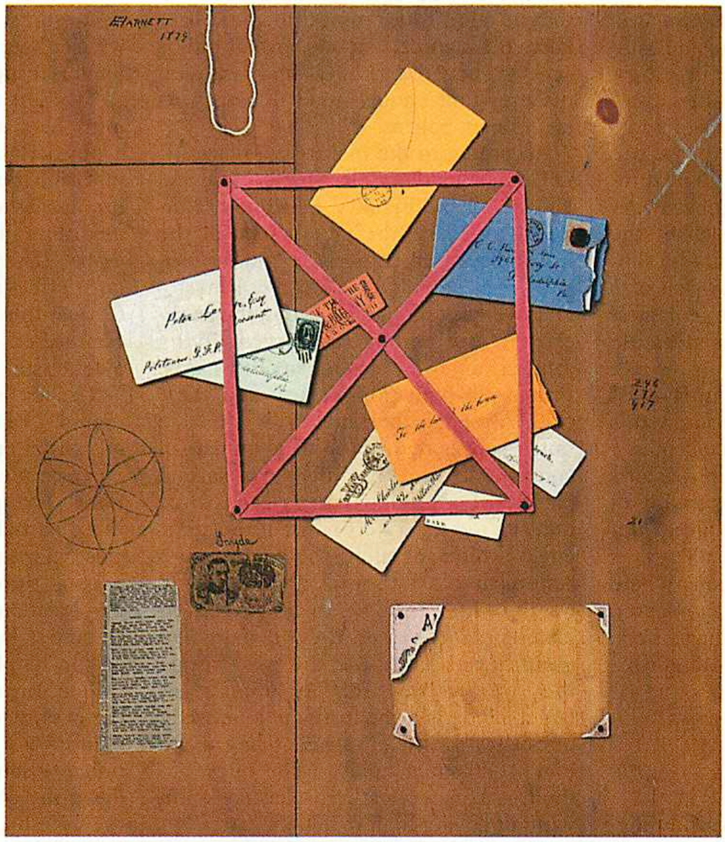
The Sciences Sep/Oct 1990

Brian Hayes



William M. Harnett, The Artist's Letter Rack, *1879*

No Forwarding Address

The rich suffer some of life's most tiresome problems. For example, when you lead a migratory existence, moving with the seasons from Palm Springs to New York or from Monte Carlo to Saint Moritz, where do you receive your mail? People burdened with the upkeep of three or four houses, a yacht and a ski chalet must forever be leaving forwarding orders at the post office and sending out change-of-address cards. Do their magazines ever catch up with them? How does American Express know where to send the bill?

At the opposite end of the economic spectrum the same trouble appears in a somewhat different form. When you move with the seasons from the cane fields to the beet fields to the onion fields—when, rather than multiple homes, you have no fixed home at all—it's still awkward to collect your mail.

Even for those of us in the economic middle, mobility has its annoyances. I have never occupied more than one residence at a time, nor have I lived out of the back of a pickup truck, and yet in the past decade I have sent out more than a thousand change-of-address notices. Once I had to mail a batch even though I hadn't moved: the post office had decided to renumber the buildings on my street. At present I find my mailbox filled with long-delayed and battered magazines forwarded from an address I left ten weeks ago as well as an intriguing selection of mail-order catalogues directed to previous tenants of my current lodgings.

I mention these trials of modern life because I think some of them might have a technological remedy. The essence of my idea is that we should send mail not to boxes but to people. To show how that might be done, I'll set aside postal procedures for a time and explore some concepts of computer architecture.

There are many ways to describe a computing machine; they range from Alan Turing's classic account of a device for reading from and writing to an infinite tape to the blueprints for the mechanism of a contemporary supercomputer. Most such accounts are formally equivalent to one another in the sense that what they describe can solve the same classes of problems. Here I want to take a highly abstract view of what constitutes a computer. It is to be a device built out of objects drawn from three realms: data, instructions and addresses. In most modern computers objects of all three kinds are represented as binary numbers of a certain fixed length, say thirty-two binary digits, or bits.

Computing requires interactions among the three kinds of objects. The most obvious interactions are the ones in which instructions act on data. For example, an instruction meaning add might take two data items as input and, if all goes well, yield a third item (the sum of the first two) as output. Other arithmetical operations can be formulated in the same way.

Another essential category of interactions reverses the flow of information: data act on instructions. Such an event is generally called a conditional branch. Typically a data item is examined, and if it meets some stated condition—say, if it is equal to zero—a certain instruction is executed; otherwise a different instruction (or perhaps no instruction at all) is executed. Thus conditional branching provides a feedback channel, carrying information from the realm of data back to the realm of instructions. Without such decision-making power a computer would be crippled. It would always execute a fixed sequence of instructions, oblivious to the data it was acting on.

Most accounts of computer architecture strongly emphasize the interactions of data and instructions, but the important role of addresses is not so widely recognized. An address is a place where you go to find something. Every data item and instruction has an address, without which it cannot be identified. An add instruction might state: "Add the number at address *x* to the number at address *y*; leave the sum at address *z*." The instruction carries out the same series of actions no matter what values are stored at *x*, *y* and *z*. Likewise a conditional branch might be expressed: "If the data value at address *x* is zero, execute the instruction at address *m*, and otherwise execute the instruction at address *n*." Again, this choice is made regardless of what instructions reside at *m* and *n*.

The workings of this abstract computing machine are easier to follow with the help of a visual model. And what better place to look for such a model than the local post office? Imagine an array of post office boxes, all the same shape and size. Each box can hold only one item, which could be a datum or an instruction, and each is assigned a unique number, which is its address. An add instruction specifies the addresses of two boxes whose contents are to be added and the address of a third box, where the sum is to be stored. A conditional branch determines which of two boxes one should look into to find the next instruction to execute.

So far, the computer I have described is just an array of boxes that hold either data or instructions. There is one more crucial idea to be introduced before the computer can do its part to speed the mails: allow any post office box to hold an address. It is important to be clear about what this means. Every box has an assigned address, which identifies the box and is unchangeable; you might imagine that address as being engraved on the door of the box. But the new idea is that a box can also contain an address, which can be inserted or removed as freely as can instructions or data. There is no necessary relation between the address marked on the door and an address stored in the box.

What is the point of stuffing addresses into postal boxes? Suppose you are analyzing some test results, and you want to construct a bar chart showing how many students received each possible score between 200 and 800. In your analysis program you set aside the sequence of boxes from addresses 200 through 800, each of which is to hold the number of students whose score matches that numerical address. You store an initial value of 0 in each box. Then you write a long sequence of statements beginning: "If the score is equal to 200, add 1 to the value in box 200; if the score is equal to 201, add 1 to the value in box 201." There are 601 such statements in all. Each test score in the sample is checked against each of the conditional-branch statements in turn, and so when the program halts, each box within the designated range holds the number of students whose score is equal to the address of that box.

The program works—but how clumsy and wasteful it is! Six hundred one conditional-branch instructions must be executed to classify a single test score. Furthermore, writing the program is a tedious and repetitive chore, rather like the punishment meted out by some grammar school disciplinarian.

There is a better way, one that also begins with storing a value of 0 in each of the boxes from 200 through 800. But now, as each test score is received, it is temporarily stored in some spare location, say the box with address 100. The long list of conditional-branch instructions is replaced with a single program statement: "Add 1 to the value found in the box whose address is found in the box whose address is 100." For example, if a test score is 712, the computer would store this value in box 100, interpret the contents of box 100 as an address, and add 1 to the value found in box 712.

This scheme is called indirect addressing. Every modern computer, from the large-scale parallel processor to the hand-held programmable calculator, offers some form of indirection. It is a powerful mechanism, though its importance is sometimes overlooked. Just as the conditional branch provides a link between data and instructions, indirect addressing bridges the realms of data and addresses.

In the bar-chart program a score enters as an item of data but gets interpreted as an address. The opposite transformation—interpreting an address as a datum—is equally important. In constructing the bar chart the final procedure might be to step through all the accumulated values, drawing a bar for each one. The program could accomplish this by loading the number 200 into a reserved location and then repeatedly carrying out the following procedure: First treat the value specified by the reserved location as an address, fetch the number stored at that address and draw a bar of a corresponding length. Then reinterpret the value at the reserved location as a datum and add 1 to it so that the value of the score frequency can be fetched from the next address in sequence. Stop when the value in the reserved location exceeds 800.

Indirect addressing is employed in all areas of computer science. Most programmers make use of it through a device known as a pointer variable, a built-in facility of programming languages such as C and Pascal. A pointer is simply an address that has been given a name, for convenient reference. For example, if `score` is the name of a variable intended to represent SAT scores in a Pascal program, then `^score` is a pointer that holds the address of `score`.

Pointers are particularly useful for keeping track of data that are subject to revision or reorganization. Consider a compendium of baseball statistics, in which a few thousand players are listed alphabetically, along with their batting averages and other vital data. If you belatedly insert an entry for Henry Aaron, all the other records will probably have to be moved to make room at the head of the list—an unwieldy operation. Pointers make inserting and deleting much easier. The physical arrangement of the records is arbitrary; they need not be kept in alphabetical order or even in one contiguous block of memory. Information about ordering is preserved by a chain of pointers linking each record to the next in the alphabetical sequence. Suppose you want to add an entry for Roger Maris. You can put the record itself in any free block large enough to hold it; then you must link the new record into the chain of pointers. Starting with the first entry in the chain, you examine the pointer stored in Henry Aaron's record and find that it sends you to the entry for Grover Cleveland Alexander. Continuing in the same way, you eventually come to Mickey Mantle and find that the next link is Willie Mays. Maris's record belongs between these two. To insert it you alter the pointer in Mantle's entry, making it point to the record for Maris, and install within Maris's record the address of the entry for Mays. No data need be moved.

Sometimes even pointers are not indirect enough. Suppose an operating system has a table of information about a printer, which must be made available to all running programs. Each of the client programs—spread sheets, word processors and the like—could be given a pointer to the table, but then all the clients would have to be notified if the table were ever moved. A better solution is double indirection. Giving out a pointer allows the table to be changed or moved at will, provided there is a single memory cell that the operating system can keep fixed in place. Client programs are never given the address of the table; they have only the address of the fixed memory cell, where the operating system stores the current address of the table.

A doubly indirect reference to a memory location is called a handle. Higher-level programming languages make it easy to deal with handles. In Pascal, for example, a handle to the variable `score` is written as `^^score`; the notation makes it clear that a handle is merely a pointer to a pointer. When a program refers to such a variable, the Pascal compiler or interpreter issues instructions that twice invoke the machine's indirect-addressing mechanism.

In some programs there are even instances of triple indirection; that is, one address points to the location of another address, which points to the location of still another address, which finally points to the location of the data. There is nothing to stop the programmer from building still higher levels of indirection.

At this point it is easy to see just how mail delivery should be reformed. The postal service should give up pointers and adopt handles. Instead of your address, you should give your correspondent the address of a place where your address is stored. In this scheme everyone is assigned a unique identifying number—a handle. (By default your handle would be your social security number, but you might also be allowed to buy a vanity handle.) When writing you a letter I would put only your handle and a stamp on the envelope (plus, optionally, my return handle). The postal service would look up your handle in a central file, find the corresponding geographic address and direct the letter to the appropriate post office and carrier.

The advantages of postal handles would become apparent if you moved, or took a house on Nantucket for the summer, or launched a round-the-world tour. There would be no need to send out a heap of change-of-address cards. Instead, by filling out a single form at the post office, you could change the entry for your handle in the central archive. You might even make the change by telephone, and so arrange for your mail to follow when you're away for the weekend. Indeed, you might routinely have your mail forwarded from home to office, or vice versa, depending on where it is likely to reach you sooner.

This system might make life a bit easier for the sender as well. Address books would no longer be cluttered with crossed-out entries for nomadic friends. Magazine publishers, credit card issuers and others who conduct business through the mail would save the large sums of money now spent on processing change-of-address notices.

What's in it for the postal service? True, it would lose the revenue generated by all those change-of-address notices, but there would be compensating savings. For one thing, it would save the cost of hauling forwarded mail. Perhaps even more important in the long run, replacing addresses with handles would make it easier to devise a fully automated mail-sorting system. A machine should be able to read a single number far more reliably than it can parse a conventional mailing address, with all its components and variations.

Apart from easing the pains of mobility, postal handles would offer a measure of privacy and security. You could give your handle to the couple you met on vacation, or to a salesman, thereby enabling them to reach you by mail without giving them the means to look you up in person. Bill collectors could dun you only by mail. On the other hand, the fund-raising officer at your alma mater would find it much easier to keep track of you.

Many people resent the proliferation of identifying numbers in modern life, and they likely would find the notion of postal handles distasteful. Andy Rooney would not be enthusiastic. And there would be legitimate cause for concern. The postal service now acts only as a messenger and does not (presumably) keep permanent records of where we live; under the proposed regime, however, it would be the only repository of geographic addresses. Abuses are easy to imagine. An automated sorting system would allow certain snoopy government agencies to learn a great deal. Even if they were too fastidious to open envelopes, they could compile lists of correspondents.

Another hazard deserves mention. If a system based on handles ever went awry, the failure could be spectacular. Even routine errors would be more vexing than they are now. Under the present system misdirected mail is likely to end up with the next-door neighbors or perhaps with the people who have the same house number but live one street over. With handles, transposing two digits could send a letter anywhere at all; people with similar handles would generally not be neighbors.

In making this grandiose proposal I am about one-third serious. I don't expect to see anything come of it (I am not even sure I would welcome it myself), but I think the idea is worth examining.

The system should be technologically feasible. Storage space for the master file of handles and addresses would not be a problem. If every person and business in the United States were assigned a handle, a file capacity of 500 million handles would still leave ample room for growth. If each entry in the file were allotted 100 bytes of information—enough to represent 100 characters—the size of the file would be 50 billion bytes, or 50 gigabytes. A few large disk drives are all that would be needed to hold the file.

Where a bottleneck could develop is in gaining access to all those addresses. If each address received one letter a day, the system would have to look up, say, 500 million addresses a day, which works out to 173 microseconds for each access to the file. The disk drives might well respond quickly enough, but if every post office had to submit inquiries to a central archive, telecommunications channels would introduce unacceptable delays. The answer might be to establish a few dozen regional centers, each having its own copy of the master file; all mail would be hauled unsorted from the local post office to the nearest regional center and then be distributed from there.

So far indirect addressing has made its greatest progress not in the postal system but in the telephone network. With a cellular phone, for example, the broadcast signal is "handed off" from one transmitter to another as you move around a city or a region. Thus a caller need not know exactly where within a region you are in order to reach you. This approach is not quite equivalent to a postal handle system, however, because the telephone number is still associated with the instrument, not with the owner. If you forget to take your cellular telephone with you, it continues to receive calls, but *you* don't.

The 911 emergency network relies on another form of indirect addressing. When you dial 911 from almost anywhere in the country, you are automatically connected to the nearest emergency dispatcher. How does that come about? In effect, 911 is a handle rather than a pointer; when you dial it, the telephone system looks up the routing information needed to direct your call to the appropriate dispatcher for your current location.

Call forwarding is still another version of telephonic indirect addressing. In this case you dial in a code and then a telephone number, and thereafter all incoming calls are redirected to the specified number. Subsequently you can forward the calls from the second telephone to a third and so on. The one snag in this system is that all the record keeping and the redirecting are done by the local telephone office. If a call from Los Angeles to New York is forwarded to San Francisco, the signal must cross the country twice; a centrally administered network based on handles would take the obvious shortcut. Furthermore, under present telephone tariffs the call is billed as two transcontinental calls, not as one intrastate conversation. (Indeed, handles may not be widely used in the telephone system until the telephone companies adopt the postal principle that the cost of sending a message should not depend on distance.)

One can imagine the ultimate telephone network, based entirely on handles. You would have no need of special numbers for phones in your home, office, car or weekend retreat; you would have just one personal telephone number for wherever you happen to be. The system might work like this: A small radio transmitter on your key chain broadcasts an identifying signal to any nearby telephone, which in turn passes the information on to a central computer. When someone calls your number, the computer looks up your current location, and in a moment the nearest telephone rings. If you are a particularly busy and important person, you might enjoy walking past a bank of public telephones in an airport, making them each sound off in turn. But if you are that kind of person, please don't step into *my* office. •

Brian Hayes is the editor of American Scientist.