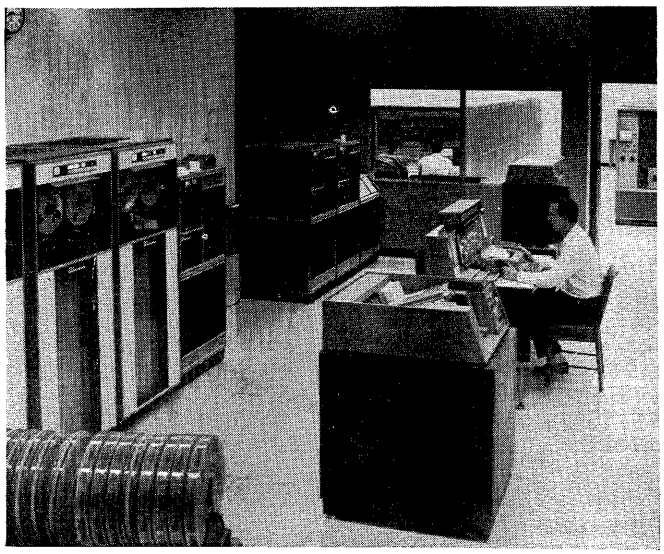
1961

SECOND IN A "PRIMER" SERIES ON MAJOR LOS ALAMOS PROJECTS AND FACILITIES



understanding

STRETCH

A Primer on Automatic Digital Computers

by John N. Savage

The author, while admitting sole responsibility for the following treatment, would like to thank Edward A. Voorhees, T-1, for patient and lucid explanations.

(*reprinted from* LASL NEWS, June 15, 1961)

The object of this primer is to answer two questions about automatic digital computers:

What are they?

How do they work?

The second question will take us inside Stretch, the Los Alamos Scientific Laboratory's newest computer. Stretch has been described by its manufacturer, the International Business Machines Corporation, as "dramatically faster than the fastest in existence . . . the most powerful computer ever built."

Words

All three words in "automatic digital computer" (a term that has certain advantages over anything with "brain" in it) are important and specific. Not all computing aids are true computers. Not all computers are automatic, and not all are digital. Let's begin by sorting things out.

Digital and analog—the difference

The two great families of computing aids are almost as old as the family of man. Picture Noah, if you will, on the first day of the big rain. He is sitting under a crude shelter on deck, watching the animals come aboard. The animals look hungry. Noah is wondering if he should send his sons into the field for one last load of hay.

Each time two hay-eating animals step on deck, Noah takes two dried beans from a heap beside his left knee and adds them to a growing pile of similar beans near his right foot. Later he'll consult the beans and see if he needs the hay.

The beans are a **digital** computing aid. They count, rather than measure.

Noah's worries are aggravated, of course, by the fact that he doesn't want his boys to drown for the sake of one load of hay, even if he needs it. Therefore he gets his wife to put an empty stewpot by his left foot, where a steady stream of rainwater from the shelter roof will fall into it. He hopes (maybe foolishly) that by watching the rising level in the stewpot he can get a useful notion of the speed with which the water is rising all over the world.

The stewpot is an **analog** device. It measures, rather than counts. The worldwide water level is represented by an analogous physical variable—the level of water in the pot—rather than by signals that represent numbers.

This difference stays essentially clear as we leave the primitive computing aids and approach the true computers. Modern digital computers do more than merely count, but they always solve problems by working with discrete signals that represent digits. Modern **analog** computers solve problems by measuring continuously-variable physical quantities—the voltage or amperage at certain points in the machine, the angle through which a certain shaft has been rotated, and so on. How **much**, rather than how many.

Most analog computers can accept problems in numerical form, of course, and produce numerical answers. (By putting marks on the stewpot, we can obviously make it count by quarts or inches.) The distinguishing fact remains that the operating quantities inside the analog device are variable in a continuous way, rather than by jumps across vacancy from one value to another.

Notice, by the way, that each bean (except when it's in Noah's hand, taking its jump across vacancy) has **only two** significant positions or "states." Each bean, by its presence in one pile or the other, says whether one animal has come aboard or not—yes or no. It is a two-state device.

We'll be seeing that term again.

When is a computing aid a "computer"?

Noah's beans are markers only. They cannot be said to compute anything, since he still has to count them all when he's through.

A real computer not only takes in information (problems and data). It performs reasonable operations on that information and produces an answer. If somebody were to hand Noah an abacus (probably a later invention, though very ancient), he could slide the captive beads in an appropriate way as the animals passed, and then he could read the total by noting the final positions of a relatively small number of beads. Or, if he knew that hay-eating animals had come aboard at the rate of 600 per hour for so many hours, he could use the abacus to multiply 600 by so many.

The abacus is a computer. It performs reasonable operations on information given it, and comes up with an answer.

The abacus is digital. It computes numerically, rather than by physical analogy.

The abacus is not, however, an **automatic** digital computer.

Where the "automatic" comes in

Automatic computers are distinguished by their ability to execute a coherent **sequence** of instructions (called a program) without stopping. At many points in the program, the next step to be taken may be determined by the results of previous steps in the sequence. Thus, in a sense, the machine may be said to select its own route to the final answer.

This kind of thing is less eerie than it may sound. The point to remember is that **every decision made by a computer is based on rules supplied to the machine by humans.**

Perhaps the best way to demonstrate the absence of magic in all this would be to follow a simple problem all the way through an automatic digital computer. What happens when a simple problem is brought to Stretch? What happens outside the computer and then what happens inside?

Getting the problem scheduled

In real life, the story of what would happen if a **simple** problem were brought to Stretch is short: Somebody would take the problem away again. It would be solved on some slower computer, or with pencil and paper. Stretch has better things (see appendix on Page 8) to do with its time.

Assuming that an exception has been made, and that we can schedule whatever we like, what shall we have Stretch do? Adding two and two wouldn't illustrate the most interesting aspects of the machine. What we need is a sort of **miniature** problem, intricate but small.

We have such a problem, as it happens, in Noah and his load of hay.

To anyone who knows how different this problem is from Stretch's usual diet, the choice may seem unpromising. Nevertheless, Noah's problem of whether or not to send his boys ashore meets important conditions: It is easy to understand, and its solution by Stretch can be made to illustrate each of the procedures we need to look at, both outside and inside the machine.

Noah's problem has just been scheduled. Let's follow its handling, step by step.

Determining what data are necessary

A great deal of preliminary work will have to be done by a man before the machine comes into the picture. The man who determines how to ask Stretch about the hay will be a carefully trained human called a programmer.

The programmer will begin work on the Noah problem (just as he would on one less simple) by sitting down at his desk to do a little common-sense planning.

He knows already that the desired answer is a simple yes or no to one question: Should Noah send his boys for another load of hay? (Actual computer answers are sometimes long enough to fill a book.)

The final yes or no will be determined by the answers to **two** questions: Is the hay needed? Can the boys get it safely? Unless both answers are yes, the boys will stay on the ark. (We have seen how Noah worked on the same two halves of the problem, using a digital approach to the "need' question and an analog approach to the "safety" question. Stretch will handle both questions digitally, since it's that kind of computer.)

Having seen that Noah's problem has two parts, the programmer may reach for pencil and paper and start on the first half—the one about need. He might begin by making an idealized list. What information would he **like** to be able to supply to Stretch?

He might write:

Data

Tons of hay already aboard when question arises (A)
Tons of hay the animals will consume per day (H)
Days the animals will spend on ark (D)

That's all he needs. If accurate numbers corresponding to A, H, and D can be supplied to the machine, the machine can produce a yes-or-no answer to the question, "Is more hay needed?"

Perhaps our programmer next makes a similar list for the "safety" half of his problem. It might look like this:

Data

Water rise, in vertical inches per minute, toward lowest point on road between ark and hay field (W)
Vertical inches still remaining between water level and lowest point on road when boys set out (S)
Minutes required for boys to get hay and return past lowest point (P)
Greatest depth of water safely fordable by hay wagon (N)

With this information, if all of it can be obtained, Stretch can figure out a yes-or-no answer to the question, "Can the boys get the hay safely?"

The two lists together call for seven numbers. Let's save time by making the barefaced assumption that most of this information turns out to be available. The programmer has access to a newly unearthed archeological treasure called "Shem's Diary," and it tells him everything except Item H—the tons of hay to be consumed on the ark each day.

This little gap in the data makes quite a difference.

If all seven numbers had been available, the problem could have been solved by the programmer's nine-year-old daughter. With H missing, the problem can benefit by the services of a machine.

The computing job has grown. It has three parts instead of two, the first part being to calculate H, the daily hay consumption on the ark. (The safety question, with all its data available, is now a cinch. It ought to be done first, since a negative answer to it would settle everything. Our man will ignore this possible shortcut, though he wouldn't ignore it in real life. Let's say he has grown interested in programming the whole problem for its own sake, or for ours.)

Calculating the tons of hay consumed each day on the ark will require a data list of its own. A look at that list will give us a preliminary hint about the one big advantage Stretch has over a human arithmetic champion.

Stretch is faster—several million times faster. Let's see how that affects the data list.

When last we saw Noah, he was about to count some beans. Presumably this count would give him an accurate "total number of hay eaters" for his mental data list. After that, he would have to make a rough guess at how much hay would be eaten by that many animals each day. He might make this guess by intuition, or he might go so far as to take one middle-sized hay eater's estimated daily consumption and then multiply by the total number of hay eaters. To refine the guess any further than that, he'd have to estimate daily consumption for **each species** and then add up the estimates. This would give him a more nearly accurate answer, but he certainly wouldn't attempt to do it. Life is too short.

The Stretch programmer can afford to take a different view. His machine is going to solve the problem in seconds, even if he complicates it quite extravagantly. He is free to consider each species, and he needn't stop even there.

Instead of just seven numbers, he now needs a dozen

lists of numbers. The following version of what he might write is perhaps a little extreme, but it will make our point:

Data

Probable daily hay consumption, one adult pair, each species, when healthy (List No. 1)
Same when seasick (List No. 2)
Susceptibility to seasickness, each species—expressed as probable days seasick per 100 days at sea (List No. 3) . . .

And so on. Probable deaths from all causes, probable births, effects of seasonal changes in appetite—these and still other factors might be considered. Notice that the arithmetic isn't really getting any **harder;** it's just that there's more of it.

Stretch won't mind that.

Planning a program

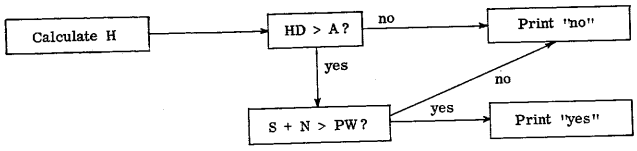
Let's say that the programmer now sends a squad of assistants to the library to make up his lists, and let's assume that all the facts (on appetites, death rates, and so on) turn out to be available. Our programmer gets his data (though still in a language Stretch wouldn't understand); what else does he need?

Along with the data, Stretch will have to be given **instructions** about what operations to perform on the data. The full set of instructions for the Noah problem will constitute that problem's "program."

As we shall see when we get inside the machine, each instruction covers a very small step in the program. Fortunately, the programmer doesn't have to worry about every small step. All he has to produce is a general plan, plus a little specific guidance; then Stretch (or even some other computer) can assemble a complete Stretch program by putting together appropriate ready-made sets of instructions selected from among those on file.

The general plan for solving the Noah problem might go like this: Calculate H (tons consumed on ark per day). Then answer the "need" question. Then, if the answer is no (meaning no more hay is needed), print "no" and stop. If the answer is yes, proceed to answer the "safety" question. If the safety answer is no (meaning the boys cannot safely get the hay), print "no" and stop. If the safety answer is yes, print "yes" and stop.

Rather than write out the preceding paragraph, the programmer might jot down a preliminary "flow chart" that says the same thing. For anyone who knows that ">" means "greater than," the chart may be clearer than the paragraph was:



The job of calculating H will be many thousands of times bigger than all the rest combined. The programmer will have to give Stretch quite a lot to go on. Probably he will begin by writing out a fairly involved equation, fundamental to the determining of H. The equation will not be made of numbers; instead it will use symbols representing "consumption when healthy," "consumption when seasick," and so on. It might end with "=Y," Y standing for total hay consumption by one species of animals while on board. Stretch will have to solve this equation over and over again, once for each species, using appropriate numbers from the lists each time and getting a different answer for Y each time. Then it will add all these answers together and get H, before proceeding to the easy parts of the problem.

Stretch will perform its complicated task by steps—very small and simple steps, taken at the rate of hundreds of thousands per second. The flow chart is a handy way of summarizing these steps, but it is not the kind of thing Stretch can read. The programmer now puts his program into the form of a penciled list of commands, more detailed than those on the flow chart and less easily comprehensible to the layman. The steps on this list are still much bigger and less numerous than those the machine will ultimately take; each step still needs to be analyzed further, broken down into smaller components. But Stretch itself can do that job.

Once the programmer's pencil work is finished, can it be tossed into some kind of slot in Stretch? Not quite. Stretch has facilities for several modes of input, not including that one.

Our programmer hands his program to a key punch operator. After a few minutes' work at a keyboard similar to that of a typewriter, the operator hands him a deck of rectangular cards. What the programmer wrote, a moment ago, now appears as a scattering of rectangular holes, arranged according to a pre-determined numerical code.

The program cards could be fed into Stretch, but our programmer prefers to use magnetic tape, one reel of which can hold instructions for solving several problems in addition to Noah's. Therefore, the cards are now fed into a device that "reads" their patterns of holes and "writes" a pattern of magnetized and unmagnetized spots in the metallic coating of a plastic tape.

When the proper time comes, Stretch will "read" the magnetic tape, analyze the instructions on it, and form a detailed, step-by-step program to match.

Preparing the data for input

The program tape is now ready for use by Stretch. The data lists are not. They consist mainly of numbers specifying the hay consumption, mortality rate, and so on, of each species. There are thousands of numbers in all. Before the programmer made his program tape, he had to decide in detail how the input and storage of these numbers would be handled.

For input, he has decided to use one punched card for each species. (If one card wouldn't hold all the necessary numbers, he could use two or more—but we're keeping this simple.)

He hands his data lists to key punch operators, who follow his instructions and produce one zebra card, one elephant card, and so on. The cards all look alike, the only difference being in the pattern of holes. This pattern represents, for each species, a single row of several dozen digits, specifying various characteristics of that species. Birth rate information, say, begins with the thirteenth digit and runs through the sixteenth, on each card.

When the data cards are finished, they and the program tape are carried to the computer. Noah's problem and Stretch are about to meet for the first time.

Input and output channels

Communications in both directions between Stretch and the rest of the world normally flow through eight electrical "channels." Some of the channels handle flow in both directions, some in only one or the other. We are about to use two of the input channels.

We mount the program tape on a tape reading device at the outside end of one of the channels that lead into the computer. We put the data cards in the rack of a card reading device at the outside end of a second input channel.

Meanwhile the computer is going about its business, solving a problem for the man who was ahead of us on the schedule. A card punch and a rapid printer, both being fed by Stretch itself through two **output** channels, are recording answers to the computation now going on. The card punch is delivering 250 finished cards per minute. The printer is printing 132 characters at a whack, ten whacks per second.

Stretch seems busy enough already. Does that mean we have to wait? Not at all. Stretch is no ordinary computer. We go ahead and press the button that starts the tape reader.

(This account will include more than the normal amount of button-pushing, for the sake of keeping procedures clearly defined. In actual practice, the starting and stopping of various devices is more often handled automatically, through programmed commands.)

The tape runs past a "reading head" and onto another reel. The reading unit translates the tape's magnetic pattern into a pattern of pulses (sharp changes in voltage) and sends the pulses along an input channel to the Exchange unit of the computer.

The Exchange

The Exchange is the input-output routing center of the computer. In addition to most of the characteristics of a telephone exchange, it has some of the characteristics of the distributor head on your car. Just as the distributor serves each sparkplug in turn, fast enough to keep them all working, Stretch's Exchange serves all input and output devices in such rapid sequence as to take care of them virtually all at once.

Pulses from the tape reader are now being received by the Exchange. These pulses carry, in numerical code, the programmer's plan for solving Noah's problem. The plan cannot yet be executed, because it isn't stated in the right kind of detail and because it calls for data the machine doesn't yet have. The plan will have to be passed along to another part of Stretch, where it can be stored (or "remembered") until the proper moment.

Storage

How can hardware remember anything? It can't, except in a fairly special sense.

Flip a coin. It comes up heads. Unless you turn it over, it will still say heads tomorrow. It will remember "heads," and divulge "heads" if consulted, for as long as you care to leave it in the "heads" position, or "state."

This gets more interesting with more coins, since the possible heads-tails combinations grow in number. Assign meanings to the combinations, in terms of any code you like, and a sufficient number of coins will remember your birth date or the Gettysburg Address.

Each of the coins, like one of Noah's beans, is a two-state device. Computer memories use two-state devices, with electronic ways of putting them into one state or the other, as well as of testing to see which state they are in.

Such devices, when you have them in quantity, can store any kind of information at all. Modern digital computers store mostly numbers. Often, but not always, these numbers are in a form called **binary,** which will be explained in a moment. Binary digits are called "bits" for short.

The Stretch computer's storage facilities are remarkable, both for their capacity and for the speed with which information can be put in or taken out.

One part of Stretch's memory contains millions of two-state devices called magnetic cores. These are tiny doughnuts of iron oxide ceramic, each capable of being magnetized in two different polarities. The magnetic core memory can hold 98,304 "words" of 64 bits each, for a total of six million and some bits (not to mention eight extra bits per word, used for error detection).

Another part of Stretch's memory consists of 39 large discs, like oversize phonograph records, with space for magnetic storage on both sides. The disc memory has a capacity of more than two million 64-bit words. It can deliver 125,000 words per second.

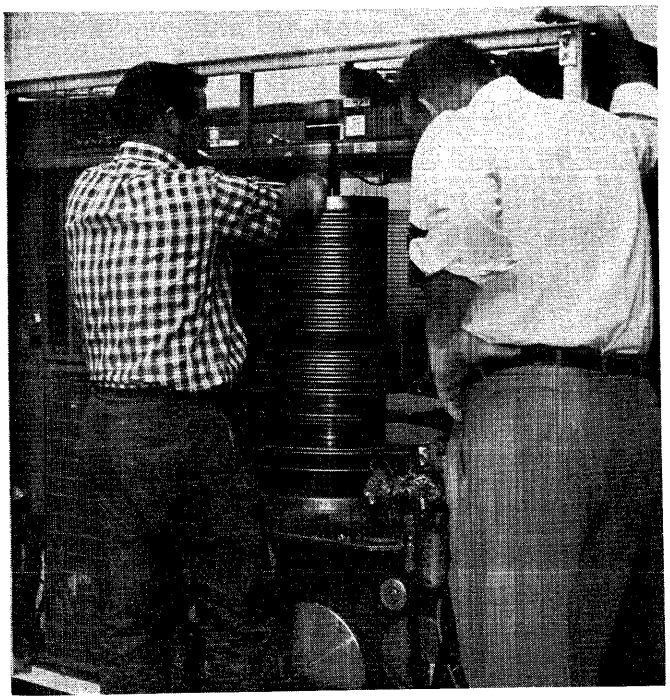
In less than a second, even while an arithmetic calculation is proceeding, the core memory can dump (a technical term meaning transfer in perfect order) **all** its data into the disc memory.

Is it still possible, among all these 64-bit words, to find the one that's wanted? Absolutely. The trick is done by means of a system of addresses, just as in a big city. Each word-location in the core memory, for instance, has its own identifying number. This number serves as the address of whatever word may be roosting there.

Much more should be said about computer memories, but perhaps not in this primer. Let's get on with Noah's problem.

The instructions from the program tape are passed along by the Exchange to the magnetic core memory, where they wait. The card reader is then switched on. It begins flipping cards at the rate of 1,000 per minute, reading the pattern of holes in each card and sending the data (again in the form of pulses) to the Exchange. The Exchange passes this information along to the core memory for storage.

The core memory is plenty capacious for this problem. We are using only a fraction of Stretch, and it may be that somebody else is loading the disc memory (through other input channels and the same Exchange) while we proceed.



Storage discs of the Stretch computer.

Essentially, we are storing the "image" of each data card in the core memory. The significance of each hole in one of these cards depends entirely on the location of the hole, since all the holes are alike. If the zebra number begins 6340952. . . ., the card's first column on the left will have a hole in the sixth vertical position, the second column from the left will have a hole in the third vertical position, and so on. To store the zebra number, using our two-state devices, we might assign one core to each possible hole location on the card, then let one polarity represent "hole" and the other polarity "no hole."

This way is not very efficient, since it uses ten cores (one saying "hole" and nine saying "no hole") for each digit in the number. We have chosen, in loading Stretch for the Noah problem, to store the numbers in a form that is unnecessarily bulky. Before the actual computation begins, we'll have Stretch convert these numbers to a form in which they can be stored more compactly.

Binary Notation

Binary notation, in which every digit is either 1 or 0, is beautifully suited to machines that use two-state devices. It not only makes storage easy; it makes **arithmetic** easy.

The kind of notation you use every day (and the kind in which we have just stored the Noah data) is called "decimal" notation, because it is based on the number **ten.** It uses ten distinct digits, from 0 through 9. A "place-value" procedure permits the writing of numbers larger than nine. In decimal notation, "1000" means exactly one thousand because it has a zero in the "ones" position, a zero in the "tens" position, a zero in the "hundreds" position, and a one in the "thousands" position. The position values get ten times bigger with each step to the left.

Binary notation works exactly the same way, but on a base of **two.** There are only two digits, 0 and 1. Position values get only twice as big with each step to the left. In binary notation, "1000" means exactly eight, since it has a zero in the "ones" position, a zero in the "twos" position, a zero in the "fours" position, and a one in the "eights" position.

Decimal notation uses a "decimal point," with place values to the right of it continuing the progression set up on the left, becoming "tenths," "hundredths," and so on. **Binary** notation uses a "binary point" in exactly the way you might expect, place values to the right being "halves" "quarters," "eighths," and so on. To register "three and five eighths," we write "11.101," which means two plus one plus one half plus no quarters plus one eighth.

That's all there is to binary notation, though you may feel like trying it on some telephone numbers if you haven't seen it before. Seven is 111 (four plus two plus one), and if you place three coins heads-up they'll remember seven forever. The binary 10010001100001010 is a longish number, but much more convenient for Stretch to remember than the equivalent 74,506.

Not only storage, but arithmetic, gets easier with binary notation. We'll see how in a moment, when the computation begins. Before that happens, the program has to be put into the final form in which it will be executed. Neither this nor the conversion of data to binary notation can be done in the memory. Both operations could have been performed by some other computer, and the assembled results loaded into Stretch, but we have preferred to leave it all up to Stretch's own Central Processing Unit.

The CPU

The part of Stretch that does the arithmetic (and many other things besides) is called the Central Processing Unit, or CPU. While we were loading the core memory, the CPU was busy producing answers for somebody else.

Now the CPU has finished the other man's problem. It signals for more work. The Noah program is electronically sent to the CPU, a piece at a time, from the core memory. Complicated circuits in the CPU, working in accordance with an "assembly" program, analyze the steps of our programmer's plan. Each step is broken down into a simple sequence of commands, put into appropriate numerical language, and returned to storage. When the whole program in final form has been thus assembled and stored, the CPU starts executing it.

The first task on the program turns out to be conversion of the stored data into binary notation. Therefore, following a set of instructions it uses almost every day, the CPU converts the data. This does not mean that all the zebra information is converted at once, even though we stored it in a form equivalent to one long string of decimal digits. This string was really **several** items of information. What happens now is that the CPU calls for the number representing "healthy zebra hay consumption," converts it, and sends it back to storage. Then the number representing "seasick zebra hay consumption" gets the same treatment, and so on. All the information on all the animals is converted into binary notation in seconds.

But we seem to be in danger of going so fast that nothing gets explained. Let's make the assumption (unlikely but possible) that when the conversion to binary is finished, the next command on the program is to stop and wait for the operator to push a button.

This gives us a chance to emphasize an important point.

Stretch has no information on the **meanings** of the numbers it is juggling. Stretch is hardware. We have fed into it certain complex patterns of electrical pulses. These patterns travel about within the machine, changing one another in ways determined by the machine's designers and our programmer. Stretch cannot be said to know what the programmer is up to. Stretch lives in a sublime and abstract world of ones and zeroes. Like all other digital computers, Stretch has total freedom from the concrete meanings of the problems it solves. This is the secret of its versatility.

Computation

The operator will soon press a button. The CPU will consult the stored program and cause itself to perform the next step, followed by the rest in order.

What are these steps?

They are numerous and small. One conceivable series of commands might go as follows (though this language is not the machine's): "Fetch the word at address 86957 and put it in Register 8 (part of the CPU's arithmetic unit). Ignoring the first nine digits of the word, divide the next six digits, treated as a single number, by the corresponding bits of the word whose address is 00050. Store the unrounded quotient in Memory Location S."

Those commands, detailed though they sound, involve yet smaller operations. The command to divide, for instance, entails a great many steps in itself.

Must one step end before the next can begin? In most computers, yes. In Stretch, no. One reason for Stretch's unrivaled speed is that steps can often overlap. For instance, as soon as one step is begun, Stretch "looks ahead" and performs the "fetches" (and some other types of activities) necessary for the next.

How hard is the arithmetic these steps perform? It is quite easy.

In most ways (including perhaps some unexpected ones), binary arithmetic is exactly like decimal arithmetic. Mul-

tiplying 11 by 101 requires exactly parallel procedures, whether it means "eleven times one hundred one" or "three times five." The radix (base) of notation determines the numerical meaning of the marks, but doesn't change the rules for operating on the digits. (The rule about "carrying" is not a real exception. You carry whenever the highest digit in your system of notation is lower than the quantity you want to write.)

So binary arithmetic resembles, procedurally, the arithmetic you do yourself. What makes binary arithmetic so much easier is that it never involves digits higher than 1.

Let's see what this means to an electronic computer (which can be thought of as using built-in "tables" of multiplication and addition):

When a human being multiplies any string of digits by another string of digits, he does it by dealing with two digits at a time, following a memorized procedure and using memorized tables of multiplication and addition. The tables state the results of multiplying or adding all possible two-digit combinations. As you may remember from the weeks when you were learning them, the tables used in the decimal system are formidable. Each of them has to account for dozens of possible combinations.

The combinations in **binary** notation are 0 and 0, 0 and 1, 1 and 0, 1 and 1,—and that's all.

The whole binary multiplication table can be reduced to a single rule: "One times one is one; any other product is zero." Electronic circuits that will produce an "on" response when confronted with two "ons," and an "off" response otherwise, are correspondingly simple. Circuits for subtracting, adding, or dividing are more complicated (those for dividing, especially), but they are far simpler than those that would be required for decimal numbers.

One neat trick deserves special mention. Frequently the effect of one binary number acting arithmetically on another is simply to shift the position of the "binary point" (equivalent, you'll remember, to the decimal point) in the number being acted on. This effect can be useful. The binary point, of course, can't be registered **as such** in the machine anyway, since it's not a digit. If its position in a given number is specified by certain digits stored with the number, as it is in "floating point" operation, then its position can be changed by changing those bits.

Stretch, unlike some computers, can operate in either the "floating point" way or the "fixed point" way. Floating point operation not only permits automatic "point shift" arithmetic; it also makes it easy to retain the most significant part of any number that grows so long as to need "rounding off."

All right. Let's get the operator to push that button.

This is the moment to which all the programming, coding, and data preparation have led. Stretch is now ready to start plodding through the millions of successive tasks involved in solving Noah's problem.

The operator presses a "start" button, steps to an output printer a few feet away, and tears off a sheet of paper. There is a "yes" or a "no" on the paper. It got there about the time he did.

How can anything work so fast?

Aside from the special features already mentioned (the huge fast-access memory and the ability to "look ahead"), one other feature helps to account for the speed of Stretch. It is called "indexing." Many modern computers have it, but none in so powerful or flexible a form as Stretch. "Index registers" in the CPU, using a relatively simple succession of subtractions or additions, make possible a great reduction in the number of coded steps necessary when the same equation (like the one we used for finding H) has to be used over and over again.

Aside from all these features, one fact is central in accounting for Stretch's speed, and for that of most other giant computers: There are (with exceptions in the input-output areas) **no moving parts.**

Early calculating machines used hand-propelled gears and ratchet wheels. More recent kinds used electricity (fast enough in itself) but controlled it by mechanical switching (very slow). Then came electromagnetic relays, which were still metallic switches of a sort, but faster.

Real speed in computing began with the employment of vacuum tubes. Electron traffic could now be controlled by other electron traffic. Nothing mechanical had to move.

By using transistors instead of vacuum tubes, "solid-state" machines like Stretch and others have made a further improvement in computing speed—becoming more compact and easier to cool at the same time.

The sources of error

How much faith can we put in answers so hastily arrived at? Several factors (**not** including the haste of the machine) are legitimate reasons for skepticism.

Stretch, like other computers, has no facilities for improving the accuracy of the data given it. Errors in the reference books from which the data came, like errors in the transcribing of such data, cannot be detected.

Moreover, if the programmer makes a mistake, specifying a procedure that will produce a wrong answer, Stretch cannot be expected to set him right.

Finally, a number in the computer sometimes grows so long that it has to be "rounded off." This can often be prevented by careful programming or by letting the number spill over into a second word, but there are times when an infinitesimal "rounding-off" error must be accepted.

Stretch has various ways of checking on its own correctness. For instance, each 64-bit word is accompanied by eight "checking bits." Stretch will detect, and often automatically correct, almost any error resulting from malfunction.

Is Stretch a brain?

Stretch is a tool for solving problems similar to some of those solved by the human brain. Though the popular nicknames are bound to catch up with it, Stretch is not a brain.

The analogy is a fascinating one, all the same. It has been considered in some detail by the late John von Neumann (**The Computer and the Brain,** Yale University Press, 1958), as well as by A. J. Turing, who wrote **Can Machines Think?**

Turing says they can, potentially, but his definition of thinking should be examined.

Both in what they are and in what they do, brains and computers have points of similarity. An obvious one is that they both "remember" information. Another is that each nerve cell in the brain likes to respond in the "all or none" fashion characteristic of two-state devices. Computers (following programs devised by ingenious humans, of course) have succeeded in composing music, translating prose and poetry, playing chess and checkers (the first rather badly), and so on. They are unquestionably the most brain-like tools man has ever devised.

Fascinating problems of consciousness, initiative, imagination, emotion, esthetics, and even religion pop up when the analogy between brain and computer is pursued. Since this is no place to pursue it, the best thing to say is that there is a similarity and a difference. This primer will stop there.

APPENDIX: SOME COMPUTER APPLICATIONS AT LOS ALAMOS

(*reprinted from* LASL NEWS, May 4, 1961)

With three IBM 704s and the MANIAC (Mathematical Analyzer, Numerical Integrator and Computer) running almost continuously, night and day, for the past several years, LASL has earned a reputation as one of the most advanced computer centers in the nation. Now, with the recent addition of an IBM 7090 and the super computer Stretch, the Laboratory has become one of the most advanced electronic computer centers in the entire world.

Why does Los Alamos require such remarkable computing facilities?

What impact will Stretch really have on Los Alamos programs?

MODELS

Electronic computers frequently work with what is called a "mathematical model" of something in the real world —a mathematical model being a system of numerical relationships that approximates the actual physical relationships in a real-life state of affairs. By means of mathematical models, a computer can be made to predict, or "imitate," the functioning of anything from an egg-beater to a thermonuclear bomb. The imitation can be more or less crude, more or less acurate, more or less useful to the designers of actual eggbeaters or bombs—depending on how detailed the mathematical model is. Computer speed and memory capacity are important because the more details the machine can handle, in a reasonable time, the more refined the mathematical model can be, and the more useful the results.

Studying real or hypothetical hardware by means of a computer is a little like studying physical events with a camera, in that a great deal depends on how much detail our picture shows. The word "resolution" is used, in both photography and computing, to mean clarity of details. One way of describing the difference Stretch will make in LASL computing facilities is to say that Stretch will make possible at least a threefold improvement in resolution.

In addition to this improvement in quality, the *quantity* of Stretch's production in any given time will be at least three times that of all the rest of the computers at Los Alamos put together.

NECESSARY

In terms of LASL programs, what makes Stretch necessary?

Both military and nonmilitary aspects of Los Alamos work will benefit from Stretch. A single design study for a new thermonuclear warhead, as performed on the existing 704 computer, requires something like a billion arithmetic calculations. It takes about ten hours. On Stretch, a higher-resolution (more realistic) study of the same device, involving perhaps thirty times as many arithmetic operations, can be performed in the same time. The usefulness of higher resolution is obvious, but the usefulness of the greater speed may be less so, at first glance. We needn't design a new warhead every ten hours; what good is the extra speed?

The answer lies in the fact that for every hypothetical warhead that becomes an actual device in the nation's defense arsenal, dozens of not-quite-so-good designs have to be studied and rejected. The whole direction of research and development on weapons will always be heavily influenced by the kind of mathematical trial-and-error that is made possible by the existence of fast computers. Even in the absence of the current ban on nuclear test detonations, it would not be feasible or desirable to obtain all the necessary information by experimenting with actual bombs. Mathematical models will always be used extensively, for the purpose of saving time and money in guiding experimentalists toward areas where experiments will do the most good.

PROJECT ROVER

In its many peaceful programs, LASL finds fast computers an equally indispensable tool. One good example is Project Rover.

A rocket reactor must be extremely powerful for its weight, and therefore must operate at very high temperatures. It must also be capable of rapid and delicately controlled changes of power. In the face of these and other requirements, designers have to answer such questions as these: How big should the fuel volume be, and what shape? What kind of fuel should be used? What reflector? and so on.

The answers depend on the tiny particles called neutrons, some of which will remain in their parent atoms and some of which will fly about inside the reactor, in all directions, at various speeds, by the billions. The designers need to know how many neutrons will be where, under certain conditions, and what their speeds will be.

The factors that will determine the neutrons' behavior are known, but they are very complex and numerous. Detailed predictions of neutron traffic at given points in the reactor entail a huge number of calculations. No single step in these calculations would be difficult for a good arithmetician with a pencil, but he might need several lifetimes to do them all. If there were no machines for performing such calculations, the only feasible way of getting the answers would be to build the reactor *without* the help of detailed predictions, and then study the neutrons. Perfecting a reactor design by these build-and-study methods would be a lengthy and far more expensive process.

DOLLAR SAVER

Reactor design studies have been performed at Los Alamos as a matter of routine for years. The advent of Stretch will make it possible to improve the realism of such studies and to increase their number, with a consequent saving in the taxpayers' dollars that might otherwise have to go into experimental hardware.

In Project Sherwood, the behavior of high temperature plasmas under the influence of strong magnetic fields presents problems so complicated that the equations for solving them have, in many cases, still to be created. It is likely that such problems in magnetohydrodynamics will become a part of Stretch's diet in due time, and that the high degree of resolution possible with Stretch will turn out to be as useful in Sherwood as elsewhere.

The Laboratory's Health Division has made frequent use of LASL computing facilities, notably for data reduction in connection with experiments on laboratory animals. Interspecies correlations by which experimental data from animals can be extrapolated to humans are made more meaningful by mathematical techniques involving electronic computers. One long-range plan (for the establishment of radiation dose contours throughout the body of a life-size plastic man) will almost certainly require the extreme speed and capacious memory of the Stretch computer itself.

The applications mentioned above are examples only, and numerous others could be listed. The present and expected quantity and complexity of computing jobs at Los Alamos are such that further expansions of the machine facilities, after the arrival of Stretch, are already being considered. In a vigorous scientific institution, there can never be a shortage of problems.