, because we cannot deduce the input from the output. Whenever we use such a "logically irreversible" gate, we dissipate energy into the environment. Erasing a bit of memory, another operation that is frequently used in computing, is also fundamentally dissipative; when we erase a bit, we lose all information about that bit's previous state.

Are irreversible logic gates and erasures essential to computation? If they are, any computation we perform has to dissipate some minimum amount of energy.

As one of us (Bennett) showed in 1973, however, they are not essential. This conclusion has since been demonstrated in several models; the easiest of these to describe are based on so-called reversible logic elements such as the Fredkin gate, named for Edward Fredkin of the Massachusetts Institute of Technology. The Fredkin gate has three input lines and three outputs. The input on one line, which is called the control channel, is fed unchanged through the gate. If the control channel is set at 0, the input on the other two lines also passes through unchanged. If the control line is a 1, however, the





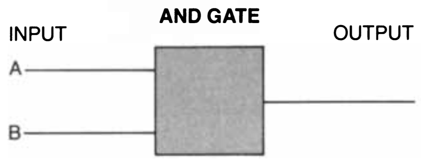
CONVENTIONAL COMPUTING DEVICES, the abacus and the logic chip, both dissipate energy when they are operated. The "logic gates" central to the design of a chip expend energy because they discard information. A chip consumes energy for a less fundamental reason as well: it employs circuits that draw power even when they merely hold information and do not process it. The abacus is dissipative because of friction between its beads and rods. It could not be built of frictionless components: if there were no static friction, the beads' positions would change under the influence of random thermal motion. Static friction exerts a certain minimum force no matter what the beads' velocity, and so there is some minium energy that the abacus requires no matter how slowly it is operated.

outputs of the other two lines are switched: the input of one line becomes the output of the other and vice versa. The Fredkin gate does not discard any information; the input can always be deduced from the output.

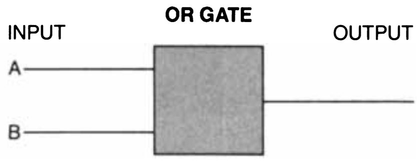
Fredkin has shown that any logic device required in a computer can be implemented by an appropriate arrangement of Fredkin gates. To make the computation work, certain input lines of some of the Fredkin gates must be preset at particular values [*see lower illustration below*].

Fredkin gates have more output lines than the gates they are made to simulate. In the process of computing, what seem to be "garbage bits," bits of information that have no apparent use, are therefore generated. These bits must somehow be cleared out of the computer if we are to use it again, but if we erase them, it will cost us all the energy dissipation we have been trying to avoid.

Actually these bits have a most important use. Once we have copied down the result of our computation, which will reside in the normal output bits, we simply run the computer in reverse. That is, we enter the "garbage bits" and output bits that were produced by the computer's normal operation as "input" into the "back end" of the computer. This is possible because each of the logic gates in the computer is itself reversible. Running the computer in reverse discards no information, and so it need not dissipate any energy. Eventually the computer will be left exactly as it was before the computation began. Hence it is possible to complete a "computing cycle"—to run a computer and then to return it to its original state—without dissipating any energy.

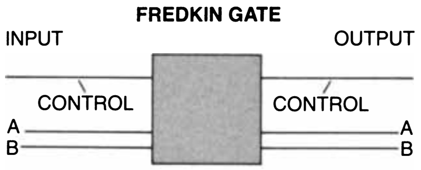


A	B	OUTPUT
1	1	1
1	0	0
0	1	0
0	0	0

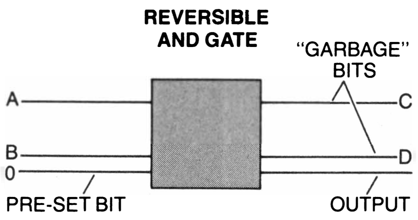


A	B	OUTPUT
1	1	1
1	0	1
0	1	1
0	0	0

CONVENTIONAL LOGIC GATES dissipate energy because they discard information. For example, if the output of an *and* gate is 0, there is no way to deduce what the input was.



INPUT			OUTPUT		
CONTROL	A	B	CONTROL	A	B
1	1	1	1	1	1
1	1	0	1	0	1
1	0	1	1	1	0
1	0	0	1	0	0
0	1	1	0	1	1
0	1	0	0	1	0
0	0	1	0	0	1
0	0	0	0	0	0



			"GARBAGE"	
A	B	OUTPUT	C	D
1	1	1	1	0
1	0	0	1	0
0	1	0	0	1
0	0	0	0	0

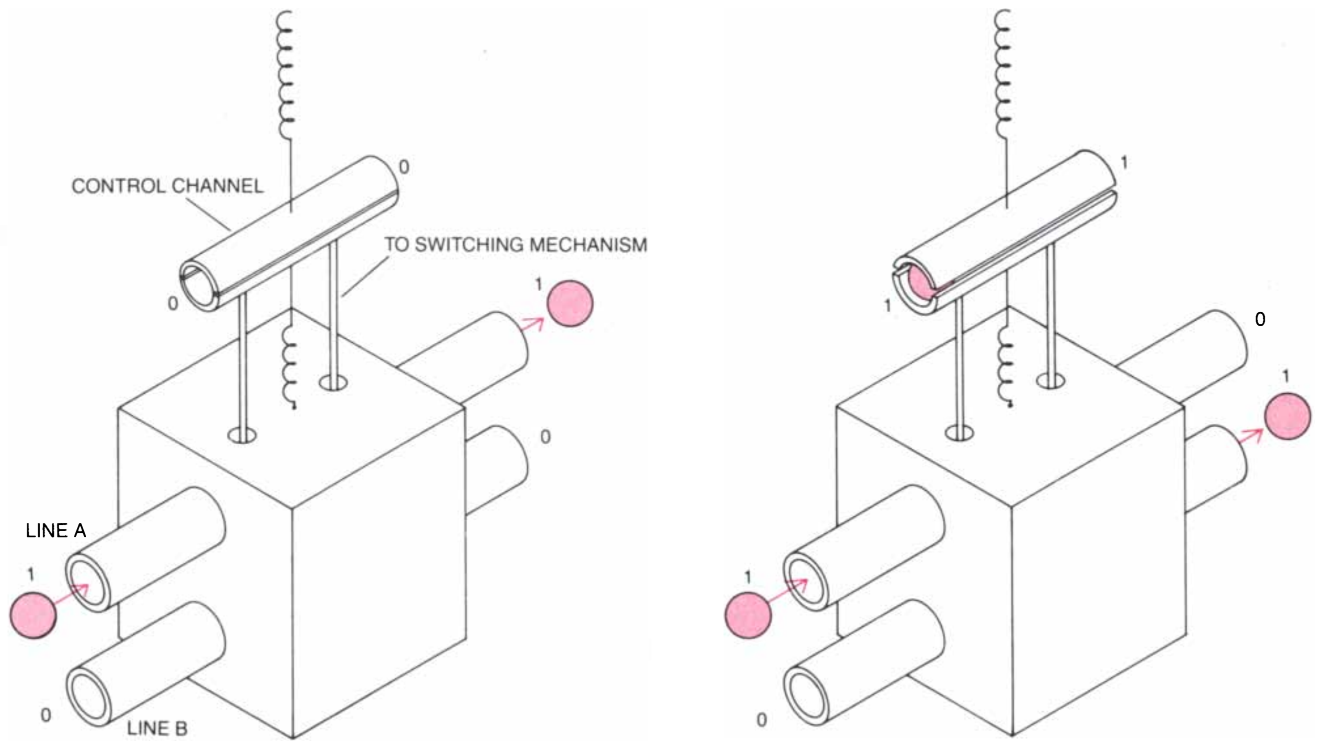
FREDKIN REVERSIBLE LOGIC GATE need not dissipate energy; the input can always be deduced from the output. The gate has a "control" line, the value of which is not changed by the gate. If the bit on the control line is a 0, the values of the other two lines are also untouched; if it is a *1*, however, the input of line *A* becomes the output of line *B* and vice versa. Reversible gates can be arranged to implement any function performed by an irreversible gate. To implement the *and* operation (*right*) one input is preset to equal 0, and two output bits, called garbage bits, are temporarily ignored. When the computation is complete, these bits are used to operate the gate in reverse, returning the computer to its original state.

So far we have discussed a set of logic operations, not a physical device. It is not hard, however, to imagine a physical device that operates as a Fredkin gate. In this device the information channels are represented by pipes. A bit of information is represented by the presence or absence of a ball in a particular section of pipe; the presence of a ball signifies a 1 and the absence of a ball signifies a 0.

The control line is represented by a narrow segment of pipe that is split lengthwise down the middle. When a ball enters the split segment of pipe, it pushes the two halves of the pipe apart, actuating a switching device. The switching device channels any input balls that may be in the other two pipes: when a ball is present in the control line, any ball that enters an input pipe is automatically redirected to the other pipe. To ensure that the switch is closed when no control ball is present, there are springs that hold the two halves of the split pipe together. A ball entering the split pipe must expend energy when it compresses the springs, but this energy is not lost; it can be recovered when the control ball leaves the split pipe and the springs expand.

All the balls are linked together and pushed forward by one mechanism, so that they move in synchrony; otherwise we could not ensure that the various input and controlling balls would arrive at a logic gate together. In a sense the forward progress of the computation is really motion along a single degree of freedom, like the motion of two wheels rigidly attached to one axle. Once the computation is done we push all the balls backward, undoing all the operations and returning the computer to its initial state.

If the entire assembly is immersed in an ideal viscous fluid, then the frictional forces that act on the balls will be proportional to their velocity; there will be no static friction. The frictional force will therefore be very weak if we are content to move the balls slowly. In any mechanical system the energy that must be expended to work against friction is equal to the product of the frictional force and the distance through which the system travels. (Hence the faster a swimmer travels between two points, the more energy he or she will expend, although the distance traveled is the same whether the swimmer is fast or slow.) If we move the balls through the Fredkin gates at a low speed, then the energy expended (the product of force and distance) will be very small, because the frictional force depends directly on the balls' speed. In



IDEALIZED PHYSICAL REALIZATION of a Fredkin gate substitutes pipes for wires and the presence or absence of a ball for a 1 or 0. A narrow, split segment of pipe represents the control channel. When a ball passes through it, the pipe spreads apart, operating a switching mechanism; the mechanism in turn switches any input ball from line *A* to line *B* and vice versa. A pair of springs keeps the control channel closed when no ball is in it. This gate does not need static friction in order to operate; it could be immersed in a viscous fluid, and the frictional forces could be made to depend only on the balls' velocity. Then the energy dissipation could be as small as the user wished: to lower the amount of energy dissipated, it would only be necessary to drive the balls through the device more slowly.

fact, we can expend as little energy as we wish, simply by taking a long time to carry out the operation. There is thus no minimum amount of energy that must be expended in order to perform any given computation.

The energy lost to friction in this model will be very small if the machine is operated very slowly. Is it possible to design a more idealized machine that could compute without any friction? Or is friction essential to the computing process? Fredkin, together with Tommaso Toffoli and others at M.I.T., has shown that it is not.

They demonstrated that it is possible to do computation by firing ideal, frictionless billiard balls at one another. In the billiard-ball model perfect reflecting "mirrors," surfaces that redirect the balls' motion, are arranged in such a way that the movement of the balls across a table emulates the movement of bits of information through logic gates [*see illustration on next page*]. As before, the presence of a ball in a particular part of the computer signifies a 1, whereas the absence of a ball signifies a 0. If two balls arrive simultaneously at a logic gate, they will collide and their paths will change; their new paths represent the output of the gate. Fredkin, Toffoli and others have described arrangements of mirrors that correspond to different types of logic gate, and they have shown that billiard-ball models can be built to simulate any logic element that is necessary for computing.

To start the computation we fire a billiard ball into the computer wherever we wish to input a 1. The balls must enter the machine simultaneously. Since they are perfectly elastic, they do not lose energy when they collide; they will emerge from the computer with the same amount of kinetic energy we gave them at the beginning.

In operation a billiard-ball computer produces "garbage bits," just as a computer built of Fredkin gates does. After the computer has reached an answer we reflect the billiard balls back into it, undoing the computation. They will come out of the machine exactly where we sent them in, and at the same speed. The mechanism that launched them into the computer can then be used to absorb their kinetic energy. Once again we will have performed a computation and returned the computer to its initial state without dissipating energy.

The billiard-ball computer has one major flaw: it is extremely sensitive to slight errors. If a ball is aimed slightly incorrectly or if a mirror is tilted at a slightly wrong angle, the balls' trajectories will go astray. One or more balls will deviate from their intended paths, and in due course errors will combine to invalidate the entire computation. Even if perfectly elastic and frictionless billiard balls could be manufactured, the small amount of random thermal motion in the molecules they are made of would be enough to cause errors after a few dozen collisions.

Of course we could install some kind of corrective device that would return any errant billiard ball to its desired path, but then we would be obliterating information about the ball's earlier history. For example, we might be discarding information about the extent to which a mirror is tilted incorrectly. Discarding information, even to correct an error, can be done only in a system in which there is friction and loss of energy. Any correctional device must therefore dissipate some energy.

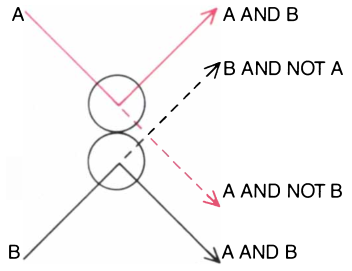
Many of the difficulties inherent in the billiard-ball computer can be made less extreme if microscopic or submicroscopic particles, such as electrons, are used in place of billiard balls. As Wojciech H. Zurek, who is now at the Los Alamos National Laboratory, has pointed out, quantum laws, which can restrict particles to a few states of motion, could eliminate the possibility that a particle might go astray by a small amount.

Although the discussion so far has been based primarily on classical dynamics, several investigators have proposed other reversible computers that are based on quantum-mechanical principles. Such computers, first proposed by Paul Benioff of the Argonne National Laboratory and refined by others, notably Richard P. Feynman of the California Institute of Technology, have so far been described only in the most abstract terms. Essentially the particles in these computers would be arranged so that the quantum-mechanical rules governing their interaction would be precisely analogous to the rules describing the predicted outputs of various reversible logic gates. For example, suppose a particle's spin can have only two possible values: up (corresponding to a binary 1) and down (corresponding to a 0). The interactions between particle spins can be prescribed in such a way that the value of one particle's spin changes depending on the spin of nearby particles; the spin of the particle would correspond to one of the outputs of a logic gate.

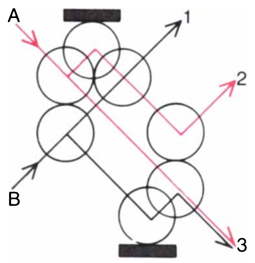
So far this discussion has concentrated on information processing. A computer must store information as well as process it. The interaction between storage and processing is best described in terms of a device called a Turing machine, for Alan M. Turing, who first proposed such a machine in 1936. A Turing machine can perform any computation that can be performed by a modern computer. One of us (Bennett) has shown that it is possible to build a reversible Turing machine: a Turing machine that does not discard information and can therefore be run with as small an expenditure of energy as the user wishes.

A Turing machine has several components. There is a tape, divided into discrete frames or segments, each of which is marked with a 0 or a 1; these bits represent the input. A "read/write head" moves along the tape. The head has several functions. It can read one bit of the tape at a time, it can print one bit onto the tape and it can shift its position by one segment to the left or right. In order to remember from one cycle to the next what it is doing, the head mechanism has a number of distinct "states"; each state constitutes a slightly different configuration of the head's internal parts.

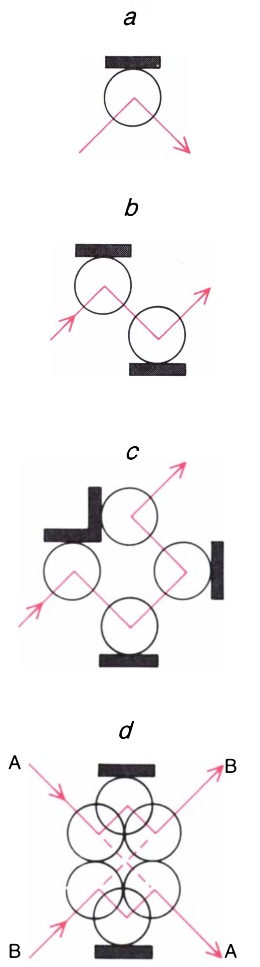
In each cycle the head reads the bit on the segment it currently occupies;

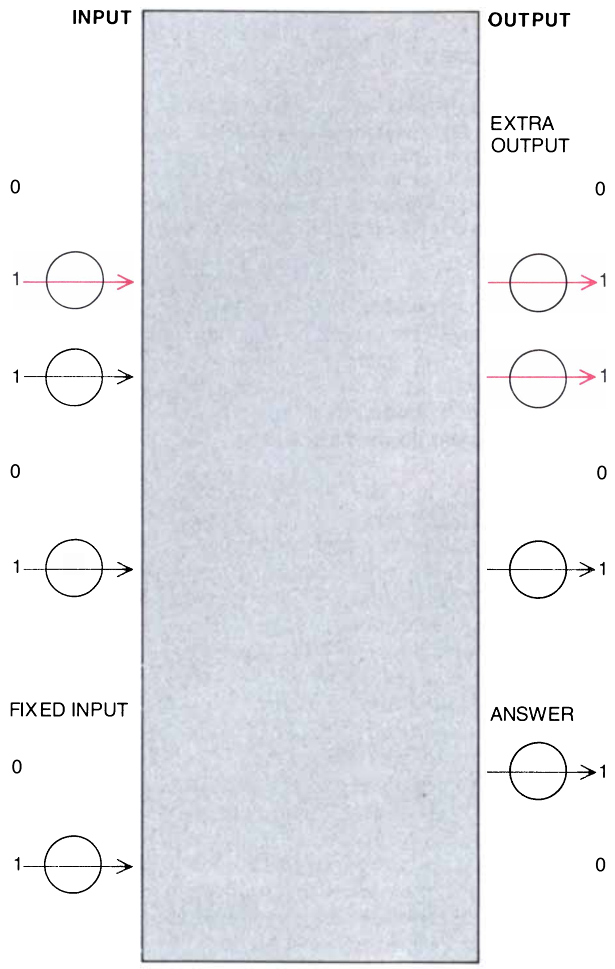


INPUT		OUTPUT			
A	B	1	2	3	4
1	1	1	0	0	1
1	0	0	0	1	0
0	1	0	1	0	0
0	0	0	0	0	0



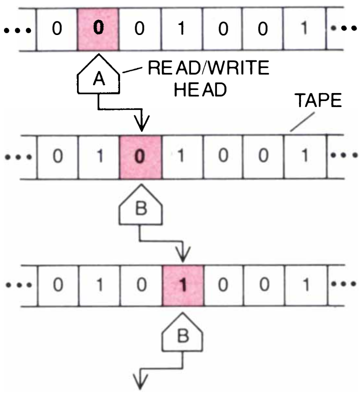
INPUT		OUTPUT		
A	B	1	2	3
1	1	0	1	1
1	0	0	0	1
0	1	1	0	0
0	0	0	0	0





BILLIARD-BALL COMPUTER employs the movement of billiard balls on a table to simulate the movement of bits through logic gates. In billiard-ball logic gates (*left*) the balls' paths are redirected by collisions with one another or with reflecting "mirrors." In addition to their role in gates, mirrors can deflect a ball's path (*a*), shift the path sideways (*b*), delay the ball's motion without changing its final direction or position (*c*) or allow two lines to cross (*d*). It is possible to arrange mirrors so that the resulting "computer" implements the function of any logic chip. For example, a billiard-ball computer could be made to test whether a number is prime. One such computer (*right*) accepts as input any five-bit number (in this case 01101, or 13) and the fixed input sequence 01. Like a Fredkin gate, a billiard-ball computer typically returns more output bits than its user needs. In the case shown, the computer returns the original input number itself (which is the "extra" output), and an "answer" sequence: 10 if the input number is prime and 01 if it is composite.

then it prints a new bit onto the tape, changes its internal state and moves one segment to the left or right. The bit it prints, the state it changes into and the direction in which it moves are determined by a fixed set of transition rules. Each rule specifies a particular set of actions. Which rule the machine follows is determined by the state of the head and the value of the bit that it reads from the tape. For example, one rule might be: "If the head is in state *A* and is sitting on a segment of tape that is printed with a 0, it should change that bit to a 1, change its state to state *B* and move one segment to the right." It may happen that the transition rule instructs the machine not to change its internal state, not to print a new bit onto the tape or to halt its operation. Not all Turing machines are reversible, but a reversible Turing machine can be built to perform any possible computation.



TRANSITION RULES				
HEAD STATE	BIT READ	CHANGE BIT TO	CHANGE STATE TO	MOVE TO
A	1	0	A	LEFT
A	0	1	B	RIGHT
B	1	1	A	LEFT
B	0	0	B	RIGHT

TURING MACHINE can be constructed in such a way that it can perform any computation a computer can. An infinitely long tape is divided into discrete segments, each of which bears either a 0 or a 1. A "read/write head," which can be in any of several internal states (here there are only two states, *A* and *B*), moves along the tape. Each cycle begins as the head reads one bit from a segment of the tape. Then, in accordance with a fixed set of transition rules, it writes a bit onto that segment, changes its own internal state and moves one segment to the left or right. This Turing machine, because it has only two head states, can do only trivial computations; more complicated machines with more head states are capable of simulating any computer, even one much more complicated than themselves. To do so they store a representation of the larger machine's complete logical state on the unlimited tape and break down each complex cycle into a large number of simple steps. The machine shown is logically reversible: it is always possible to deduce the machine's previous configuration. Other Turing machines, with different transition rules, are not reversible.

The reversible Turing-machine models have an advantage over such machines as the frictionless billiard-ball computer. In the billiard-ball computer random thermal motion causes unavoidable errors. Reversible Turing-machine models actually exploit random thermal motion: they are constructed in such a way that thermal motion itself, with the assistance of a very weak driving force, moves the machine from one state to the next. The progress of the computation resembles the motion of an ion (a charged particle) suspended in a solution that is held in a weak electric field. The ion's motion, as seen over a short period of time, appears to be random; it is nearly as likely to move in one direction as in another. The applied force of the electric field, however, gives the net motion a preferred direction: the ion is a little likelier to move in one direction than in the other.

It may at first seem inconceivable that a purposeful sequence of operations, such as a computation, could be achieved in an apparatus whose direction of motion at any one time is nearly random. This style of operation is quite common, however, in the microscopic world of chemical reactions. There the trial-and-error action of Brownian motion, or random thermal motion, suffices to bring reactant molecules into contact, to orient and bend them into the specific conformation required for them to react, and to separate the product molecules after the reaction. All chemical reactions are in principle reversible: the same Brownian motion that accomplishes the forward reaction sometimes brings product molecules together and pushes them backward through the transition. In a state of equilibrium a backward reaction is just as likely to occur as a forward one.

In order to keep a reaction moving in the forward direction, we must supply reactant molecules and remove product molecules; in effect, we must provide a small driving force. When the driving force is very small, the reaction will take nearly as many backward steps as forward ones, but on the average it will move forward. In order to provide the driving force we must expend energy, but as in our ball-and-pipe realization of the Fredkin gate the total amount of energy can be as small as we wish; if we are willing to allow a long time for an operation, there is no minimum amount of energy that must be expended. The reason is that the total energy dissipated depends on the number of forward steps divided by the number of backward steps. (It is actually proportional to the logarithm of this ratio, but as the ratio increases or decreases so does its logarithm.) The slower the reaction moves forward, the smaller the ratio will be. (The analogy of the faster and slower swimmers is valid once again: it requires less total energy to go the same net number of reaction steps forward if the reaction moves slowly.)

We can see how a Brownian Turing machine might work by examining a Brownian tape-copying machine that already exists in nature: RNA polymerase, the enzyme that helps to construct RNA copies of the DNA constituting a gene. A single strand of DNA is much like the tape of a Turing machine. At each position along the strand there is one of four "bases": adenine, guanine, cytosine or thymine (abbreviated *A, G, C* and *T*). RNA is a similar chainlike molecule whose four bases, adenine, guanine, cytosine and uracil (*A, G, C* and *U*) bind to "complementary" DNA bases.

The RNA polymerase catalyzes this pairing reaction. The DNA helix is normally surrounded by a solution containing a large number of nucleoside triphosphate molecules, each consisting of an RNA base linked to a sugar and a tail of three phosphate groups. The RNA-polymerase enzyme selects from the solution a single RNA base that is complementary to the base about to be copied on the DNA strand. It then attaches the new base to the end of the growing RNA strand and releases two of the phosphates into the surrounding solution as a free pyrophosphate ion. Then the enzyme shifts forward one notch along the strand of DNA in preparation for attaching the next RNA base. The result is a strand of RNA that is complementary to the template strand of DNA. Without RNA polymerase this set of reactions would occur very slowly, and there would be little guarantee that the RNA and DNA molecules would be complementary.

The reactions are reversible: sometimes the enzyme takes up a free pyrophosphate ion, combines it with the last base on the RNA strand and releases the resulting nucleoside triphosphate molecule into the surrounding solution, meanwhile backing up one notch along the DNA strand. At equilibrium, forward and backward steps would occur with equal frequency; normally other metabolic processes drive the reaction forward by removing pyrophosphate and supplying the four kinds of nucleoside triphosphate. In the laboratory the speed with which RNA polymerase acts can be varied by adjusting the concentrations of the reactants (as Judith Levin and Michael J. Chamberlin of the University of California at Berkeley have shown). As the concentrations are brought closer to equilibrium the enzyme works more slowly and dissipates less energy to copy a given section of DNA, because the ratio of forward to backward steps is smaller.

Although RNA polymerase merely copies information without processing it, it is relatively easy to imagine how a hypothetical chemical Turing machine might work. The tape is a single long backbone molecule to which two types of base, representing the binary 0 and 1, attach at periodic sites. A small additional molecule is attached to the 0 or 1 group at one site along the chain. The position of this additional molecule represents the position of the Turing machine's head. There are several different types of "head molecule," each type representing a different machine state.

The machine's transition rules are represented by enzymes. Each enzyme is capable of catalyzing one particular reaction. The way these enzymes work is best demonstrated by an example.

Suppose the head molecule is type *A* (indicating that the machine is in state *A*) and is attached to a 0 base. Also suppose the following transition rule applies: "When the head is in state *A* and reads a 0, change the 0 to a 1, change state to *B* and move to the right." A molecule of the enzyme representing this rule has a site that fits a type-*A* head molecule attached to a 1 base. It also has one site that fits a 0 base and one site that fits a *B* head [*see illustration on opposite page*].

To achieve the transition, the enzyme molecule first approaches the tape molecule at a location just to the right of the base on which the *A* head resides. Then it detaches from the tape both the head molecule and the 0 base to which the head was attached, putting in their place a 1 base. Next it attaches a *B* head to the base that is to the right of the 1 base it has just added to the tape. At this point the transition is complete. The head's original site is changed from a 0 to a 1, the head molecule is now a type *B*, and it is attached to the base that is one notch to the right of the previous head position.

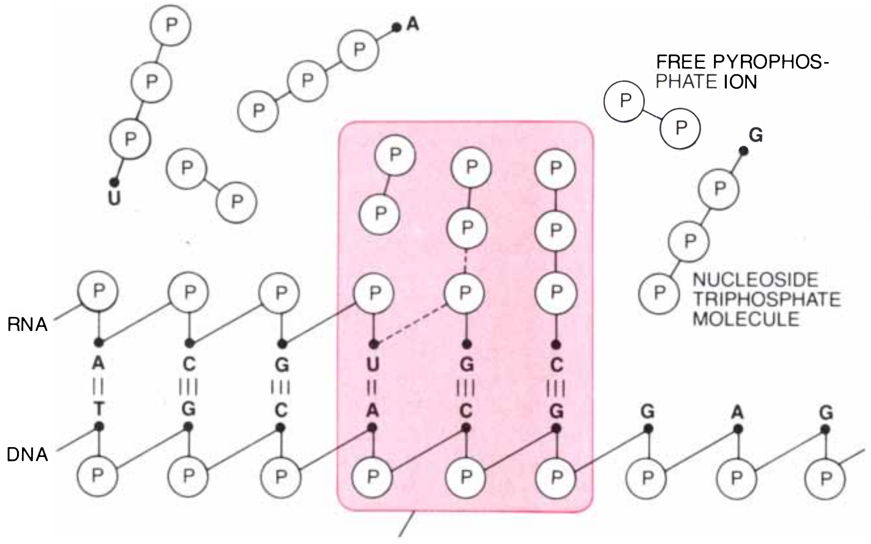
During the operation of a Brownian Turing machine the tape would have to be immersed in a solution containing many enzyme molecules, as well as extra 0's, 1's, *A*'s and *B*'s. To drive the reaction forward there would have to be some other reaction that cleaned the enzyme molecules of detached heads and bases. The concentrations of the reactants that clean the enzyme molecules represent the force that drives the Turing machine forward. Again we can expend as little energy as we wish simply by driving the machine forward very slowly.

The enzymatic Turing machine would not be error-free. Occasionally a reaction that is not catalyzed by any enzyme might occur; for example, a 0 base could spontaneously detach itself from the backbone molecule and a 1 base could be attached in its place. Similar errors do indeed occur during RNA synthesis.

In principle it would be possible to eliminate errors by building a Brownian Turing machine out of rigid, frictionless clockwork. The clockwork Turing machine involves less idealization than the billiard-ball computer but more than the enzymatic Turing machine. On the one hand, its parts need not be manufactured to perfect tolerances, as the billiard balls would have to be; the parts fit loosely together, and the machine can operate even in the presence of a large amount of thermal noise. Still, its parts must be perfectly rigid and free of static friction, properties not found in any macroscopic body.

Because the machine's parts fit together loosely, they are held in place not by friction but by grooves or notches in neighboring parts [*see illustration on page 56*]. Although each part of the machine is free to jiggle a little, like the pieces of a well-worn wood puzzle, the machine as a whole can only follow one "computational path." That is, the machine's parts interlock in such a way that at any time the machine can make only two kinds of large-scale motion: the motion corresponding to a forward computational step and that corresponding to a backward step.

The computer makes such transitions only as the accidental result of the random thermal motion of its parts biased by the weak external force. It is nearly as likely to proceed backward along the computational path, undoing the most recent transition, as it is to proceed forward. A small force, provided externally, drives the computation forward. This force can again be



RNA POLYMERASE, an enzyme, acts as a reversible tape-copying machine; it catalyzes the reaction that constructs RNA copies of segments of DNA. As the enzyme moves along a strand of DNA, it selects from the surrounding solution a nucleoside triphosphate molecule (an RNA base bound to a sugar and a "tail" of three phosphate groups) that is complementary to the DNA base about to be copied. It then attaches the new base to the end of the RNA strand and releases a free pyrophosphate ion consisting of two phosphates. The reaction is reversible: sometimes the enzyme takes up the last link of RNA, attaches it to a pyrophosphate ion and returns the resulting molecule to the solution, backing up a notch on the DNA strand. When the reaction is close to chemical equilibrium, the enzyme takes almost as many backward as forward steps and the total energy needed to copy any segment of DNA is very small. The reaction can be made less dissipative by being run more slowly; there is no minimum amount of energy that must be expended to copy a segment of DNA.

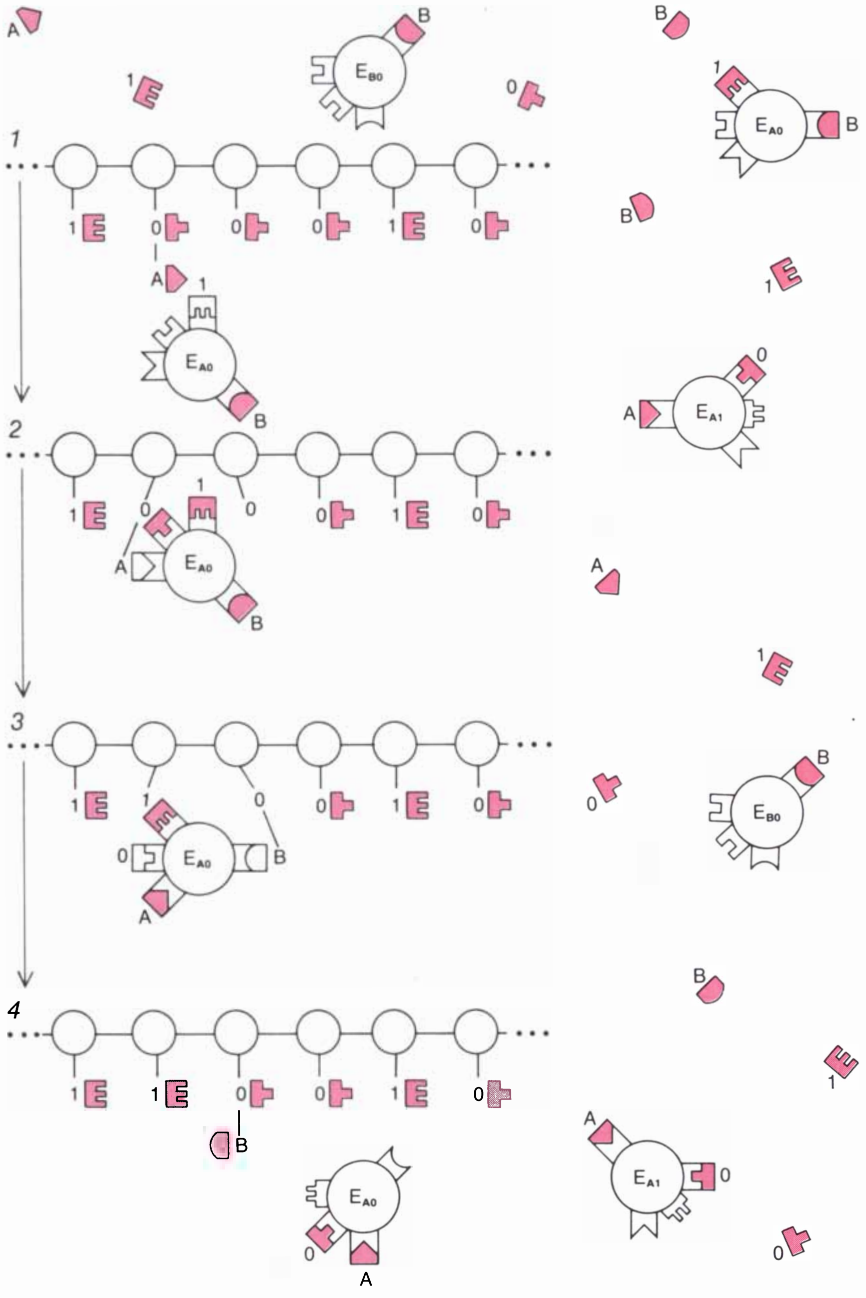
as small as we wish, and so there is no minimum amount of energy that must be expended in order to run a Brownian clockwork Turing machine.

According to classical thermodynamics, then, there is no minimum amount of energy required in order to perform a computation. Is the classical thermodynamical analysis in conflict with quantum theory? After all, the quantum-mechanical uncertainty principle states there must be an inverse relation between our uncertainty about how long a process takes and our uncertainty about how much energy the process involves. Some investigators have suggested that any switching process occurring in a short period of time must involve a minimum expenditure of energy.

In fact the uncertainty principle does not require any minimum energy expenditure for a fast switching event. The uncertainty principle would be applicable only if we attempted to measure the precise time at which the event took place. Even in quantum mechanics extremely fast events can take place without any loss of energy. Our confidence that quantum mechanics allows computing without any minimum expenditure is bolstered when we remember that Benioff and others have developed models of reversible quantum-mechanical computers, which dissipate no energy and obey the laws of quantum mechanics.

Thus the uncertainty principle does not seem to place a fundamental limit on the process of computation; neither does classical thermodynamics. Does this mean there are no physical limitations to computing? Far from it. The real limitations are associated with questions that are much harder to answer than those we have asked in this article. For example, do elementary logic operations require some minimum amount of time? What is the smallest possible gadgetry that could accomplish such operations? Because scales of size and time are connected by the velocity of light, it is likely that these two questions have related answers. We may not be able to find these answers, however, until it is determined whether or not there is some ultimate graininess in the universal scales of time and length.

At the other extreme, how large can we make a computer memory? How many particles in the universe can we bring and keep together for that purpose? The maximum possible size of a computer memory limits the precision with which we can calculate. It will limit, for example, the number of decimal places to which we can calculate pi. The inevitable deterioration proc-

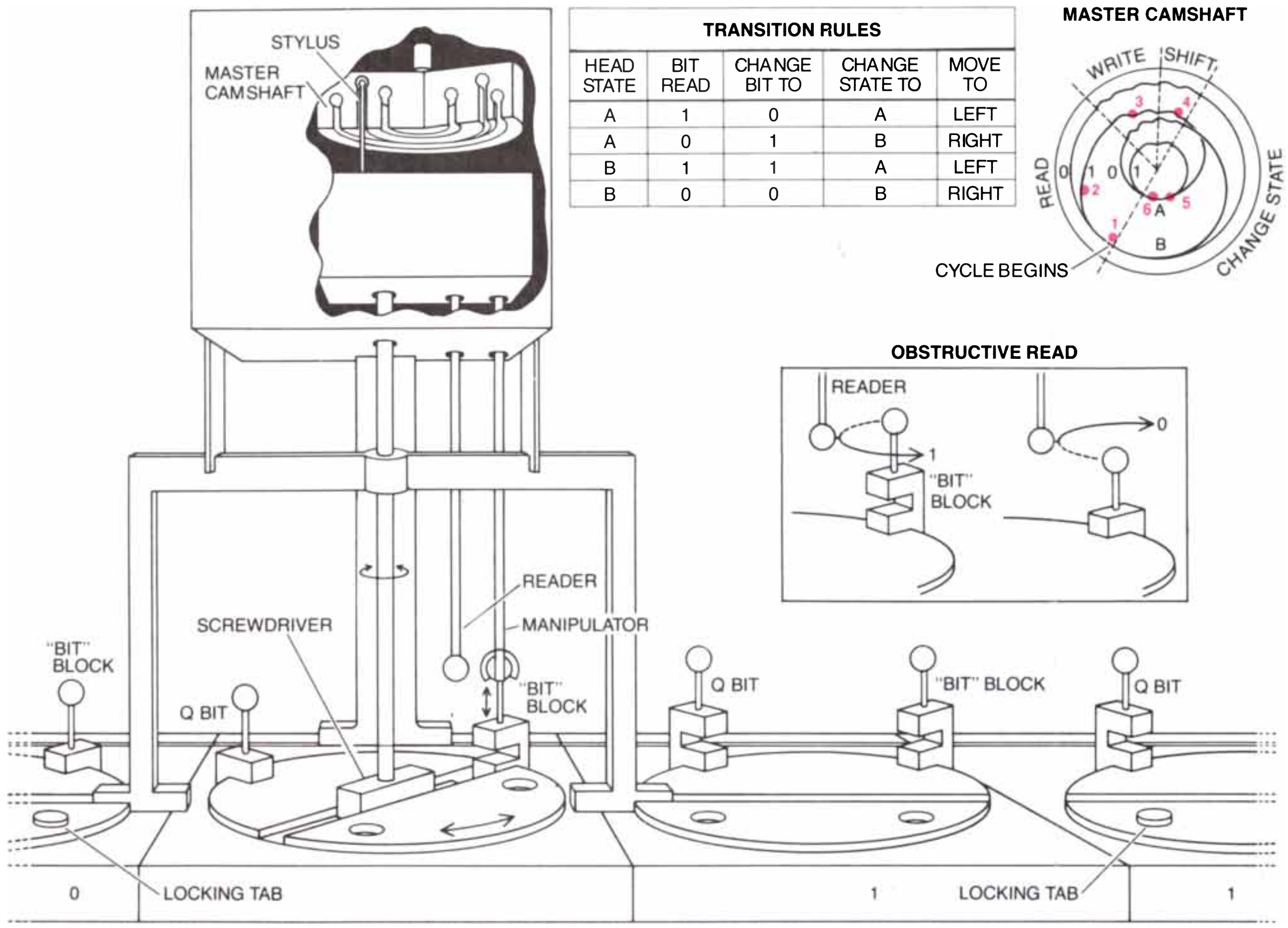


TRANSITION RULES				
HEAD STATE	BIT READ	CHANGE BIT TO	CHANGE STATE TO	MOVE TO
A	1	0	A	LEFT
A	0	1	B	RIGHT
B	1	1	A	LEFT
B	0	0	B	RIGHT

HYPOTHETICAL ENZYMATIC TURING MACHINE could perform a computation with no minimum expenditure of energy. Molecules representing 0 and 1 bits are attached at periodic intervals to a backbone molecule. A small additional molecule, representing the Turing machine's head, is attached to the 0 or 1 group at one site on the chain (*1*). There are several types of head molecule, each type representing a different internal machine state. Transition rules are represented by enzymes. In each cycle an enzyme attaches itself to the head molecule and the bit molecule to which the head is attached (*2*); then it detaches them from the chain, putting in their place the appropriate bit molecule (*3*). As it does so it rotates, so that it attaches the appropriate head molecule to the bit that occupies the site one notch to the right or left of the bit it has just changed. Now the cycle is complete (*4*): the value of a bit has been changed, and the head has changed state and shifted its position. Each kind of enzyme is able to catalyze one such set of reactions. As in the case of RNA synthesis, these reactions can be made to dissipate an arbitrarily small amount of energy.

esses that occur in real computers pose another, perhaps related, question: Can deterioration, at least in principle, be reduced to any desired degree, or does it impose a limit on the maximum length of time we shall be able to devote to any one calculation? That is, are there certain calculations that cannot be completed before the computer's hardware decays into uselessness?

Such questions really concern limitations on the physical execution of mathematical operations. Physical laws, on which the answers must ultimately be based, are themselves expressed in terms of such mathematical operations. Thus we are asking about the ultimate form in which the laws of physics can be applied, given the constraints imposed by the universe that the laws themselves describe.



TRANSITION RULES

HEAD STATE	BIT READ	CHANGE BIT TO	CHANGE STATE TO	MOVE TO
A	1	0	A	LEFT
A	0	1	B	RIGHT
B	1	1	A	LEFT
B	0	0	B	RIGHT

BROWNIAN CLOCKWORK TURING MACHINE, made out of rigid, frictionless parts, relies on random jiggling of its loosely fitted parts to change from state to state. When a part is held in place, it is not by friction but by grooves or notches in neighboring parts. Parts interlock in such a way that they can follow only one "computational path"; although they are free to jiggle a little, the only large-scale motions they can make correspond to forward or backward computational steps. The operation of the machine is driven slowly forward by a very weak force; at any instant the machine is almost as likely to move backward as forward. On the average, however, the machine will move forward and the computation will eventually end. The machine can be made to dissipate as small an amount of energy as the user wishes, simply by employing a force of the correct weakness. Segments of tape are represented by grooved disks; bits are represented by E-shaped blocks, which are locked onto the disks in either the up (1) or the down (0) position. The head consists of a rigid framework and a complicated mechanism (most of which is not shown) from which are suspended a reader, a manipulator and a screwdriver-shaped rod. The machine's operation is controlled by a grooved "master camshaft," which resembles a phonograph record (*top left and far right*); different grooves correspond to different head states. At the beginning of a cycle the head is positioned above one of the disks and a "stylus" is in the "read" segment of the groove in the master camshaft that corresponds to the machine's current head state. During the "read" part of the cycle (*1*) the reader determines whether the disk's "bit" block is up or down by a process called an obstructive read (*center right*). In an obstructive read the reader moves past the block, following a high or a low path; one of the paths will be obstructed by the knob on the end of the block, and so there will be only one path for the reader to follow. At the point on the master camshaft that corresponds to this "decision" the grooves branch; each groove splits into two, and the stylus is guided into the groove that corresponds to the bit's value (*2*). Then the master camshaft turns until the stylus is in the "write" segment (*3*). In this segment each groove contains a different set of "instructions" for the machine to follow; the instructions are transmitted by a complex linkage between the stylus and the rest of the mechanim. If the instructions call for the bit's value to change, the manipulator moves over and grasps the knob; then the screwdriver rotates the disk until the block is free to move, the manipulator moves the block up or down and the screwdriver rotates the disk again to hold the block in place. After the stylus passes through the "write" segment of the master camshaft it enters the "shift" segment (*4*). Each groove in this segment contains instructions to move the head one segment to the left or right. Then the stylus enters the "change state" segment of the camshaft (*5*), where grooves merge in such a way that the stylus falls into the groove representing the next head state. The cycle is now complete (*6*). Disks adjacent to the one being read are held in place by the head's framework. Disks that are farther away are held by "locking tabs." The locking tab on each disk is coupled to a special bit, called the Q bit, on an adjacent disk. The linkages between Q bits and locking tabs are constructed so that the disk currently being read is free to move, while disks far to the right or left are held still.