LGP-30 General-Purpose
Digital Computer

Here it is—a clear and thorough introduction to the theory, design, philosophy, programming and use of general-purpose digital computers—based on the new LGP-30. This is a much-needed contribution to the literature of the computer field.

STANLEY FRANKEL
and
JAMES CASS
Librascope, Inc.

Historical Introduction

MANY early digital computers were planned to perform fairly specific tasks. An example is ENIAC (Electronic Numerical Integrator and Computer), completed in Philadelphia in 1946. It was the first of the large-scale electronic digital computers. ENIAC was built to compute shell trajectories and prepare firing tables. It was, by present standards, a very limited machine. It held the numbers involved in its calculations in ten units called "accumulators," each of which could hold a ten decimal digit number and its sign, and also had a certain degree of arithmetic ability. Arithmetic operations were performed by passing numbers back and forth among these ten accumulators. The successive operations were controlled by auxiliary units called "program control units." For each operation two or more of these came into action and took over control of some of the accumulators and/or various other operational units. Instead of interconnecting the program control units in a permanent fashion to determine the entire sequence of operations, it proved more convenient to provide plug wires by which they were interconnected. As a result of this plug-wire system of interconnection, ENIAC proved to be a much more widely useful machine than its original directive called for. There were, however, two essential limitations on the class of problems it could handle, both having to do with what we might call its "memory capacity." It could only "remember" ten numbers and about one hundred steps of instruction. Complex problems could not be fitted to this limitation.

In the period following the completion of the ENIAC, designers tried to provide greater flexibility by seeking various kinds of bulk memory storage, both for numbers to be operated on and for the instructions which were to control these operations. It soon proved possible to provide memory devices holding a few hundred to a few thousand numbers, using equipment no more complex than was required for ENIAC's memory of ten numbers.

There then remained the problem of providing capacity for the storage of a great many instructions. It was soon recognized that instructions could be stored in coded form in the same way that numbers are stored, and in the same devices. The computer can then be allowed to draw these instructions from the memory store one by one, as needed. These two developments broke the bottleneck of memory capacity, and made possible computers capable of performing almost any imaginable kind of arithmetic calculation.

Most of the large-scale electronic computers that followed ENIAC are of such flexibility that they have come to be called "general-purpose" computers. At first, it might appear that the finite, though large, memory capacity of each machine would impose a definite limit on its range of use. As we will show later, that limitation is easily evaded, so that any such computer with a large memory is just as flexible as if its memory capacity were infinite. Since that limitation can be evaded, these general-purpose computers may justifiably be called "universal computers" in the sense introduced by A. M. Turing in 1936. The word "universal" is not used idly. It is difficult to say precisely how broad is the "universe" of computable problems, chiefly because it is difficult to describe a process which could be called a "computation" which could not be performed by a universal computer.

Necessary Operations of a General-Purpose Computer

The process of setting up a general-purpose computer to carry out some desired process consists of choosing and setting into the computer's memory an appropriate sequence of instructions and numbers; its *program*. Each instruction causes the machine to carry out one of its basic operations. The process of planning such a sequence of instructions is called *programming*. We now consider the question of what must be included in the set of elementary operations available for choice; in other words, what should be the vocabulary of the language of the computer.

The vocabulary must include some arithmetic operations. The computer must be able to add, subtract, multi-

Presented at the Computer Clinic, Second International Automation exposition, Chicago, Nov. 14-17, 1955.

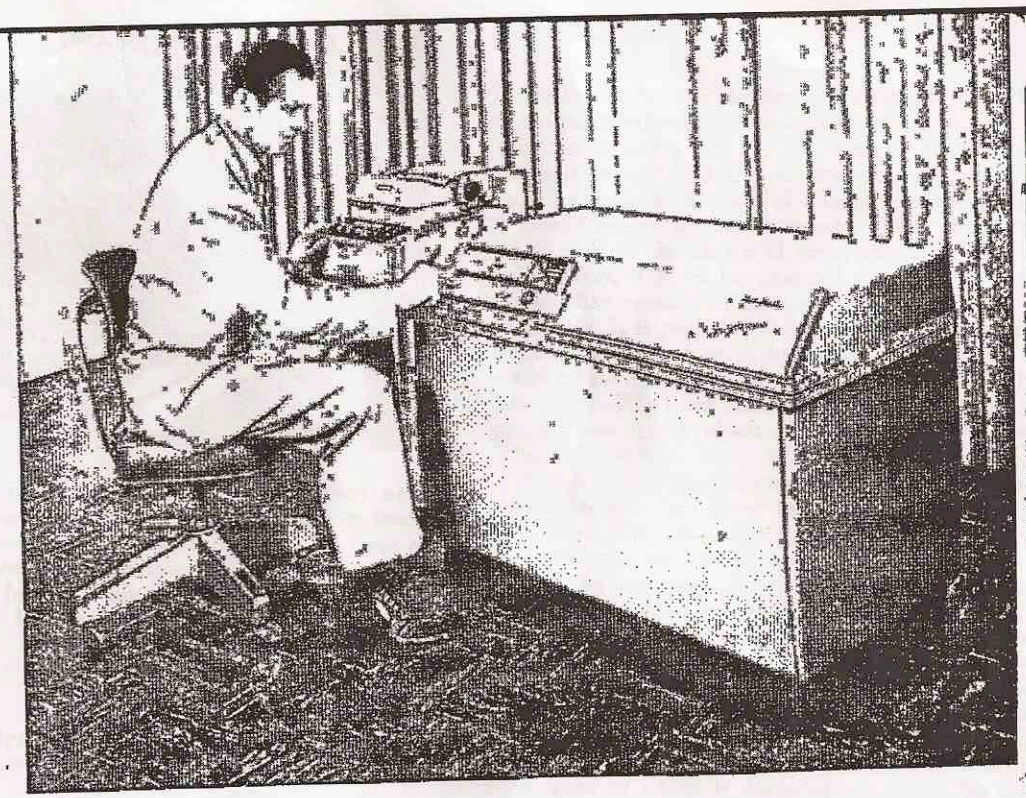

Fig. 1. The LGP-30 general-purpose digital computer is a mobile, serial, single-address, stored-program digital computer.

ply, divide, extract roots, form logarithms, exponentials, etc. However, not all of these need to be included in the vocabulary of elementary operations because the more complex operations may be built up as sequences of elementary steps. Most computers now include only the first four of these operations—add, subtract, multiply and divide. In a few computers some further processes have been included (e.g., ENIAC included square root), in some others less than this set has been used (e.g., one high-speed electronic computer does not have built-in division). The fewer arithmetic processes a computer is required to perform as elementary operations, the simpler its design can be. On the other hand, additional elementary operations may prove a convenience to the user and can increase the over-all speed of computation. It is generally considered that the inclusion of the four primary arithmetic operations provides a good compromise between these conflicting pressures, and that is the choice made in the design of the LGP-30.

Let us consider a situation in which an addition is to be performed. The computer reads from its memory an instruction which includes the order, ADD. That alone is not a complete instruction. The computer must also be told what two numbers are to be added. In the LGP-30, as in many general-purpose computers, one of the two numbers is that which is held in a specialized unit of the memory called the *accumulator*. The second number is designated by including in the instruction a specification of the part of the memory in which it will be found. The memory is divided into units, called *words*, each of which is identified by a number called its *address*. One address is included in each instruction to identify one of the two numbers used in the arithmetic process. The other number is that previously held in the "accumulator." At the end of the operation the arithmetic result shows up in the accumulator. This method of specifying operations is called a *one-address code*. Other systems using two or more addresses in each instruction have also been used widely.

As the number resulting from an arithmetic operation is not always to be used immediately in another arithmetic process, some means must be provided for placing the number held in the accumulator in storage in the general memory. This is done by RECORD orders.

After an operation has been completed, the computer must read from its memory the instruction which is to be obeyed next. Some system must be provided for determining where that next instruction will be found. In many computers the system normally used is that the instructions to be obeyed in sequence will be found in successively numbered memory locations. However, some exceptions must be provided to this normal sequence of instruction words. These are provided by CONTROL TRANSFER orders. Such an order includes an address at which the next instruction will (or may) be sought. Various sorts of *conditional control transfer* orders are provided, which do or do not effect this transfer, depending on the results of previous calculations or on external circumstances. The conditional control transfer orders permit the computer to make "decisions"– that is, to follow one or another course of action depending somehow on information at its disposal.

We have now described all of the operations essential to the internal economy of a general-purpose computer, including (1) arithmetic operations, (2) control transfer to obtain instructions, and (3) record the results for the next operation. In addition, some means must be provided for setting numbers and instruction into the computer memory before it starts its work, and some way for it to report out results. A great variety of such "input-output" mechanisms have been used. Results of calculations, or intermediate data, may be pumped out on punched paper tape, on magnetic tape, punched on cards, etc., or may simply be typed on paper, and may at a later time be re-inserted into the computer if further processing is required. As there is no limit to the number of rolls of tape or sheets of paper which could be provided, we may regard the available memory capacity of any general-purpose computer as indefinitely large. It is this theoretically infinite memory capacity which justifies our calling these computers "general purpose." Any such computer is a "universal computer" in the sense defined by Turing; that is, it can perform any computation which could be performed by any computer whatsoever.

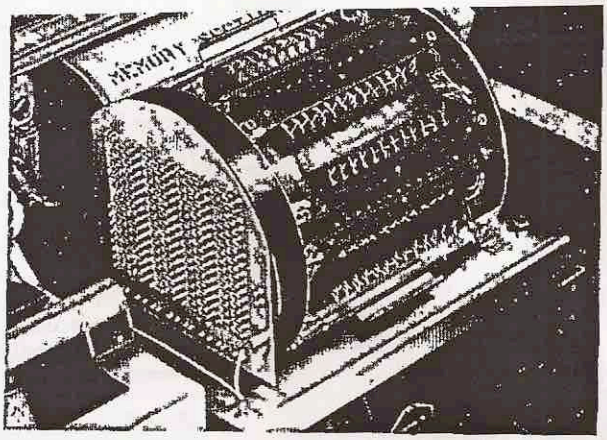

Fig. 2. The magnetic-drum memory has 64 heads, each of which can record 64 words of information on its track, giving a bulk memory of 4096 words. As drum rotates at 3600 rpm, any word is accessible in 17 ms—but special interlace feature for minimum-access-time coding reduces usual access time to 2 ms.

Practical Criteria for General-Purpose Computers

This universal quality of general-purpose computers might at first sight make it appear that all general-purpose computers are equivalent and interchangeable. Various practical considerations modify this apparent equivalence. Perhaps the most important of these is speed. If you have a definite problem, it may be important to you whether a computer can solve it in an hour or a week. For example, we have substantially all of the theoretical background necessary to enable us to predict tomorrow's weather on the basis of today's observations. The prediction would be of little use if it came a week late.

A second practical distinction to be drawn between various computers is the ease with which problems can be prepared for solution. Preparing a problem for a general-purpose computer ("programming") is a process requiring skill and patience. The labor of programming often represents a major part of the cost of problem solution. To minimize this cost, it is desirable that the code of a computer be simple and conform to accustomed ways of thinking.

A third practical consideration bearing on the merit of a computer is that of cost. If two alternative computers were equally able to do a job, the less expensive of the two would be preferred. One important aspect of the cost of a computer is that arising from maintenance requirements.

Aside from breakdowns, another aspect of reliability is the frequency of occurrence of isolated or sporadic errors, worst of all being undetected errors. On this score an extremely high level of reliability is required to make a computer satisfactory. For example, we might imagine a computer which "drops a stitch" about once in a million times. Although this computer might be called 99.9999% reliable, it would in fact be worse than useless.

One of the ways of increasing computer reliability is by cutting down the number of components subject to error or breakdown. Of course, the reliability of a computer is not to be measured only in terms of the number of components subject to breakdown, but also by component reliability. On this score great progress has been made in the past few years. The development of long-life computer tubes, of glass sealed diodes, of printed circuits, and so forth, has permitted us to raise our sights spectacularly regarding standards of reliability. Whereas a few years ago we thought of an error-free running period of a day as encouraging, we may now hope for months or years of trouble-free operation.

Nevertheless, if two computers are designed with equal skill in the choice of components and methods of using them, it is to be expected that the machine with the smaller number of components will have the greater reliability. It is probable that this consideration, as much as consideration of price, has motivated the trend toward simplication of design of digital computers in the past few years. When we describe the LGP-30 in particular, we will not boast about how many tubes it has, but rather of how few.

How is it that the current crop of general-purpose computers require far fewer components than the earlier machines? One reason is simply the general improvement of technology and the greater fund of experience available to the computer designers. For example, ingenious methods have been developed which give increased speed of operation at little or no cost in increased complexity. Another important advance is the development of sophisticated methods of mathematical representation of the internal activities of a computer, which permit a designer to treat the entire computer as an integrated unit rather than having to break it up into compartmentalized sections, each of which is designed separately. In this way much duplication of equipment has been avoided. Where the earlier computers used a large number of components, each called into action only occasionally, present computers use a few components, each doing useful work most of the time.

Another way in which the prevailing mode of computer design has made progress is that communication between the designers and the users of digital computing equipment has improved. Many types of computers have been built, and experience in their use has accumulated. We are able to benefit from this experience by making sounder estimates as to which of the possible features of a computer are really essential to convenient and efficient operation, and which are inessential embellishments which can be eliminated profitably.

Functional Design of LGP-30

We will now turn our attention to a particular general-purpose computer, the LGP-30 (Fig. 1).

Perhaps the most fundamental choice in the design of a digital computer is that of its memory unit, which usually involves the bulk of the hardware of the computer. The LGP-30 computer makes use of a magnetic drum (Fig. 2)—that is, a rotating cylinder coated with a material on which information can be recorded magnetically. At the present time this is, by a wide margin, the simplest method by which a considerable bulk of information can be stored so as to be available for use in times of the order of a hundredth of a second. We use a number of "heads" which record and recover information. Most of these heads serve both purposes, recording and reading. Each head makes use of its own "track" on the drum, in which 64 words of information are recorded. Sixty-four heads are used, thus providing a bulk memory of (64 x 64), or 4096 words. The comparative adequacy of this size for the internal memory is shown by the fact that many high-powered (and high-priced) computers operate with considerably less. A memory capacity of less than, say, one thousand words would severely limit the utility of a computer, a capacity of four thousand odd words provides a comfortable margin for most types of

problems. Of course, a still more capacious memory would be even more comfortable; however, 4096 words puts us well into the region of diminishing returns. Fortunately, we have been able to find ways of providing this relatively large memory without uncomfortably increasing the overall complexity of the computer.

The drum is turned at a rate of about 3600 rpm; thus a word stored anywhere in the memory is accessible in no more than 17 milliseconds. As our later discussion will show, a considerably shorter time, about 2 ms, is in practice a more typical "access time."

Words

A word consists of 32 bits (binary digits) placed in succession in one track of the drum. This way of recording information is called "serial," and its choice is dictated by our requirement of simplicity.

A word of the memory may represent either a number or an instruction. When an instruction, there are 4 bits devoted to *order,* and 12 to *address* (one address per instruction).

The numbers are represented in binary, actually fractional binary. That is, each bit represents a number 2^{-n} (rather than 2^{n} as in straight binary). Thus 30 of the 32 available bits represent all numbers down to 2^{-30}, or approximately 1 billionth. Of the two remaining bits, one is used to specify whether the number is positive or negative; the other serves merely as a rest or "meditation period" at the end of each word. (It is interesting to note that this fractional binary system is really a familiar one—more familiar than the word itself. For example, the ordinary ruler has subdivisions of an inch—a half, a quarter, a sixteenth, etc.).

There are two reasons for the choice of the binary system. One is that it is the most efficient way of using the memory capacity. If, for example, the "binary coded decimal" representation were used, the 30 bits available for indicating the magnitude of the number would be two more than necessary for seven decimal digit accuracy (and two short of enough for eight decimal digits), whereas in the binary system they provide slightly better than nine decimal digit accuracy. The other, and more important, reason for using the binary rather than decimal system is that it considerably reduces the complexity of the number-handling mechanisms. Many digital computers have been designed to operate in an internally decimal (or, more strictly, quasi-decimal) fashion because it was felt that this provided some added convenience for the operator, who presumably feels more at home in the decimal than in the binary system. We felt in designing this computer that that convenience would be outweighed by the disadvantages of the necessary increase in complexity. In this connection it should be noted that in normal use the operator seldom needs to be aware of the binary internal character of the computer; he delivers numbers and instructions in the decimal system and reads decimal answers. The translation into and out of the binary system is performed by a permanent input-output routine which has been prepared in advance.

Input-Output

The input-output device which we have chosen is the Flexowriter. This is an electrically operated typewriter which is equipped to produce, and to be operated by, punched paper tape. This punched tape serves as an external medium for the storage of information. In preparing a problem for the computer, the Flexowriter is used to make an input tape. This may be done in part by directly typing in data and orders, in part by copying and editing such previously prepared tapes as may be useful in the problem at hand. When the problem is to be run, the input tape is put into the Flexowriter, which feeds its information into the computer and simultaneously produces a typed copy for verification. After the input tape has been "read," the computer normally takes over control of the Flexowriter to type out answers or to type and punch on tape data to be reintroduced at a later time. The computer can exercise full control over the Flexowriter so that it can be used to write its results in any desired format, or to type any headings or explanatory material which the Flexowriter is equipped to type.

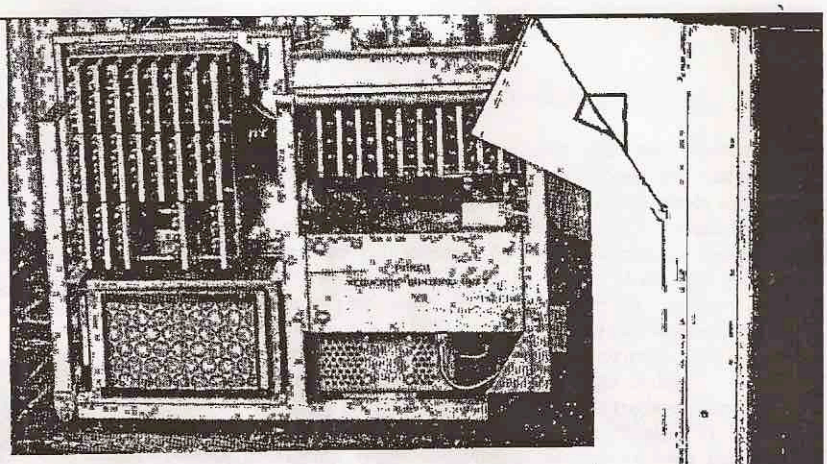

Fig. 3. Rear view of LGP-30, with cover off. Unitized circuits are on etched circuit cards.

Order Structure and Code

Each word placed in succession on each track of the drum contains 32 digits if a number, or 16 digits if an instruction (4 devoted to the specific command order, and 12 devoted to address). The instructions command the computer to perform its built-in operations; the addresses refer to the locations where the number is to be found. The result stays in the accumulator.

We have discussed the various elementary operations performed by all general-purpose computers. Each operation is represented by some symbol or code which may be held in the memory of the computer and causes it to perform that operation. The set of such symbols for a particular computer is its *order code.* As we have seen, a small list is (in principle) sufficient to permit a computer to call itself "general purpose." Usually many more than the necessary minimum number of orders are included for the sake of speed and convenience of operation. The choice of this list results from a compromise between the conflicting aims of simplicity and convenience.

The LGP-30 has a simplified command structure of 16 orders, which are designated on the electric typewriter by a letter and in the machine by a corresponding code number. Table 1 shows the 16 instructions, including code, typewriter letter, address, and explanation. The address *m* refers the machine to the memory location at which the operand may be found.

The LGP-30 has three registers: a *counter register* (control counter) to designate the next instruction to be obeyed, an *instruction register* to hold the instruction being executed while the address is being searched, and an *accumulator* in which the arithmetical operation takes place. One of the operands is always the previous occupant of the accumulator; the result always appears in the accumulator. The second operand is identified by the address which accompanies the order in an instruction.

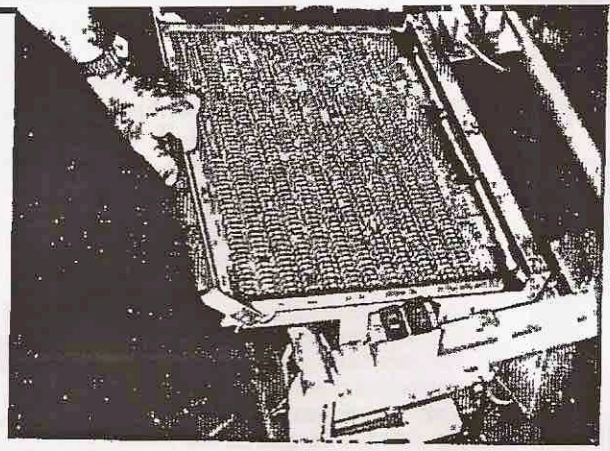

Fig. 4. Logic board contains necessary gating circuits.

Table 1.—Instruction Order List Showing Code for Each Instruction

	Code	Instruction	Effect
ARITHMETIC	0001	B m*	Bring. Clear the accumulator, and add the contents of location m to it.
	1110	A m	Add contents of m to the contents of the accumulator, and retain the result in the accumulator.
	1111	S m	Subtract the contents of m from the contents of the accumulator, and retain the result in the accumulator.
	0111	M m	Multiply the number in the accumulator by the number in memory location m, terminating the result at 30 binary places.
	0110	N m	Multiply the number in the accumulator by the number in m, retaining the least significant half of the product.
	0101	D m	Divide the number in the accumulator by the number in memory location m, retaining the rounded quotient in the accumulator.
	1001	E m	Extract, or logical product order—i.e., clear the contents of the accumulator to zero in those bit positions occupied by zeros in m.
TRANSFER CONTROL	1010	U m	Transfer control to m unconditionally—i.e. get the next instruction from m.
	1011	T m	Test, or conditional transfer. Transfer control to m only if the number in the accumulator is negative.
RECORD	1100	H m	Hold. Store contents of the accumulator in m, retaining the number in the accumulator.
	1101	C m	Clear. Store contents of the accumulator in m and clear the accumulator.
	0010	Y m	Store only the address part of the word in the accumulator in memory location m, leaving the rest of the word undisturbed in memory.
	0011	R m	Return address. Add "one" to the address held in the counter register (C) and record in the address portion of the instruction in memory location m. The counter register normally holds the address of the next instruction to be executed.
MISC.	0100	I	Input. Fill the accumulator from the Flexowriter.
	1000	P t	Print a Flexowriter symbol. The symbol is denoted by the track number part of the address (x).
	0000	Z t	Stop. Contingent on five switch ($T_1 \ldots T_5$) settings on the control panel.

*The address part of the instruction is denoted by m when it refers to a memory location, by t when only the track number is significant. For example, m might be 4732, meaning Sector 32 of track 47.

Arithmetic Orders

The four fundamental arithmetic operations—*add, subtract, multiply* and *divide*—are produced by orders represented by the symbols *A, S, M, D* (the symbols used by the operator and typed on the Flexowriter). Inside the computer they are represented by binary codes, as shown in the order code table (Table 1). Note that *Add* and *Subtract* are represented by fairly similar codes: *A* by 1110 and *S* by 1111. As the internal operations evoked by the two orders are similar, it proved possible to save in design by assigning them similar codes.

In some computers a separate special register, aside from the accumulator, is used to hold a multiplier or the quotient resulting from a division, or for various other special uses. This requires the introduction of special orders to fill that register or to recover numbers from it. Here we have chosen to adhere consistently to the more straightforward form used in the simpler orders: *one of the operands is always the previous occupant of the accumulator, the result always appears in the accumulator.* The second operand is identified by the address which accompanies the order code in an instruction. Thus, for example, the instruction *S* 4732 (as typed on the Flexowriter) means: "subtract the number found in sector 32 of track 47 from the number in the accumulator: keep the difference in the accumulator." (It is more convenient for the operator to identify words of the memory by track number and sector number—00 to 63—rather than by the word number—0000 to 4095.)

On the order *Multiply* the accumulator retains the sign and most significant 30 bits of the product; the remaining 30 bits of the product are discarded. As it is sometimes desired to make use of the less significant half of the product, an alternative multiply order is provided, denoted by *N*, in which it is the least significant 30 bits of the product which are retained. It is sometimes convenient to think of *M* as multiplication applied to fractions, *N* as the multiplication of integers.

One more operation is of partially arithmetic character. It is the order *Extract*, with the symbol *E*. It produces a (binary) digit-by-digit product of the two operand numbers. That is, wherever one number has the digit 1, the corresponding digit of the other number is kept; where it has an 0, the corresponding digit is replaced by zero. Thus a 1 occurs in the result only where both operands held 1's. Though not essential to the operation of a computer, it is useful in various ways. If, for example, several pieces of information of different meaning are stored in one word, the *Extract* order facilitates our later slicing them apart for separate use. An example arises in the use of stored function tables in which interpolation is to be performed. After an argument has been computed, it must be separated into an "integral" and a "fractional" part which are used in different ways. The two parts are easily separated by use of the *Extract* order.

Sometimes it is desired to bring a word from the memory into the accumulator without regard for the present accumulator content; that is, to clear the accumulator to zero and then add the indicated word. This is accomplished by the order *Bring*, denoted by *B*, as shown in Table 1.

Transfer Control Orders

These orders are useful when it is desired to interrupt the normal sequence of operations in a program.

Ordinarily the machine looks for its next instruction in the counter register, which counts forward by one as each instruction has been read. A *U* (unconditional con-

trol transfer order) replaces the address held in the counter register by the address accompanying such an order. Thus the instruction *U* 1234 does nothing to the memory location 1234, but sets the number 1234 into the control counter. Then the computer looks for its next instruction in memory location 1234 rather than in the place indicated by the normal sequence. This is one method by which the computer may be guided into subroutines.

Another order, *Test*, may replace the counter register content by its accompanying address—but only if the number in the accumulator is negative; otherwise the *Test* order is ignored. This order permits the introduction of branching of a sequence of operations, contingent upon results of computation (or on the setting of a switch on the control panel).

Record Orders

After a number has been formed in the accumulator as the result of one or more arithmetic operations, it may be necessary to store it in the memory for use at a later time. This is accomplished by either of two orders denoted *H* and *C* (Table 1). In some cases one wishes to make further immediate use of that number, in other cases one wishes to have the accumulator clear for other use. For this reason the two forms of the record order are provided: *H* is "store the accumulator content but *Hold* it in the accumulator," *C* is "store the accumulator content somewhere and *Clear* the accumulator." For short, these are called *Hold* and *Clear*.

Although we have spoken separately of numbers and instructions, the separation is not basic. It often occurs that one wishes to operate arithmetically on the address part of an instruction. A convenient way of doing this is provided by the order *Record Address*, denoted by *Y*. This causes the recording of just the address part of the accumulator content in the indicated memory location, leaving undisturbed the remainder of the word previously held in that memory location—presumably an order code symbol.

There is one more record order, *Record Return Address*, denoted *R*. Like *Y* it records only in the address part of the indicated word of the memory. The order part of that word is presumably a *U* order, and remains undisturbed. However, unlike all of the other record orders, the information recorded is not drawn from the accumulator but from the control counter. Specifically, the control address (which has already been increased by one since reading this instruction) is added to unity and recorded. The effect of this is to create in the operand word an instruction which refers control back to the second word following that in which the *R* order occurs. The *R* order simplifies the use of "subroutines" (sequences of instructions which do not form an integral part of the main sequence of operations but which stand off by themselves and may be used repeatedly at stages of computation). In using a subroutine, before taking control to its beginning, arrangements must be made to ensure that control is returned to the appropriate place at its end. This is handily done by the *R* order. To call a subroutine into action requires only two words of instruction—an *Rm*, and a *Un*, where *m* is the address of the end of the subroutine, *n* of its beginning.

Break Point and One-Operation Mode

Most discussions of electronic computers emphasize the extremely high speed with which the computer operates. Usually some such statement as this is made: "This machine can perform in two hours a computation which would take a mathematician x years." Such statements are misleading, since mathematicians are notoriously poor computers. The intent of the statement is, however, correct. Any computation the machine can perform the human computer also can, but he might take an unreasonably long time. The only essential advantage of the machine lies in its speed. The human computer is usually less expensive, uses less power, and may even be more decorative (when feminine).

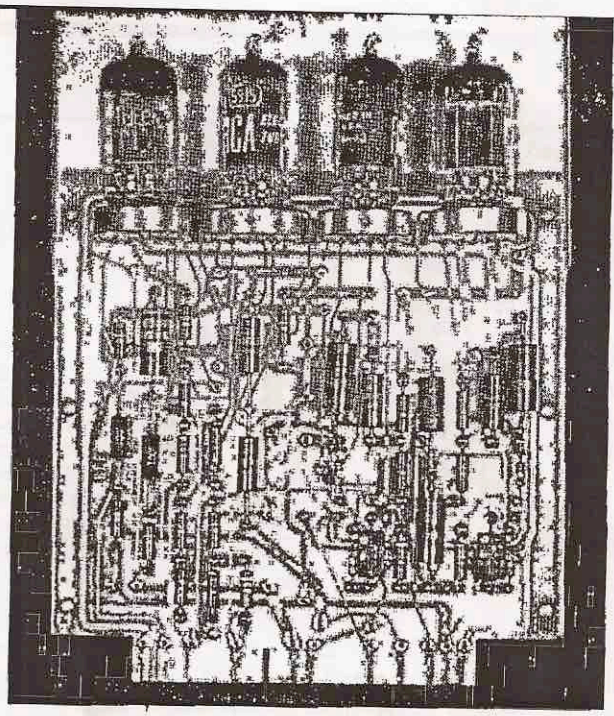

Fig. 5. Etched circuit card, showing one of the unitized circuits.

There are, however, some situations in which the high speed of the computer is not an advantage but a drawback. After a newly programmed problem of considerable complexity has been prepared and put on tape, it is presented to a computer which zips through the calculation at electronic speed and presents an answer; usually wrong. In that situation, it is possible that the computer made an error, but it is much more likely that the fault lies with the mathematician who prepared the program. In trying to track down the source of the error, he finds the high speed of the machine an embarrassment. It runs through the program too quickly for him to be able to see where it deviates from his intended path. Thus, he needs ways of slowing down the action of the computer so that he can make a step-by-step check on its behavior.

Two devices are provided in this computer to help the mathematician out of this difficulty. One is the provision of a *one-operation mode*. With the mode switch set to one-operation the computer executes only one step and then stops. It is restarted, to execute another step, by the start button. By repeatedly pressing the start button, the operator may "pedal through" the calculation at a slow rate. After each step the number held in the accumulator, the instruction just executed, and the address of the next instruction are visible on the display oscilloscope.

Provision of the one-operation mode does not completely solve the mathematician's problem. If the problem is one which will require many minutes for computation with the machine running continuously, it would take days to pedal through the entire program. This is not

LGP-30 Interlace Pattern

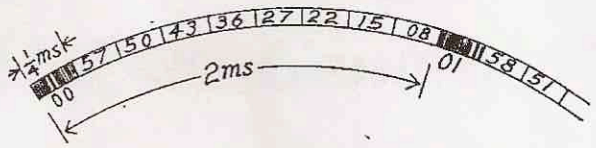

Fig. 6. Word 00 contains instruction 1. Word 01 contains instruction 2. However, 01 is placed 1/7th drum circumference away from 00, and 8 words are placed in the space between. This permits instruction 1 to apply to 6 of the 8 words between 00 and 01 before instruction 2 is reached—and permits a complete computer operation to occur in only 2.3 ms. Average access time can thus be reduced from 17 ms (one complete revolution of drum) to 2 ms.

because the program has a very tremendous number of instructions but because it has loops of instructions which are executed many times in the course of the problem. After the operator has pedaled through a loop once and verified that it performs as required, he needs some method of by-passing the later repetitions of that loop and examining in detail the performance of the next part of the problem. What is needed is a method of stopping the computer at selected points in the program to permit a local step-by-step examination. To satisfy this need, the Break-point order is provided. It is denoted by *Z*. The effect of a break-point instruction is to stop the computation, provided a *break-point switch* is open. If the switch is closed the instruction has no effect. By opening the switch and running in the continuous mode the problem may be brought up to the point where the first break-point order occurs; then by turning to the one-operation mode the part of the program immediately following that break-point can be examined.

The order *Print*, denoted *P*, provides the means by which answers are reported out on the Flexowriter. In some computers an instruction of this kind causes the typing out of an entire word; usually the word designated by the accompanying address. That is an excellent system, particularly if the computer is able to go on about its business while the typing is going on—but it has two drawbacks: One is that it gives little control over the format of the typed-out results. The other drawback is that it requires the use of a considerable amount of auxiliary equipment. We have chosen another system; namely, to control the Flexowriter directly by the computer without the use of intermediate machinery. Our Print order types just one symbol. The Print instruction is modified in its effect by the six digits of the track number of its address (as in the break-point order). Thus, there are, in effect, 64 different print instructions (although not all of these would be meaningful to the Flexowriter). These cause the typing of the decimal digits, letters, punctuation marks, shifts to upper case or lower case, tab release, carriage return, etc.

There is one more order code, called *Write-in* and denoted *I*. Its use is specialized to the input process; that is, the filling of the computer memory by the Flexowriter. Since the input "order" almost never occurs in ordinary programming, it should perhaps not be regarded as an order but rather as a specialized state of the computer used during the input process.

Table 2.—Efficient Program For Formation of Sine of Angle x.

Memory Location	Content	Effect	Memory Location	Content	Notes
0000	H 0043	x to 0043	0035	.00015148	C_9
0001	H 0044	x to 0044	0036	−.00467377	C_7
0002	U 0007	delay	0037		x^2
0007	M 0043	form x^2	0038	.07968968	C_5
0008	H 0037	x^2 to 0037	0039		x^2
0009	U 0010	delay	0040	−.64596371	C_3
0010	H 0039	x^2 to 0039	0041		x^2
0011	U 0012	delay	0042	.57079632	C_1
0012	H 0041	x^2 to 0041	0043		x
0013	M 0035	mult. by C_9	0044		x
0014	A 0036	add c_7			
0015	M 0037	mult. by x^2			
0016	A 0038	add C_5			
0017	M 0039	mult. by x^2			
0018	A 0040	add C_3			
0019	M 0041	mult. by x^2			
0020	A 0042	add C_1			
0021	M 0043	mult. by x			
0022	A 0044	add x			
0023	U	exit.			

Interlace Pattern for Optimum Address

One special aspect of the construction of the LGP-30 is important to the way programming is done for problems in which speed of calculation is important. As previously described, each track contains 64 words recorded around the circumference of the drum. These words are numbered 00, 01, etc., to 63. The normal sequence of orders follows this sequence of numbers; that is, after an instruction in word 00 is obeyed (if it is not a *U* or *T* instruction) the next instruction is found in word 01 of the same track, then 02, etc. After word 63 comes word 00 of the next-numbered track. The numbering of words thus indicates the order in which they would be obeyed as instructions in normal sequence.

However, the words are not placed in succession on the drum—that is, word 01 and word 00 are placed one-seventh of a drum circumference apart to allow a 2-millisecond interval to elapse between successive orders. The space between the words 00 and 01 is devoted to 8 other words, as shown in Fig 6.

The purpose of this *interlace pattern* is to permit programming in a way to speed operations. For example, instruction 1 (in word 00) can instruct the computer to operate on any of the words between 00 and 01, and the operation can be performed before instruction 2 is reached in word 01. (Actually, words 57 and 08 should not be used—but any of the other 6 words can be used.) The problem may thus be planned and programmed so that the average access time is 2 ms rather than the 17 ms required for a complete revolution of the drum. Such planning is called *minimum latency coding*, or *minimum access time coding*. A circular slide rule has been prepared for optimum coding.

A Typical Program

The program shown in Table 2 is an efficient program, prepared by C. F. Flannell, for the formation of the sine of an angle, x. The angle (expressed in quadrants) is in the accumulator on entering the routine, thus the first instruction (in word 00) is to bring the angle x to location 43 on track 00, or to location 0043. The second instruction is to bring the same angle to location 0044 (so that the contents of 0043 and 0044 can be multiplied to give x^2 in the fourth instruction). The delay orders are merely "skips" to get to later optimum-coded instructions. The entire routine, shown in Table 2, gives the sine with about eight-decimal-place accuracy—in about 150 milliseconds.