Technical Newsletter No. 11

APPLIED SCIENCE DIVISION

A COMPLETE FLOATING-DECIMAL INTERPRETIVE SYSTEM FOR THE IBM 650 MAGNETIC DRUM CALCULATOR

V. M. Wolontis
Bell Telephone Laboratories, Inc.
Murray Hill, New Jersey

ABSTRACT

This report describes an interpretive system which transforms the 650 into a three-address, floating-decimal, general-purpose computer, primarily suited for scientific and engineering calculations. The system is complete in the sense that all mathematical, logical and input-output operations normally called for in such calculations can be performed within the system, i. e., without reference to the basic operation codes of the 650. The guiding principles in designing the system have been ease of use, as defined in the introduction, high speed of arithmetic and frequently used logical operations and full accuracy and range for the elementary transcendental functions.

The report serves a dual purpose. It presents the external characteristics of the interpretive system to the potential user by means of detailed explanations accompanied by illustrative examples, assuming no previous familiarity with internally programmed machines. It also describes the internal structure of the system to the professional designer of such systems, enabling him to modify it to suit his particular needs or to borrow ideas or building blocks he may find useful.

The system is available in punched card form to anyone who requests it.

CONTENTS

Note: The material of immediate concern to those who wish to learn how to program problems in the interpretive system is contained in sections II-X. Section I is devoted to general considerations and may be bypassed. Section XI deals with the internal structure of the system, primarily for the benefit of those interested in the design of interpretive systems, but the discussion of possible modifications in Sec. XI. 1. and the contents of Sec. XI. 2. and XI. 3. should be of wider interest and do not require familiarity with the basic language of the 650.

The experienced programmer may absorb the essentials of the system by reading the definitions of the operations. Page references to them are given in the summary of operation codes.

CONTENTS (continued)

I INTRODUCTION

I. 1. GENERAL DESIGN CONSIDERATIONS

The use of most existing computing devices whose degree of automatic performance substantially exceeds that of a desk calculator entails certain problems not encountered in desk computing. To cope with these problems, one may incorporate additional circuitry into the machine--this, indeed, appears to be the trend in recently announced commercially available machines--or, alternatively, one may program, in terms of the basic language of the machine, a system or super-language in terms of which the general user will program his problems. The user may consider the machine and the super-language as one entity, and no knowledge of the basic machine language is required of him. Before actual calculation, the programmer's instructions are translated by the machine into the basic language. If this translation or interpretation takes place each time an instruction is to be executed, rather than once for all at the beginning of a problem, the super-language is referred to as an interpretive language or system. Limitations in storage capacity may necessitate the choice of an interpretive system rather than a system of the once-for-all type in the case of most small or medium-sized computers.

The designers of an interpretive system are faced with a very large number of decisions. To provide a basis of motivation for these decisions, it is convenient to list here, in somewhat arbitrary order, some of the above-mentioned problems which the present interpretive system proposes to solve. All of them may fundamentally be measured in terms of total time spent by a programmer in learning to use the machine and in using it on a specific problem. In this sense, the "ease of use" referred to in the abstract above is implicitly defined by the list that follows. The price paid for the saving of programmer time is, of course, to be found in substantially reduced speed of operation.

A. Scaling

The storage medium--paper--normally used in desk computing places no practical restriction on the size of numbers or on the location of decimal points. In using a computer that automatically stores intermediate results in registers of fixed length and with the position of the decimal point fixed in advance, a great deal of time must in most cases be spent on estimating the range of all intermediate results to prevent errors due to overflow. The well-known way of avoiding this at the expense of a very substantial increase in the internal complexity of the arithmetic operations is to represent each non-zero number in floating-decimal form, i. e., as a signed quantity whose absolute value lies in a fixed range, accompanied by an exponent of 10 or decimal point indicator.

B. Length and Complexity of the Program

Floating decimal arithmetic and frequently needed special functions could be incorporated into a program written in the basic machine language in the form of a set of subroutines reached by a two-way transfer of control, ("calling sequence" or "basic linkage") and there are indeed problems for which this is the best choice. In many cases--particularly in the case of relatively short problems where the results are needed quickly--a further reduction of the programming effort is desirable. This may be achieved by combining under single operation codes, in an interpretive language, groups of steps in the basic language needed for performing frequently occurring tasks. For example, a single instruction in which three locations are specified may be used for adding two arbitrarily located numbers and storing the result or a block of information of any length may be punched on cards as a result of a single instruction. In particular, the task of repeating a calculation a specified number of times, each time with appropriate modifications, must be made easy, and the interpretive system described in this report goes as far as is believed possible in this direction by providing an order ("LOOP") with which simple cases of this task can be handled by a single instruction.

To preserve the simplicity gained by introducing an interpretive system, the system must be made complete or self contained so that most problems can be conveniently programmed without reference to two different systems of operation codes, although, of course, leaving and re-entering the interpretive system should be made possible in order to provide the experienced programmer with complete flexibility.

C. Restrictions

In desk computing, one cannot fail to notice if the argument for which a function value is to be found in a table falls outside the range covered by the table or if in transcribing a set of numbers from one area on a piece of paper to another, overlapping area, some of the numbers to be transcribed are erased before transcription. A machine will avoid or detect such blunders only if programmed to do so, and as much as possible of this programming should be included in the interpretive system. For example, it is desirable that all mathematical functions included in the system be available for the full range of argument consistent with their definition and with the limitations imposed by the machine itself and that they are computed to the full accuracy of the number system used. Error stops indicating violations of unavoidable restrictions should be included to the fullest extent that space limitations permit.

D. Program Testing

The usefulness of a general-purpose computer or interpretive system depends decisively on the methods provided for testing ("debugging") new programs--for definitions and details, see Sec. VI. In the case of an interpretive system whose operating speed* is only one order of magnitude above the speed of card punching, testing by means of a tracing routine included in the system compares favorably to console testing, at least in the case of programmers whose familiarity with the machine is limited. Either of these methods is thought of as a tool normally used only when memory print-outs have been found insufficient. To facilitate testing by any of the methods mentioned and to keep programming as concrete as possible, the system described in this report assumes that the actual machine location of each instruction and stored number is assigned by the programmer. The system may, of course, be used in conjunction with regional or symbolic assembly programs.

I. 2. CHANGES AND ADDITIONS

Numerous minor changes suggest themselves when the system is viewed in the light of the experience gained in designing it; some of them are discussed in detail at the end of the report. Corrections of errors not yet revealed must be expected. External changes and additions will undoubtedly be proposed after a period of use. The present system should, therefore, be considered primarily as a first version which each user may consider changing to better suit his needs. Comments and suggestions on both internal and external aspects of the system will be greatly appreciated.

II GENERAL INFORMATION

II. 1. THE 650

The IBM 650 is an electronic computer whose basic storage consists of a magnetic drum capable of holding 2000 words (numbers) of ten decimal digits and sign. The machine is internally programmed, i. e., the program of instructions which the machine is expected to follow is kept on the drum, and the machine automatically reads one instruction at a time from the drum, executes it, reads another instruction, and so on. Initially, the program is loaded onto the drum from punched cards but each instruction is loaded only once, although the machine may be expected to execute it many times in the course of a problem. Special orders are inserted into the program to cause repetition of prescribed sections the desired number of times. In many cases, the instructions executed by the machine are changed or modified under control of the program between successive executions. This ability to modify its own instructions is one of the distinguishing characteristics of an internally programmed machine.

In the basic model of the 650, all answers are punched by the machine into cards, which may be printed on a separate tabulator.

The 650 is a general-purpose, fixed-decimal machine and any programmer may, with the aid of a detailed manual published by IBM, learn to use it as such. There are many large problems and many problems of a data processing nature for which fixed-decimal operation is definitely indicated, and the programmer is asked to give serious consideration to this alternative for all but the very smallest problems, since the gain in machine time over floating-decimal operation (explained below) may be as high as 10:1. The machine-language programmer may relieve himself of many tasks by using the interpretive system for loading, punching, calculation of special functions, etc., provided 1000 storage locations suffice for his problem. (See TR OUT, Sec. IV. 1)

This report describes a system which enables the programmer to use the 650 as a floating-decimal machine, without being familiar with the fixed-decimal mode of operation. Beginning with the next section, all statements will concern the system rather than the 650 itself, but it should be borne in mind that if anything in the system appears restrictive from the viewpoint of a particular application, --storage capacity, speed, card form, word length, etc. --total or partial use of basic 650 coding may be the answer.

II. 2. THE INTERPRETIVE SYSTEM: STORAGE; DATA AND INSTRUCTION FORM

When the interpretive system is in use 999 ten-digit storage locations, numbered 001-999, are unrestrictedly available to the programmer for storing instructions and data. The location 000 has a special use ("previous result") which will be discussed below.

Throughout the system, numbers upon which mathematical operations are performed are stored and used in so-called (normalized) floating-decimal form, which will be defined as follows: The number zero is written as ten zeros with a plus sign ("machine zero"). Any number A other than zero is expressed as

$$\bar{A} = \pm A_1 \cdot 10^{a_1}$$

where $1 \leq A_1 < 10$ and $-50 \leq a_1 \leq 49$. In the machine, $\bar{A}$ is written as the pair $\pm(A_1, a)$, where $a = a_1 + 50$ and A_1 is an eight-digit number with seven decimal places. The "machine exponent" a is a two-digit (positive) integer located at the right end of the number. Non-zero numbers A not in the range $10^{-50} \leq |\bar{A}| < 10^{50}$ cannot be correctly used in the machine, and some of the mathematical operations will give an error stop if the result would fall outside this range (see STOPS). Numbers loaded into the machine must also be in the form prescribed above, unless special precautions are taken (see Sec. X. 2).

The system instructions are signed ten-digit numbers of the following form:

±	0_1	A or 0_2	B	C

Here, 0_1 is a one-digit operation code and B and C are three-digit addresses. The three-digit quantity "A or 0_2" is interpreted as an address A if $0_1 \neq 0$ and as an operation code 0_2 if $0_1 = 0$. The sign of the instruction is used in connection with the LOOP order (see LOOP OPERATIONS). The only difference between the mutually exclusive 0_1 and 0_2 operations is that all operations which require three addresses have been designated 0_1, all others, 0_2.

An example will illustrate how the addresses are used in a program: To add the number stored in register 200 to the number stored in register 201 and store the result in 500, the operation code $0_1 = 1$ (ADD) is used and the instruction reads: 1 200 201 500. To take the square root of the number in 200 and store the result in 500, the 0_2-operation $0_2 = 300$ (SQRT) is used: 0 300 200 500. As will be shown in later sections, it is also possible to call out an instruction stored in memory and operate upon it, e.g., increase one of the addresses in it. In storage, no distinction is made between instructions and data so that the programmer is free to use any memory location for storing an instruction or a number as he sees fit.

To facilitate explanations, the following notation will be used: The ten-digit quantity whose storage location has the address A will be denoted by $\bar{A}$; analogously, $\bar{B}$ will denote the contents of location B. $\bar{C}$ denotes the result of a calculation about to be stored in location C.

In addition to being stored at C, the result $\bar{C}$ of any mathematical operation (i.e., arithmetic operations or special functions) and of MOVE 000 and CONS (see READING) is automatically stored in the special location 000. If this result is needed on the next step, (or, more generally, before it has been replaced by the result of a subsequent mathematical operation or MOVE 000 or CONS) calling it out by using 000 as an A-address will reduce the execution time in the case of the arithmetic operations. Also, time will be saved in any mathematical operation by using 000 as a C-address when $\bar{C}$ will be needed only on the next step. Execution times are discussed in detail in a later section, but it should be emphasized that timing considerations only affect the running time of a problem, never the correctness of results. All locations are accessible at any time.

Special addresses for obtaining frequently needed numbers, such as, 0 and 1, are not provided by the system. The programmer should load such numbers into locations of his own choosing.

III MATHEMATICAL OPERATIONS

III. 1. ARITHMETIC OPERATIONS

The operations will be introduced in an order chosen to facilitate the learning process. Later, a concise summary of all operation codes will be given. Alphabetic operation codes are listed in addition to the numerical ones merely to facilitate programming; they are not introduced into the 650 and need not be used at all.

The result of each arithmetic operation is rounded. If the result is zero, a machine zero is given, i. e., the machine exponent will be 00. An error stop occurs if the result of a multiplication or division would fall outside the range of the floating-decimal number representation; another error stop detects attempts to divide by zero (see STOPS).

A list of the arithmetic operations follows:

Numer. code	Alpha. code	Function
$0_1 = 1$	ADD	Add (in floating-decimal form) the number $\bar{A}$ stored at A to the number $\bar{B}$ stored at B, store the result $\bar{C}$ at C and 000. Abbreviated: $\bar{A} + \bar{B} = \bar{C}$
$0_1 = 2$	SUB	Subtract: $\bar{A} - \bar{B} = \bar{C}$
$0_1 = 3$	MPY	Multiply: $\bar{A} . \bar{B} = C$
$0_1 = 4$	DIV	Divide: $\bar{A} / \bar{B} = \bar{C}$
$0_1 = 5$	NGMPY	Multiply negatively: $-\bar{A} . \bar{B} = \bar{C}$

III. 2. SPECIAL FUNCTIONS

The system is intended to give eight-digit accuracy (i. e., an error less than 1 in the eighth digit) in computing the special functions included whenever the input makes this accuracy possible. For trigonometric functions of an argument exceeding one revolution and for logarithms of numbers near 1, loss of accuracy follows from the mathematical properties of the respective functions and stops (which may be bypassed by the setting of a console switch) are provided when this loss exceeds two digits. For small values of the argument, where an eight-digit, fixed-decimal representation of the sine or arc tangent would contain leading zeros, the floating-decimal representation would normally introduce meaningless digits at the right end. To reduce this nuisance to a tolerable level and also make possible trigonometric calculations with extremely small arguments, the formulas $\sin x = x$ and $\arctan x = x$ are used for $|x| < .0025$ and $|x| < .001$, respectively. Those interested will find the methods of computing the special functions described in Section XI. 3.

Aside from the limitations imposed by the above mentioned inherent loss of accuracy and by the floating-decimal representation of the result, no restrictions apply to the natural range of the argument for the special functions. Error stops will prevent attempts to take the square root of a negative number or the logarithm of a non-positive number.

The special functions (or, more precisely, elementary transcendental functions) are:

Numer.	Alpha.	Function
$0_2 = 300$	SQRT	$\sqrt{\overline{B}} = \overline{C}$
$0_2 = 301$	EXP E	$e^{\overline{B}} = \overline{C}$
$0_2 = 302$	LOG E	$\log_e \overline{B} = \overline{C}$
$0_2 = 303$	SIN R	$\sin \overline{B} = \overline{C}$, $\overline{B}$ in radians
$0_2 = 304$	COS R	$\cos \overline{B} = \overline{C}$, $\overline{B}$ in radians
$0_2 = 305$	ART R	$\arctan \overline{B} = \overline{C}$, $\overline{C}$ in radians, $\lvert\overline{C}\rvert < \pi/2$
$0_2 = 350$	ABS	$\lvert\overline{B}\rvert = \overline{C}$
$0_2 = 351$	EXP 10	$10^{\overline{B}} = \overline{C}$
$0_2 = 352$	LOG 10	$\log_{10} \overline{B} = \overline{C}$
$0_2 = 353$	SIN D	$\sin \overline{B} = \overline{C}$, $\overline{B}$ in degrees
$0_2 = 354$	COS D	$\cos \overline{B} = \overline{C}$, $\overline{B}$ in degrees
$0_2 = 355$	ART D	$\arctan \overline{B} = \overline{C}$, $\overline{C}$ in degrees, $\lvert\overline{C}\rvert < 90$

Subdivisions of a degree are expressed decimally, not in minutes and seconds.

III. 3. MOVE 000

In many cases (particularly in connection with the use of subroutines) it may be convenient to be able to call out a number $\bar{B}$ from B and deposit it in C, as well as in 000, without the time-consuming use of a floating-decimal arithmetic operation. This is accomplished by the logical operation $0_1 = 9$ ("MOVE") with $A = 000$. The normal use of MOVE with $A \neq 000$ is described in Sec. IV.4.

III. 4. AN EXAMPLE

For the benefit of anyone with no previous computer experience, a simple example illustrating the use of the mathematical operations will be inserted here. Suppose that, as a part of a program which is assumed to be already on the drum, it is desired to evaluate the function

$$f(x) = \frac{\sin x}{\sqrt{1 + e^{-x^3}}}$$

Here, x in radians is assumed to be in storage register 500 and the constant 1 in 600. The quantity e^{-x^3} is to be stored in 501, and f(x) in 502. A program might look as follows:

Alpha.	0_1	A or 0_2	B	C	Comments
MPY	3	500	500	000	x^2
NGMPY	5	000	500	000	$-x^3$
EXP E	0	301	000	501	e^{-x^3}, store in 501
ADD	1	000	600	000	$1 + e^{-x^3}$
SQRT	0	300	000	400	$\sqrt{1 + e^{-x^3}}$, store temporarily
SIN R	0	303	500	000	sin x
DIV	4	000	400	502	f(x), store in 502

The extensive use of the "previous result" address, 000, is worth noting.

IV LOGICAL OPERATIONS

IV. 1. TRANSFER OPERATIONS

Suppose the machine has been instructed (see LOADING) to begin a program by executing the instruction stored in, say, location 101. When this execution is completed, the machine will automatically execute instruction 102, then 103, etc., until told by the program to do otherwise. Operations whose primary function is to influence either the order in which instructions are executed by the machine or the selection of stored data upon which the instructions make the machine operate will be called logical operations. A simple example of such an operation is $0_2 =$ 203, "Transfer Control". If in the sequence 101, 102, 103 above, instruction 103 should read "TR 0 203 000 080", the next instruction executed by the machine would be 080 instead of 104. This may be expressed by saying that "control was transferred to 080". The B-address was ignored in this case. The transfer of control may be made to depend on the result of calculations (mathematical or logical) in which case a "conditional transfer" is said to occur. Logical operations--conditional or unconditional--are needed whenever several blocks of instructions, located on various parts of the drum, are to be tied together to form a program, whenever it is desired to repeat a calculation several times, etc.

For simplicity in grouping, the following list of transfer operations includes two (UNC STOP and NOOP) whose transfer function is of a degenerate nature. In a first reading, it may be advantageous to omit the TR SUBR and TR OUT operations.

Numer.	Alpha.	Function
$0_2 = 000$	UNC STOP	Stop unconditionally. The machine stops regardless of the setting of console switches (see CONSOLE) and displays 9999 on the address lights and $\bar{B}$ on the display lights. This operation should be used only where it is intended to discontinue the execution of the program, since a continuation of the program cannot be effected by a simple depression of the PROGRAM START key (see STOPS). The C-address is ignored but should be filled, e.g., with zeros (see LOADING).
$0_2 = 200$	COND STOP	Stop conditionally and transfer. The machine stops if the PROGRAMMED STOP switch on the console is in the STOP position. The number 1120 is displayed on the address

Numer.	Alpha.	Function
		lights and $\bar{B}$ on the display lights. When the PROGRAM START key is depressed, control is transferred to C. If the PROGRAMMED STOP switch is in the RUN position, control is transferred to C without stopping. Caution: If the PROGRAMMED STOP switch is on RUN, the stops for loss of accuracy in sine, cosine and logarithm and the stop in the CONS operation will not occur.

This operation may be used for stopping at check points in the early running stages of a problem, with the option of avoiding the stops during later runs.

$0_2 = 201$	TR SGN	Transfer on sign. Control is transferred to C if the result of the last mathematical operation or MOVE 000 or CONS is negative, to B if it is non-negative. (i. e., zero is regarded as having a plus sign).
$0_2 = 202$	TR EXP	Transfer on exponent. The exponent, c, of the result of the last mathematical operation or MOVE 000 or CONS is compared to B (the leading digit of B should be 0). Control is transferred to C if $c \geq B$. If $c < B$, control proceeds to the next instruction.

This operation is particularly suited for the summation of series where terms are to be added until they have a prescribed number (50 - B) of leading zeros. For example, in order to return to instruction 080 only as long as the absolute value of the previous result is .0001 or greater, one would write "TR EXP 0 202 046 080". This saves a time-consuming floating subtraction preceding the test. The TR EXP operation is also intended to take the place of the TR ZERO operation found in most systems. Due to the accumulation of small errors during a calculation, it is unwise in most cases to expect a result to be exactly zero to eight figures; here a TR EXP with a suitably chosen B may prevent a never-ending repetition of a part of a program.

$0_2 = 203$	TR	Transfer. Control is transferred to C, i. e., the next instruction exe-

Numer.	Alpha.	Function
		cuted will be the one stored at C. The B address is ignored but should be filled, e.g., with zeros.
$0_2 = 204$	TR SUBR	Transfer to subroutine. The C-address of the instruction located at C is set equal to B, whereupon control is transferred to C. The sign of the instruction at C is made positive. (For an elucidation and applications see SUBROUTINES.)
$0_2 = 205$	TR OUT	Transfer out. Control is transferred to C and the instruction stored there is executed in the basic language of the machine (i.e., outside the interpretive system). When an instruction address 1095 is given in the basic language, control is returned to the interpretive system beginning at the instruction following the TR OUT. The B-address of the TR OUT instruction is ignored but should be filled, e.g., with zeros. The programmer in basic language must be careful not to use locations above 999, which are occupied by the interpretive system.
$0_2 = 454$	NOOP	No operation. Control proceeds to the next instruction. The B- and C-addresses are ignored but should be filled, e.g., with zeros.

This operation is likely to occur in connection with tracing (see TRACING, particularly the ST TR ERAS operation) or when a superfluous instruction has been deleted from a program.

IV. 2. LOOP OPERATIONS

A highly repetitive character is required of any problem to be economically handled on an automatic computer. In certain instances, such as Newton's iteration procedure for the solution of equations, a repetitive process or "loop" is conveniently programmed, merely using conditional transfer operations. In many cases, however, some of the instructions to be repeated must be slightly modified in a systematic way before each new repetition. For example, in the evaluation of a linear expression $\sum_{i=1}^{N} a_i x_i$ with the a_i and the x_i stored in blocks of consecutive locations, the addresses of a_i and x_i must be increased by 1 each time a new term is to be computed. To facilitate programming of this kind, the system provides two methods of so-called address modification. The simpler--but less general--of these methods employs a special counter called the loop box, which is stored in a location normally inaccessible to the programmer. If an instruction carries a minus sign, the current contents of the loop box will be added to the instruction (in fixed-decimal arithmetic and without regard to the sign) before it is executed. If, for example, the instruction - 1 531 600 901 is given and the loop box contains + 0 009 000 009, the instruction actually executed by the machine would read 1 540 600 910. The original instruction remains unchanged in its storage location. At the end of a calculation, an 0_2 instruction called LOOP enables the programmer to increase the contents of the loop box by 1 in one or several address positions and to transfer control back to the beginning of the calculation. Hence, the calculation may be carried out repeatedly, each time with different addresses used in the execution of instructions with minus signs. A test provision included in the LOOP order stops the repetition after a specified number of executions and resets the loop box to zero for future use. An example will be given after the following list of LOOP operations.

Numer.	Alpha.	Function
0_2 = 100	LOOP A	Loop on A. The contents of the loop box are increased by 0 001 000 000. In other words, the A-segment of the loop box is increased by 1. <u>After</u> the increase, the A-segment of the loop box is compared to the B-address of the LOOP instruction. If the A-segment is less than B, control is transferred to C. If the A-segment is equal to B, (or greater, which will never be the case in normal use) the loop box is reset to zero and control proceeds to the next instruction.
0_2 = 010	LOOP B	Loop on B. Analogous to LOOP A, with the B-segment of the loop box now being increased and compared to the B-address of the LOOP instruction.
0_2 = 001	LOOP C	Loop on C. Analogous to LOOP A, with the C-segment of the loop box being increased and compared to B.

Numer.	Alpha.	Function
0_2 = 110	LOOP AB	Loop on A and B. Analogous to LOOP A. The A- and B-segments of the loop box are increased by 1 and the A-segment is compared to B.
0_2 = 101	LOOP AC	Loop on A and C. Analogous.
0_2 = 011	LOOP BC	Loop on B and C. Analogous. The B-segment is used for the comparison.
0_2 = 111	LOOP ABC	Loop on A, B and C. Analogous. The A-segment is used for the comparison.

To illustrate the use of a LOOP order, consider the evaluation of the linear expression $L(x) = \sum_{i=1}^{20} a_i x_i$, where the a_i and the x_i are stored in memory. In choosing storage locations for numbers, it is wise to plan in advance how they are to be used in the program. In this case, since the a_i and the x_i are to be reached using the LOOP operation, it is advantageous to store them in blocks of consecutive locations, say the a_i in 800 + i and the x_i in 900 + i, (i = 1, 2, ..., 20). Suppose L(x) is to be stored in 700. For simplicity, assume that register 700 contains zero at the beginning of the calculation and that the loop box has been reset. The entire program for this calculation might be written as follows:

Instr. No.	Alpha.	Sign.	0_1	A or 0_2	B	C
101	MPY	—	3	801	901	000
102	ADD	+	1	000	700	700
103	LOOP AB	+	0	110	020	101
104	Next instruction in the problem.					

Note that the B-address of the LOOP order simply indicates the number of times the arithmetic calculation is to be performed, including the first time when the addresses are actually unmodified (modified by adding zero). The practice of starting the instruction numbering at, e.g., 101, rather than 001 facilitates later additions to the beginning of a program.

The loop box is automatically reset at the beginning of a new problem (see LOADING), and whenever a transfer out of a loop is effected by a loop order (as stated in the definitions above). Hence, the resetting of the loop box need not concern the programmer under normal conditions. If the need for resetting the loop box should arise, however, this is easily done by giving, e.g., the order LOOP A with the B-address 000. According to the definition of LOOP A, this will cause control to proceed to the next instruction with a resetting of the loop box.

The C-address is irrelevant in this case. This situation would arise if control were transferred out of a loop in the middle of it by one of the conditional transfer operations.

It is worth observing that a LOOP operation may be advantageously used in some cases where address modification is not involved, simply to repeat a sequence of steps a prescribed number of times, e.g., each time adding a fixed increment to a parameter. In such a case, any one of the loop orders could be chosen, (see EXECUTION TIMES, however) and no negative instructions would occur.

The advantages of the loop-box method are its simplicity and high speed and the fact that the original instructions remain unchanged in memory. It is limited by the fact that there is only one loop box and hence, all instructions to be modified are modified in the same way. To handle situations more complicated than this, the system provides a set of operations described in the next section.

IV. 3. ADDRESS CHANGE OPERATIONS

Many problems can be completely programmed without the use of address change operations, and for someone approaching the field of internal programming for the first time, it might be advantageous to ignore this section until the need for more general logical operations arises.

The functions of the address change operations are: (a) To increase or decrease a designated address of an instruction in storage by any given amount; (b) To set such an address to a given value (without reference to its previous value); and (c) To transfer control as a result of comparing such an address to a given number.

There are nine 0_2-operations among the address change operations. In each of these, the B-address gives the location of the instruction ($\bar{B}$) to be changed and the C-address is the amount of change. For example, suppose the instruction 0 600 750 005 (using the operation $0_2 = 600$, ADD A) is given and suppose location 750 contains the instruction I 320 400 000. Then the A-address, 320, of this instruction will be increased by 005 and the resulting instruction 1 325 400 000 stored back in 750. Similarly, if 0 050 750 333 were given, (using $0_2 = 050$, SET B) the instruction in 750 would be changed to read 1 320 333 000. In brief:

Numer.	Alpha.	Function
$0_2 = 500$	SET A	Set the A-address. The A-address of the instruction ($\bar{B}$) specified by B is set equal to C.
$0_2 = 050$	SET B	Set the B-address. The B-address of the instruction ($\bar{B}$) specified by B is set equal to C.
$0_2 = 005$	SET C	Set the C-address. The C-address of the instruction ($\bar{B}$) specified by B is set equal to C.

Numer.	Alpha.	Function
$0_2 = 600$	ADD A	Add to the A-address. The A-address of the instruction ($\bar{B}$) specified by B is increased by C.
$0_2 = 060$	ADD B	Add to the B-address. The B-address of the instruction ($\bar{B}$) specified by B is increased by C.
$0_2 = 006$	ADD C	Add to the C-address. The C-address of the instruction ($\bar{B}$) specified by B is increased by C.
$0_2 = 700$	SUB A	Subtract from the A-address. The A-address of the instruction ($\bar{B}$) specified by B is decreased by C.
$0_2 = 070$	SUB B	Subtract from the B-address. Analogous to SUB A.
$0_2 = 007$	SUB C	Subtract from the C-address. Analogous to SUB A.

The sign of the instruction being modified remains unchanged and does not affect the outcome of the modification. Attempts to increase an address beyond 999 or decrease it below 0 will result in erroneous operation not prevented by error stops.

Three 0_1-operations, TR A, TR B and TR C, complete the set of address change operations. In each of them, the A-address specifies the instruction ($\bar{A}$) to be called out and the B-address is the constant to which a specified address is to be compared. In case of inequality, control is transferred to C. For example, if the instruction 6 750 325 200 (using $0_1 = 6$, TR A) is given, control will be transferred to 200 if the instruction in 750 reads 1 320 400 000 but control will proceed ahead if 750 contains 1 325 400 000. Summarizing:

$0_1 = 6$	TR A	Transfer on the A-address. The A-address of the instruction ($\bar{A}$) specified by A is compared to B. Control is transferred to C if they are unequal but proceeds to the next instruction if they are equal.

Numer.	Alpha.	Function
$0_1 = 7$	TR B	Transfer on the B-address. The B-address of the instruction ($\bar{A}$) specified by A is compared to B. Control is transferred to C if they are unequal but proceeds to the next instruction if they are equal.
$0_1 = 8$	TR C	Transfer on the C-address. The C-address of the instruction ($\bar{A}$) specified by A is compared to B. Control is transferred to C if they are unequal but proceeds to the next instruction if they are equal.

As an introductory example, the summation in the section on LOOP OPERATIONS will be programmed again using address change methods. This would be an inefficient choice in an actual problem, but it will best illustrate the difference, as well as the analogy between the two methods. It is again assumed that register 700 contains zero at the start, but the steps analogous to the resetting of the loop box will be included.

Inst.	Alpha.	Sign	0_1	A or 0_2	B	C
101	SET A	+	0	500	103	801
102	SET B	+	0	050	103	901
103	MPY	+	3	[ ]	[ ]	000
104	ADD	+	1	000	700	700
105	ADD A	+	0	600	103	001
106	ADD B	+	0	060	103	001
107	TR A	+	6	103	821	103
108	Next instruction in the problem.					

The brackets in the A- and B-addresses of instruction 103 are used to indicate that these addresses are variable and will be supplied by the program before the instruction is executed, hence what is written there when the program is loaded into the machine is irrelevant. At the end of the program when instruction 108 is reached, memory location 103 will contain + 3 821 921 000. It is assumed that the summation just programmed is part of a larger problem in which it is used repeatedly. This is the reason for the SET A and SET B instructions. If 801 and 901 were simply loaded into their respective positions in instruction 103 initially, the summation would be performed correctly the first time it is used, but the next time when the summation is called for, instruction 103 would read + 3 821 921 000 and erroneous calculations would result. The SET instructions could, of course, have been inserted after the completion of the summation, restoring instruction 103 to its proper value for the next application, but this procedure is not recommended because it makes it more difficult to restart the problem from the beginning without reloading the program in case of an interruption (e.g., error stop) during the loop.

A more realistic example of the use of address change methods would be a calculation involving more than one summation index or parameter. Then, one of the fast and convenient LOOP orders would normally be used in the "inner loop", i.e., the loop occurring most frequently, with address change operations controlling the "outer loop" or loops. Suppose, for example, that it is desired to calculate $S_j = \sum_{i=1}^{10} a_{ji} x_i$ for j = 1, 2, ... 5, where the a_{ji} are stored in $800 + 10_j + i$ (i.e., a_{11} is in 811, a_{12} in 812, etc.; a_{21} in 821, a_{22} in 822, and so forth), the x_i in 900 + i, and the S_j are to be stored in 700 + j. It will be assumed that register 500 contains zero. For completeness, the setting of all variable addresses to their initial values for repeated use of the summation program will be included.

Instr.	Alpha.	Sign	0_1	A or 0_2	B	C	Comments
101	SET A	+	0	500	104	811	Set variable addresses to their initial values
102	SET C	+	0	005	107	701	
103	MOVE	+	9	000	500	400	Set register 400 to zero
104	MPY	−	3	[ ]	901	000	"Inner loop" i.e., summation on i
105	ADD	+	1	000	400	400	
106	LOOP AB	+	0	110	010	104	
107	MOVE	+	9	000	400	[ ]	Store the result
108	ADD A	+	0	600	104	010	Increase addresses for next repetition in the outer loop (j-loop)
109	ADD C	+	0	006	107	001	
110	TR C	+	8	107	706	103	Test for end of j-loop
111	Next instruction in the problem.						

A superficial examination of this program might suggest that only 1/5 of the program is devoted to actual arithmetic calculation (!), but it should be observed that in terms of the number of instructions executed by the machine when one complete summation is performed, the arithmetic ones are still in the majority, and in terms of execution time they comprise about 3/4 of the program.

In programming problems involving several loops, it may be helpful to consider the structure of a loop in terms of four phases:

1. Initialization. Where addresses in the loop are set to their initial values, registers used for summation are set to zero, etc.. The automatic resetting of the loop box and the fact that addresses remain unchanged in memory tend to reduce the initialization when the loop is controlled by a LOOP operation. In the summation problem above, steps 101 and 102 constitute the initialization for the outer loop, step 103 is the initialization for the inner loop. Notice that step 103 is repeated as a part of the outer loop.

2. Execution. Comprising the mathematical operations of the loop, as well as any logical operations associated with a loop inside the one being executed. Above, the execution of the inner loop consists of steps 104 and 105 and the execution of the outer loop consists of 103-107.

3. Modification. Where addresses, parameter values, etc., are increased or decreased. The modification of the inner loop above is included in the LOOP instruction. The modification of the outer loop consists of steps 108 and 109. The position of the modification in the program in relation to the execution and test is frequently subject to choice.

4. Test. Determining whether the loop is completed or further repetition is required. The LOOP instruction includes the test for the inner loop and step 110 is the test for the outer loop.

   Note: It is important to write loops in such a way that all initializations are performed by the program, not by loading. If this rule is not followed, it will not be feasible to restart the program during testing or after a machine stop without reloading. For example, if a register is used for summation, it should be reset before being used in the summation loop by moving zero into it from another location, not by loading zero into it from a card.

Many programmers find it helpful in programming a large problem to draw a block diagram or flow chart with one box representing each phase of each loop and arrows connecting the boxes showing the flow of control.

The address change operations, particularly the SET operations, are frequently useful in non-repetitive situations as well. An example of this will be found in the section on SUBROUTINES.

If a program appears to require a large amount of address modification and particularly, if this occurs because a quantity whose address is subject to change is needed in many places in the execution of a loop, it may be advantageous to write the execution largely in terms of fixed addresses and perform the modifications by moving data. Instruction 107, in the example above, illustrates this in a simple way: If the registers $700 + j$ themselves had been used in the summation process, (step 105) both the B- and C-addresses of step 105 would have required modification in the outer loop, as well as the C-address of instruction 103. For cases where several numbers are to be moved at the same time, a more general MOVE operation than the MOVE 000 used so far is available and will be described in the next section.

IV. 4. MOVE

The MOVE operation is defined as follows:

Numer.	Alpha.	Function
$0_1 = 9$	MOVE	Move. If $A \neq 000$, the block of A consecutive words beginning at B is moved into the set of A consecutive locations beginning at C. The words in the original locations are not destroyed, except where the two regions overlap. The number in location 000 ("previous result") is not affected when $C \neq 000$. Both $C > B$ and $C < B$ are permissible. An error stop occurs if $C + A - 1 \geq 1000$. If $A = 000$, the word $(\bar{B})$ specified by the B-address is moved into location C and into 000. It also remains in location B.

MOVE with $A = 000$ differs from MOVE with $A = 001$ only in that the execution time with $A = 000$ is shorter and that the previous result location is affected.

Note: If a number is to be moved from location B into 000 for use in a TR SGN or TR EXP operation on the next step, MOVE 9 001 B 000 must not be used, since these transfer operations work strictly according to their definitions (see Sec. IV. 1.). The correct instruction would be MOVE 9 000 B 000. (Internally, these transfer operations inspect a duplicate "previous result" location rather than 000!)

As an example, suppose x_1 is in 701, x_2 in 702, ..., x_5 in 705 and the instruction MOVE 9 005 701 703 is given. Then x_1 will be found in 703, x_2 in 704, ..., x_5 in 707, after execution.

In conclusion, it should be pointed out that the use of the logical operations is by no means restricted to the straightforward functions for which they are primarily intended. The programmer will find innumerable ways of increasing the efficiency and elegance of his programs by unusual applications, particularly of the address change operations. As a weird example, suppose it is desired to multiply the numbers located in registers 1, 4, 9, 16, 25, 36, ..., 400 (!) by a constant located in 600 and store the results in 501, 502, 503, ..., 520:

Instr.	Alpha.	Sign	0_1	A or 0_2	B	C
898	SET A	+	0	500	900	001
899	SET C	+	0	005	901	003
900	MPY	−	3	[ ]	600	501
901	ADD A	+	0	600	900	[ ]
902	ADD C	+	0	006	901	002
903	LOOP C	+	0	001	020	900
904	Next instruction in the problem.					

V INPUT-OUTPUT OPERATIONS

V. 1. CARD FORM

By a card form is meant a specific assignment of card columns to form fields for data, instructions, identification, etc., in connection with a given program or interpretive system. In the 650, information is transmitted to and from cards through a control panel, and anyone whose needs call for a special card form can adapt it for use in connection with the interpretive system merely by simple control-panel wiring. For most needs, the following card form, associated with the interpretive system, is likely to be found adequate. At this point, only brief definitions of the card fields will be given for reference in subsequent sections where their use will be explained in detail:

Columns	Definition
1-4	Card number
5-6	Deck number
7-9	Location
10	Word count
11	Sign of word 1
12-21	Word 1
22	Sign of word 2
23-32	Word 2
33	Sign of word 3
34-43	Word 3
44	Sign of word 4
45-54	Word 4
55	Sign of word 5
56-65	Word 5
66	Sign of word 6
67-76	Word 6
77-79	Problem number
80	Tracing identification

The same card form is used in all input-output operations, as well as in tracing. Both instructions and data are signed ten-digit words and are entirely indistinguishable in connection with input-output operations.

V. 2. PUNCHING

At any point in the problem, the machine may be ordered to punch into cards the contents of any set of memory locations, together with appropriate identification. In some problems, it may be desirable to punch out answers one at a time, perhaps together with the values of relevant parameters; in others it may be preferable to punch out a large amount of information at less frequent intervals. There are also cases where it is advantageous to punch out instructions: In connection with testing (see PROGRAM TESTING) in order to examine a program interrupted at a chosen point, and in connection with loading,(see LOADING) in order to reduce the size of a deck of cards. All of these ends are served by the following instruction:

$0_2 = 410$	PCH	Punch cards. The block of consecutive words beginning at B and ending at C (inclusive) is punched into cards. Five words and a word count of 5 are punched into each card but the last, whose word count will be the remainder when C-B+1 is divided by 5. On each card, the location from which word 1 was punched is punched into columns 7-9. The words in storage are not destroyed. A cumulative count of the number of cards punched during the problem (i. e., since LOADING) is punched into columns 1-4. The problem number (see LOADING) is punched into columns 77-79 and zero is punched into columns 6 and 80. An error stop occurs if B > C.

If it is desired to punch six words to a card, this may be done by adding a special card behind the punching deck (see LOADING). This card should have an x-punch in column 5, 1969 in columns 6-9, 1 in column 10, a 12 punch in column 11, and 00 0006 0000 in column 12-21.

The punched cards are likely to be used for one (or both) of two purposes (in addition to possible processing on other equipment): The information on them may be printed on a tabulator or they may be loaded (or READ) into the 650 at a later time. Details of the printing will not be given here, since they depend on the characteristics of the tabulator, but the printing form may be assumed to be roughly identical with the card form with proper spacing between words. (Suggestions on tabulator wiring are given in Sec. XI. 5.) It is assumed that the suppression of the superfluous words punched into the last card, if its word count is not 5, will be performed on the tabulator control panel. If this is not feasible, it may be done in the 650 by adding three cards to the punching deck. For details, see Sec. XI. 1.

Selective spacing between lines in printing may be accomplished in several ways, even though no operation in the 650 is provided for this purpose. A brief discussion will be given here, since spacing considerations may affect the use of the PCH operation in programming. Through the setting of switches on the tabulator, a choice of any of the following spacing alternatives may be provided:

(a) Single or double spacing.

(b) Spacing between every n lines (with n chosen by wiring, normally, e. g., n = 10).

(c) Spacing after any line whose word count is less than the word count of the preceding line.

(d) Spacing before any line whose location number has a units digit smaller than the units digit of the location number of the preceding line.

Alternative (c) is suited for the printing of information punched from fairly large blocks of locations by one PCH order. Spacing will occur after each block, unless the block length is a multiple of 5, which can be avoided by programming. Alternative (d) is intended for information punched repeatedly from the same set of locations and provides the option of spacing when the loop is interrupted, e. g., for changing a parameter value.

V. 3. LOADING

When a program has been written, and careful inspection reveals no further errors, it is key punched into cards following the card form given in Sec. V. 1. To reduce to a minimum the number of errors to be found with the aid of the 650, the cards should be run through a verifier operated by another person or, alternatively, key punched independently by two operators and compared on a reproducer. The programmer has the option of specifying the number of words to be punched to a card: Punching 5 or 6 to a card will keep the program deck small from the outset and eliminate the need for condensing the deck on the 650 later. Punching one word to a card is felt by some programmers to facilitate changes. Each card must have in columns 7-9 the location into which word 1 is to be loaded, and in column 10 the number of words to be loaded from the card into consecutive locations. Columns 1-6 and 77-80 are not read by the 650 (except that the problem number is read from the last card, see below) and may be used by the programmer as he deems best. Each column of each field to be used by the machine must contain one and only one punch and an error stop is provided to enforce this rule. A 12-punch is used for plus, an 11- or x-punch for minus and a 0-punch--not a blank column--for zero. If the word count is less than 6, unused word fields and sign columns may be left blank. No distinction is made between data and instructions in key punching and loading.

LOADING is the process of feeding data and instructions into the machine at the beginning of a problem. If the previous user of the 650 was not using the interpretive system or if there is any reason to doubt that the system is correctly stored on the drum, the program deck should be preceded in loading by a deck which loads the interpretive system (in 51. 9 seconds) into the memory locations above 999. Before the program deck, the programmer may also place a Reset Memory Card, which will (in 6. 3 seconds) reset each of the memory locations 001-999 to minus zero. (This is useful in connection with the punching out of sections of memory in testing.) Immediately behind the program deck--no blank cards are used in the card reader in connection with this interpretive system--the programmer places one of two nine-card decks to inform the machine whether he wants normal operation or TRACING described in a later section. (If he knows that he wants the same mode of operation as the previous user, he can omit these cards but the gain is only 2.7 seconds.) Last, he must place a so-called transfer card with a zero punched in column 10, the problem number in 77-79 and the location of the instruction at which the program begins in columns 7-9. The word fields on this card may be left blank.

The loading program automatically resets the loop box, the card counter (see PUNCHING) and location 000 to zero.

The order in which the program cards are loaded is irrelevant, unless the same location is loaded into from more than one card, in which case the last such card, of course, determines the contents of the location. This may occur in connection with changes of a temporary nature, which may be placed at the end of the deck and later removed, leaving the program in its original form. In the deck which loads the interpretive system, the order of the cards must be preserved, and an error stop is provided to insure this, thereby ascertaining that no part of the system is missing.

In summary, a complete deck to be loaded must contain:

System deck (173 cards)
Reset Memory card (optional)
Program deck
Mode-of-operation deck (9 cards)
Transfer card

The control console of the 650 need be of almost no concern to the user of the interpretive system under normal conditions. He must only make sure that all switches on the console are set in a fixed manner required by the system, and these settings will now be listed without any description of the function of the switches. Certain ways of using the console are described in the sections on READING and PROGRAM TESTING.

Switches	Settings
Storage entry	70 1951 1333 +
Programmed stop	Stop (see COND STOP)
Half cycle	Run
Address selection	1338 (see STOPS)
Control	Run
Display	Upper Accumulator
Overflow	Stop
Error	Stop

To start a problem, the deck to be loaded is placed in the card reader, and the following keys are depressed in order:

(1) COMPUTER RESET	(on the console)
(2) PROGRAM START	(on the console)
(3) START	(on the card reader)

When the last card leaves the hopper, the machine stops and the key labelled

(4) END OF FILE	(on the card reader)

is depressed. If the deck has been correctly put together, the execution of the program will then start automatically.

The program deck may be run out at any time after loading by depressing the START key, unless a READ instruction is contained in the program. Blank cards should be inserted into the PUNCH hopper and the START key on the punch side depressed.

To make the 650 produce a condensed program deck in case the program was originally key punched one instruction to a card, a PCH instruction should be given at the very beginning of the program. This instruction may be bypassed during subsequent executions of the program merely by changing the location number on the transfer card.

V. 4. READING

In some problems, particularly in applications of a data processing nature, it may be desirable to read information into the machine during the execution of the program without manual interference. This is accomplished by the READ operation:

$0_2 = 400$	READ	Read cards. The block of consecutive storage locations beginning at B and ending at C (inclusive) is read into from cards. The address B must appear in the location field on the first card, as well as in the READ instruction, and the location field on each card following must contain the sum of the word count and location on the previous card. The sum of the word counts of all cards to be read must be C-B + 1. Violations of these requirements, which have been included for the programmer's protection, will result in error stops.

The cards to be read should be placed in the hopper of the card reader immediately following the transfer card (no blank cards).

The decisions made with the aid of conditional transfers and other logical operations are normally based on criteria predetermined by the programmer and incorporated into the program. If the programmer wishes to influence the program during its execution, e.g., on the basis of a result displayed on the console in connection with a COND STOP instruction, he may do so using the CONS operation:

0_2 = 401	CONS	Read console. The machine stops if the PROGRAMMED STOP switch is on STOP. Zero is displayed on the display lights and 1131 on the address lights. When the PROGRAM START key is depressed, the number entered on the STORAGE ENTRY SWITCHES is stored in location C and in 000 (the "previous result" location). If the PROGRAMMED STOP switch is on RUN, the storing takes place without a stop preceding. The B-address is ignored but should be filled, e.g., with zeros.

An example of an application of CONS might be the feeding in of an "educated guess" for a starting value in connection with the solution of algebraic equations. Another application, involving only the storage entry SIGN switch, might be to continue a program until another user is ready to take the machine, at which time a change in the SIGN switch setting, interpreted by a TRSGN operation, causes the program to punch out intermediate results for later restart.

VI PROGRAM TESTING

VI. 1. MEMORY PRINT-OUT

The choice of methods for testing ("debugging") a program by comparing results of machine calculation to known quantities or to results of independent calculations by other means is governed by the relative availability of machine time and programmer time. If machine time is freely available, testing with the aid of the control console is highly efficient, as well as instructive and enjoyable, as soon as a certain facility for operating the console has been acquired. Particularly in the case of small problems, the method of tracing--where a card is punched for each instruction executed, showing all numerical and logical quantities associated with the execution--may be the most desirable in that it gives an almost certain clue to the difficulty within a predictable, if not very short, period of machine time and allows the programmer to study the material at his leisure. The method most economical of machine time and yet frequently sufficiently illuminating is that of memory print-out. It might be suggested that on most problems in a busy but not heavily over-loaded installation, the methods be used in the order reverse to that in which they were mentioned here. Some directions for their use will now be given.

The memory print-out method simply consists of inserting temporarily into the program at one or several suitably chosen points PCH orders (see PUNCHING) calling for the punching of blocks of information--data or instructions--which, when printed on the tabulator, will give a picture of the progress of the program. Since 1000 words may be punched 5 to a card in two minutes, it is not out of the question to punch out the contents of every register used in a problem--including all the instructions--several times. To get the most benefit from this method, the programmer should, in any problem that does not threaten to fill the entire available memory, avoid using the same storage location for storing different quantities at different times whenever feasible, so that as many partial results as possible are preserved for the memory print-outs. Whenever a test case of a problem is run, even if memory print-out is not chosen as the primary testing method, it would certainly be advisable to make the last instruction of the test deck punch out the entire memory used. A flexible alternative would be to have scattered through the program CONS--TR SGN combinations which transfer control to a PCH order if the storage entry sign switch is turned to minus.

Temporary instructions may be inserted into a program in two ways: Either they are included in the normal sequence of instructions when the program is initially written and replaced either by NOOP instructions (see Sec. IV. 1) or by transfer to the next non-temporary instruction when no longer needed, or else one of the regular instructions of the program is replaced by a TR to a vacant location L, the regular instruction is placed in L, the temporary ones in L + 1, L + 2, etc., and at the end of this temporary sequence a TR back to the normal program is given. In either case, the temporary instructions may (as suggested in LOADING) be kept as a separate deck at the end of the program deck, eliminating any changes in the main program deck and simplifying bookkeeping.

VI. 2. TRACING

If the tracing deck of nine cards is loaded with the program deck, (see LOADING) the machine will automatically start tracing from the beginning of the program, as specified by the transfer card. Before the execution of each instruction, a card with the following information will be punched:

Columns	Definition
1-4	Card number (cumulative)
6	Zero
7-9	Location of the instruction about to be executed.
10	Six
11-21	The instruction as stored in memory.
22-32	The instruction as modified for execution (i. e., with the loop box added if minus).
33-43	The contents of the loop box.
44-54	$\bar{A}$ if A ≠ 000, zero if A = 000.
55-65	$\bar{B}$
66-76	The contents of location 000 (i. e., the result of the last mathematical MOVE 000 or CONS operation).
77-79	Problem number.
80	Eight (used by the tabulator for automatic selection of a different printing form for trace cards).

The punching rate will be 100 cards per minute except in the case of very time-consuming operations, such as, the moving of a large block of information. The advantage of punching the trace card before execution is that information will be punched for an instruction whose execution is interrupted by an error stop. In the case of instructions (such as LOOP or TR EXP) whose B-address does not refer to a memory location, the quantity $\bar{B}$ is irrelevant. Tabulator wiring to suppress the printing of $\bar{B}$ in such cases can be provided if sufficient selection equipment is available. The PCH operation is bypassed when the machine is operating in the tracing mode, i.e., PCH is equivalent to NOOP.

If a program is too long to be traced in its entirety or if this is unnecessary, selective tracing may be effected by using the following operations:

$0_2 = 450$	START TR	Start tracing. If the nine-card tracing deck has been loaded, the machine will start tracing from the next instruction. If it is already tracing, it will continue to trace. The B and C addresses are ignored. If the deck for normal operation has been loaded, START TR will be equivalent to NOOP.
$0_2 = 451$	STOP TR	Stop tracing. If the machine is tracing, it will discontinue tracing immediately. If it is not tracing, STOP TR will be equivalent to NOOP. The B and C addresses are ignored.
$0_2 = 452$	ST TR ERAS	Start tracing and erase itself. If the tracing deck has been loaded, the machine will start tracing from the next instruction. If it is already tracing, it will continue to trace. If the deck for normal operation has been loaded, tracing will not begin. In all cases, the ST TR ERAS instruction will be replaced in memory by a NOOP ($0_2 = 454$) during its first (and only!) execution. The B and C addresses are ignored.

The bypassing of the PCH operation is in effect as long as the trace program is on the drum and is not affected by the selective tracing orders. To make PCH operative, the nine-card deck for normal operation must be loaded.

The purpose of the ST TR ERAS operation is to make it possible to trace the repetitive steps of a loop either once or twice and then stop tracing until the loop is completed. To get the steps traced once, one may place the pair STOP TR, ST TR ERAS at the beginning of the repeated portion of the loop; to get them traced twice, one places this pair of instructions at the end immediately preceding the test. As a specific example, suppose it is required to trace twice the steps of the loop programmed in the section on LOOP OPERATIONS and suppose vacant locations are available from 900 up. Assume that the machine is tracing as it enters the loop. The original program reads as follows:

101	MPY	—	3	801	901	000
102	ADD	+	1	000	700	700
103	LOOP AB	+	0	110	020	101
104	Next instruction in the problem.					

The following instructions could be added as a temporary deck at the end of the program deck:

102	TR	+	0	203	000	900
900	ADD	+	1	000	700	700
901	STOP TR	+	0	451	000	000
902	ST TR ERAS	+	0	452	000	000
903	TR	+	0	203	000	103

Notice that the TR instruction gets loaded into 102 after the regular program, replacing the ADD instruction, as explained in LOADING. This example is, of course, unrealistic in that selective tracing would hardly be needed for testing such a simple loop.

VI. 3. CONSOLE TESTING

Testing with the aid of the control console requires some familiarity with the internal structure of the interpretive system (see Sec. XI) and with the basic language of the 650. Console testing is more attractive on the 650 than on most machines due to the ADDRESS STOP feature: If the CONTROL switch is turned to the ADDRESS STOP position, the execution of the program will proceed at electronic speed until the address set up on the ADDRESS SELECTION switches is reached. At that point, the machine stops, and the contents of various registers may be examined on the display lights or the program may be continued manually one step at a time. Alternatively, the program may be punched out on cards at this point by merely feeding in one card with a PCH instruction, going into any vacant location, followed by a transfer card specifying this location. Console testing, in connection with the interpretive system, is likely to be needed only in exceptional cases.

The ADDRESS STOP feature of the 650 may be used in conjunction with a special address stop transfer card when it is desired to start tracing from a certain instruction N in the middle of a program after running at full speed up to that point. (This may, of course, alternatively be accomplished using the tracing operations described in Sec. VI. 2, but then the value of N must be decided upon in advance and the proper program changes key punched.) The procedure is as follows: Set the ADDRESS SELECTION switches to N and turn the CONTROL switch to ADDRESS STOP. Load as usual and run until the machine stops at the instruction N. (For details on possible earlier stops see below.) Then set the CONTROL switch to RUN and load the tracing deck followed by the address stop transfer card. Tracing will begin immediately and the first instruction traced will be N.

In choosing N it must be remembered that the loop box and location 000 are reset to zero when the tracing deck is loaded. If this restriction is inconvenient, it can be circumvented by placing a special card in front of the tracing deck. The card counter and the problem number are also reset to zero, unless the tracing deck has been modified to prevent it.

If the CONTROL switch is kept in the ADDRESS STOP position when the program deck is loaded, one stop will occur when location N is reset by the memory reset card and another when the programmer's instruction is loaded into N. Also, stops may occur before instruction N is reached in the program, if N is referred to in an ADDRESS CHANGE or MOVE operation (but not if N is one address in a conditional transfer instruction and control is transferred to the other address). After each stop, operation will resume when the PROGRAM START key is depressed. If the CONTROL switch is left in the ADDRESS STOP position during tracing, two stops will occur each time N is referred to (and one if N is the B-address of a transfer instruction).

The program can be continued at full speed (punching mode) after a period of tracing by following the procedure described above with the punching deck in place of the tracing deck.

The address stop transfer card has 69 1976 1952 24 1061 1098 in columns 1-20 and a 12-punch in each of columns 1, 10 and 20. The special card for bypassing the resetting steps in loading has 69 1953 1952 24 1278 1953 70 1951 1344 in columns 1-30 and a 12-punch in each of columns 1, 10, 20 and 30. (See Deck 7, Sec. XI. 7.)

If the value of N has been decided upon in time to get it key punched into a regular transfer card, (Sec. V. 3) this card may, of course, be used in place of the address stop transfer card in the procedure described above.

A programmer familiar with the internal structure of the interpretive system will be able to think of many other cases where special needs can be met using maching language cards ("load cards").

VII SUMMARY OF OPERATION CODES

0_1 OPERATIONS

Num.	Alpha.	Page Ref.
0	GO to 0_2	8
1	ADD	10
2	SUB	10
3	MPY	10
4	DIV	10
5	NGMPY	10
6	TR A	19
7	TR B	20
8	TR C	20
9	MOVE	23

0_2 OPERATIONS

Num.	Alpha.	Page Ref.
000	UNC STOP	13
200	COND STOP	13
201	TR SGN	14
202	TR EXP	14
203	TR	14
204	TR SUBR	15
205	TR OUT	15
100	LOOP A	16
010	LOOP B	16
001	LOOP C	16
110	LOOP AB	17
101	LOOP AC	17
011	LOOP BC	17
111	LOOP ABC	17
500	SET A	18
050	SET B	18
005	SET C	18
600	ADD A	19
060	ADD B	19
006	ADD C	19
700	SUB A	19
070	SUB B	19
007	SUB C	19

Num.	Alpha.	Page Ref.
300	SQRT	11
301	EXP E	11
302	LOG E	11
303	SIN R	11
304	COS R	11
305	ART R	11
350	ABS	11
351	EXP 10	11
352	LOG 10	11
353	SIN D	11
354	COS D	11
355	ART D	11
400	READ	29
401	CONS	30
410	PCH	26
450	START TR	33
451	STOP TR	33
452	ST TR ERAS	33
454	NOOP	15

VIII STOPS

Error circuits in the 650 will stop the machine on attempts to use invalid information, such as, that represented by blank columns or double punches, as well as on several kinds of machine malfunctioning, and will indicate on the control console the nature of the error. If this occurs during the loading of a new deck, the cards should be examined. In other cases, a note should be made of the indications on the console, and the procedure that led to the stop should, if possible, be repeated exactly in order to determine whether the error is systematic in nature.

All stops, which are part of the interpretive system, will now be listed. Conditional stops will occur only if the PROGRAMMED STOP switch is set to STOP. On a conditional stop, the PROGRAM LIGHT in the OPERATING section of the console will be on and no lights in the CHECKING section should be on. The program will continue if the PROGRAM START key is depressed. On an unconditional stop, the STORAGE SELECTION light in the CHECKING section will be on. Normally, operation should be discontinued after an unconditional stop and changes made in the program in order to avoid the stop. Alternatively, the program may be continued by having a transfer card (see LOADING) in the card reader, specifying the instruction to which control should proceed when the COMPUTER RESET and PROGRAM START keys are depressed.

The location of the interpretive system instruction xxx on which the machine has stopped, may be determined by displaying the contents of location 1098 on the console. The display lights will show 60 0xxx 1107. This process, called "monitoring", may be performed as described in the 650 manual or, alternatively, by setting the storage entry switches to 60 0xxx 8000 and depressing the COMPUTER RESET, PROGRAM START and PROGRAM STOP keys.

If, in an exceptional case, it would be advisable to proceed to the next instruction after an unexpected unconditional stop, this may be done manually as follows:

(1) Set the CONTROL switch to MANUAL.
(2) Check that the ADDRESS SELECTION switches are set to 1338.
(3) Depress the COMPUTER RESET key.
(4) Depress the TRANSFER key.
(5) Set the CONTROL switch to RUN.
(6) Depress the PROGRAM START key.

As a result of this procedure, zero will be stored at C and 000 before the next instruction is executed. If this is not desired, the ADDRESS SELECTION switches should be set to 1095 in step (2). To repeat the same instruction (on which the stop occurred) the switches are set to 1098.

There is an alternative manual procedure for restarting after an unconditional stop which is simpler in the case of frequent use but is not recommended in general because it requires changing the setting of the STORAGE ENTRY switches. They are used in LOADING and must be set back to their normal positions for the next user:

(1) Set the STORAGE ENTRY SWITCHES to 00 1951 1338+ (or 00 1951 1095+ if zero is not to be stored or 00 1951 1098+ to repeat).
(2) Depress the COMPUTER RESET key.
(3) Depress the PROGRAM START key.

CONDITIONAL STOPS

Address Lights	Normal Cause
1120	Programmed COND STOP. (Display lights show $\overline{B}$.)
1131	CONS (Check STORAGE ENTRY switch setting.)
1715	Loss of accuracy in SIN (Exponent of $\overline{B}$ exceeds 52) or COS.
1835	Loss of two or more digits of accuracy in LOG.

UNCONDITIONAL STOPS

2222	MOVE with $C + A - 1 \geq 1000$. PCH with $B \geq C + 1$. READ with incorrect loc. or word count.
3333	DIV with $\overline{B} = 0$.
4444	SQRT with $\overline{B} < 0$.
5555	MPY with result out of range. DIV with result out of range.
6666	EXP with result out of range. LOG with $\overline{B} \leq 0$. SIN with exp. of $\overline{B}$ exceeding 58. COS with exp. of $\overline{B}$ exceeding 58.
7777	Cards missing or out of order in the system deck being loaded.
9999	Programmed UNC STOP (Display lights show $\overline{B}$).

IX EXECUTION TIMES

The execution times listed in this section are based on the standard 650 drum speed of 12, 500 r.p.m. They represent approximate theoretical estimates derived, in the case of the mathematical operations, from simple assumptions regarding the distribution of the numbers to be operated upon. For example, the part A_1 of a floating-decimal number $\bar{A} = A_1 \cdot 10^{a_1}$ is assumed to be uniformly distributed between 1 and 10, although in physical problems there are reasons that favor a logarithmic distribution; extremely small and extremely large exponents are considered very unlikely, etc. It is further assumed that the programmer has chosen storage locations on the drum without regard to timing, ignoring the fact that in the case of some operations the execution time will be a few milliseconds shorter for numbers stored in certain sections of memory. Some, but not nearly all, of the time estimates have been verified by tests.

It should be stressed that the estimates of execution times are needed only for making comparisons or estimates of running time for problems or for choosing efficient ways of programming and will never affect the result of an operation. In comparing these estimates to estimates given for other interpretive systems or subroutines, it is important to verify by sample calculations or machine tests that the assumptions are realistic.

To minimize the size of the table, the execution times listed refer to a basic case and corrections to be added in other cases are given at the beginning of the table.

650 INTERPRETIVE SYSTEM

ESTIMATED AVERAGE EXECUTION TIMES IN MILLISECONDS

(a) If A ≠ 000, add 7.2 ms. for ADD and SUB, 6.3 ms. for MPY, NGMPY and DIV*.

(b) If C ≠ 000, add 6.1 ms. for all mathematical operations, MOVE 000 and CONS*.

(c) If the instruction has a minus sign, add 4.8 ms. for all operations.

(d) If, after a TR EXP or LOOP operation, control will proceed to the next instruction rather than to C, add 4.8 ms.

ADD	65.7	UNC STOP	28.8	SQRT	206
SUB	65.7	COND STOP	29.8	EXP E	197
MPY	67.2	TR SGN	19.2	LOG E	202
DIV	74.3	TR EXP	24.0	SIN R	192
NGMPY	67.2	TR	19.2	COS R	187
		TR SUBR	44.4	ART R	238
		TR OUT	26.0	ABS	33.2
				EXP 10	187
TR A	37.3	LOOP A	24.0	LOG 10	207
TR B	37.3	LOOP B	28.8	SIN D	240
TR C	42.1	LOOP C	24.0	COS D	235
		LOOP AB	24.0	ART D	271
		LOOP AC	24.0		
MOVE 000	37.7	LOOP BC	28.8	READ	One card:
MOVE	40.8 + 12A	LOOP ABC	24.0		101 + 14n
	(A = no. of				(n = no. of words.)
	words.)	SET A	55.3		Succeeding cards:
		SET B	55.3		300 each.
		SET C	55.3	CONS	28.8
		ADD A	44.9	PCH	One card:
		ADD B	44.9		163 + 12.5n
		ADD C	44.9		(n = no. of words.)
		SUB A	44.9		Succeeding cards:
		SUB B	44.9		600 each.
		SUB C	44.9		
				START TR	28.8
				STOP TR	24.0
				ST TR ERAS	38.9
				NOOP	24.0
				TRACING	600 per card.
				LOADING	300 per card.

*(See next page for footnote.)

---

*Those who are particularly interested in time considerations may wish to know the exact increments on which the weighted averages in (a) and (b) are based:

In ADD and SUB, 4.8 ms. if $17 \leq A \leq 41$ (mod 50)
9.6 ms. if $1 \leq A \leq 16$ or $42 \leq A \leq 49$ (mod 50)

In MPY, NGMPY and DIV, 4.8 ms. if $A \geq 17$ or $A = 1$ (mod 50)
9.6 ms. if $2 \leq A \leq 16$ (mod 50)

In all mathematical operations, MOVE 000 and CONS:
4.8 ms. if $7 \leq C \leq 42$ (mod 50)
9.6 ms. if $1 \leq C \leq 6$ or $43 \leq C \leq 49$ (mod 50)

An easily remembered programming rule could be extracted from this information: If locations between 17 and 41 (mod 50) are used for storing numbers, the increments given in (a) and (b) may be replaced by 4.8 ms.

X SPECIAL TOPICS IN PROGRAMMING

X. 1. SUBROUTINES; TRANSLATION

A subroutine is a program expected to be of use as a part of the program in several problems or in several sections of the same problem. The mathematical operations in the interpretive system are indeed subroutines written in the basic language of the machine and reached through their operation codes, and anyone desirous of preparing an additional subroutine of this type may avail himself of a vacant operation code (see Sec. XI) and write the program in a part of the memory below 1000.

Subroutines written wholly or partly in the interpretive language may be reached conveniently using the TR SUBR operation defined in Sec. IV. 1. Suppose the subroutine begins at 900 and ends at 935. Instruction 900 should read: SET C 0 005 935[ ] and instruction 935 should read: TR 0 203 000[ ]. Now suppose the subroutine is needed at step 700 in a program, and when it has been used, control is to be transferred to 680. Instruction 700 should read: TR SUBR 0 204 680 900. The TR SUBR operation will take the quantity ("return address") 680, place it in the C-address of instruction 900 and then transfer control to 900. Instruction 900, in turn, places 680 in the C-address of 935, and when instruction 935 is reached at the end of the subroutine, it transfers control to 680 as originally desired. Hence, the programmer using the subroutine only needs to know the identifying number 900; the transfer of control to and from the subroutine is handled by the TR SUBR in conjunction with the two instructions 900 and 935 provided by the writer of the subroutine. Subroutines needing only one input number and giving only one result (such as, the evaluation of one Bessel function for a given value of the argument) will normally assume the input to be in 000 and will deliver the result there; in the case of several numbers, specified locations normally within the block occupied by the subroutine would be used for input and/or results. Subroutines may, of course, be used inside other subroutines without restriction.

If the locations occupied by a subroutine are needed for another purpose, e.g., another subroutine in the same problem, the subroutine may be translated to a different set of locations by a translating program developed by Miss D. C. Leagus. When the subroutine is written entirely in the interpretive system, the programmer is required only to separate data and constants from instructions, and the translating program will automatically decide which addresses of each instruction are subject to translation. Machine language instructions may also be used in a subroutine to be translated, provided certain conditions specified by the translating program are adhered to.

Subroutines for the solution of cubic equations and of systems of linear equations have been written at the Laboratories.

X. 2. UNNORMALIZED INPUT; TRANSITION BETWEEN FLOATING- AND FIXED-DECIMAL FORM

Nearly all of the mathematical operations in the system assume that the floating decimal numbers to be operated upon are in the normalized form defined in Sec. II. 2, i. e., that the leading digit is different from zero unless the entire number is zero. In processing empirical data, key punching is often facilitated by permitting leading zeros and reproducing a constant exponent. Such unnormalized data may be used in the interpretive system provided the first operation in which it is used is ADD or SUB with operand exponents differing by less than 10.

A special case of unnormalized input is that of a zero with a non-zero machine exponent. If such a zero is added to a non-zero number with a smaller exponent, a number of digits equal to the difference between the exponents are lost. Consequently, zero should be equipped with exponent 00 unless the programmer knows in detail how the zero will be used in his program. Special provisions in the MPY and DIV routines make it possible to use a zero with machine exponent 00 in them without danger of exceeding the exponent range negatively.

The converse problem of producing unnormalized output, e. g., for the printing of tables in fixed-decimal form or for calculations in machine language is easily solved at the expense of one digit. Suppose for example that the numbers N_i to be "unnormalized" or "fixed" are known to be less than 10^4 and output in the form XXXX. xxx is desired. Add the number 1000000054 (i. e., 10, 000. ooo) to N_i if $N_i \geq 0$, subtract it from N_i if $N_i < 0$ (using TR SGN) and punch. The output of the form $\pm(10,000 + |N_i|)$, is ready to be printed on the tabulator with the leading 1 and the constant exponent 54 suppressed by hammerlocks or wiring. If the numbers are to be used in machine language, the 1 and 54 are shifted out. Rounding to a smaller number of digits is obtained by choosing the exponent of the additive constant (1000000054) correspondingly larger.

X. 3. EXAMPLES

In conclusion, two problems will be programmed in order to illustrate the use of many of the operations and methods described.

First, suppose it is desired to evaluate the "error function",

$$(1) \quad \varphi(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

for a set of values $x = a, a+\Delta, a+2\Delta, \ldots, a+10\Delta$, using the RAND approximation

$$(2) \quad \varphi^*(x) = 1 - (a_1\eta + a_2\eta^2 + a_3\eta^3 + a_4\eta^4 + a_5\eta^5)\,\varphi'(x),$$

where

$$\text{(3)} \quad n = 1/(1+px), \ (p \text{ is a numerical constant}),$$

and

$$\text{(4)} \quad \varphi'(x) = \frac{2}{\sqrt{\pi}} e^{-x^2},$$

and to punch out the results as well as to store them for later use. The evaulation of the polynomial in n will be faster if (2) is written in the form

$$\text{(5)} \quad \varphi^*(x) = 1-(n(a_1+n(a_2+n(a_3+n(a_4+n\,a_5))))) \varphi'(x).$$

To make it possible to use the LOOP order in evaluating $\varphi^*(x)$ this way, the coefficients a_i will be stored in consecutive locations in decreasing order. The LOOP program will be given a form applicable to an arbitrary polynomial by including a "dummy" coefficient $a_o = 0$. Storage locations will be chosen as follows:

Location	Contents	
101-119	instructions	(cards 1 - 4)
200	$2/\sqrt{\pi}$	constants (card 5)
201	1	
202	a	
203	Δ	
204	p	
221	x	temporary storage ("erasable")
222	$\varphi'(x)$	
223	n	
301	a_5	coefficients in $\varphi^*(x)$ (card 6)
302	a_4	
303	a_3	
304	a_2	
305	a_1	
306	$a_o = 0$	
401-410	$\varphi^*(x)$	results

The program might be written as follows:

Card	Loc.	Alpha.	Sign	0_1	A or 0_2	B	C	Comments
1	101	SET C	+	0	005	114	401	Set address of first $\varphi^*(x)$
	102	MOVE	+	9	000	202	221	First x is x = a
	103	NGMPY	+	5	000	000	000	$-x^2$
	104	EXP E	+	0	301	000	000	e^{-x^2}
	105	MPY	+	3	000	200	222	$\varphi'(x) = \frac{2}{\sqrt{\pi}} \cdot e^{-x^2}$
2	106	MPY	+	3	204	221	000	px
	107	ADD	+	1	000	201	000	1+ px
	108	DIV	+	4	201	000	223	$\eta = 1/(1+px)$
	109	MOVE	+	9	000	301	000	a_5 into 000 for LOOP
	110	MPY	+	3	000	223	000	prev. res.. η
3	111	ADD	−	1	000	302	000	add next coeff.
	112	LOOP B	+	0	010	005	110	loop in polynomial eval.
	113	NGMPY	+	5	000	222	000	polyn. $\cdot\varphi'(x)$
	114	ADD	+	1	000	201	[ ]	$\varphi^*(x)$ = 1+ prev. res.
	115	ADD	+	1	221	203	221	$x + \Delta$ = next x
4	116	ADD C	+	0	006	114	001	next $\varphi^*(x)$ address
	117	TR C	+	8	114	412	103	test for end
	118	PCH	+	0	410	401	410	punch two cards
	119	COND STOP	+	0	200	221	500	end; stop, display last x, go to 500 on PROGRAM START

500 Next instruction in the problem.

An important remark should be made: If there is no shortage of storage locations and if the programmer does not mind writing a somewhat larger number of instructions, the running time for many problems can be decreased and the logic simplified by "unwinding" the innermost loop, i.e., by writing out the mathematical instructions in the loop in a straight sequence instead of using the LOOP operation. In the present problem, a sequence containing five MPY and four ADD instructions could replace the instructions 109-112 and also eliminate the use of the dummy coefficient a_0. The execution time for the polynomial loop would be reduced by nearly 1/3 and the LOOP operation could be used to replace the address change operations in the outer loop. The polynomial evaluation accounts for about 1/2 of the total running time of this problem. In many large problems, the innermost loop consumes an even larger fraction of the running time, making it important to program the innermost loop efficiently even at the expense of apparent inefficiencies elsewhere.

The second illustrative problem reads as follows: For a given set of numbers x_v, v = 1, 2, . . . , 50, not necessarily equally spaced, the values of the Chebyshev polynomials $T_n(x_v)$, n = 1, 2, . . . , 10, are to be computed using the recursion formula

$$(6) \quad T_{n+1}(x_v) = 2x_v T_n(x_v) - T_{n-1}(x_v),$$

($T_o(x_v) \equiv 1$, $T_1(x_v) \equiv x_v$) and punched out in a compact form.

In addition, the sum

$$(7) \quad \sum_{v=1}^{50} \frac{[T_{10}(x_v)]^2}{\sqrt{1-x_v^2}} (x_{v+1} - x_v),$$

($X_{51} = 1$) is to be punched out and the operator is to be given the option of also calling for the punching of partial sums of (7) at any time.

Storage locations will be assigned as follows:

Location	Contents		
050	0	constants (card 7)	
051	1		
052	2		
095-120	instructions (cards 1 - 6)		
199	The sum (7) and its partial sums		output
200	$T_o(x_v) \equiv 1$		
201	$T_1(x_v) \equiv x_v$		
202	$T_2(x_v)$		
---	-----		
210	$T_{10}(x_v)$		
300	$2x_v$		temporary storage ("erasable")
301	$[T_{10}(x_v)]^2$		
302	$[T_{10}(x_v)]^2/\sqrt{1 - x_v^2}$		
400 + v	x_v		input (cards 8 - 18)
451	1		

In addition, locations 1-8 will be used in connection with a trick in programming the LOOP.

The program may be written in many ways. The following is not necessarily the best:

Card	Loc.	Alpha.	Sign	0_1	A or 0_2	B	C	Comments	
1	095	MOVE	+	9	000	051	200	$T_o = 1$	Outer loop initialization
	096	MOVE	+	9	000	050	199	(7) initially 0	
	097	SET B	+	0	050	099	401	Set address of first x_v	
	098	SET A	+	0	500	109	402	Set address of first x_{v+1}	
	099	MOVE	+	9	000	[ ]	201	Call out x_v for calculation	
2	100	MPY	+	3	000	052	300	$2x_v$	Inner loop init.
	101	MPY	—	3	300	201	000	$2x_v \cdot T_1(x_v)$	Inner loop exec.
	102	SUB	—	2	000	200	202	$T_{n+1} = 2x_v T_n - T_{n-1}$	
	103	LOOP BC	+	0	011	009	101	$n_2 = 1, 2, \ldots, 9$	Inner mod. & test
	104	MPY	+	3	000	000	301	T_{10}	
3	105	MPY	+	3	201	201	000	x_v	Outer loop exec.
	106	SUB	+	2	051	000	000	$1 - x_v^2$	
	107	SQRT	+	0	300	000	000	$\sqrt{1-x_v^2}$	
	108	DIV	+	4	301	000	302	$T_{10}^2/\sqrt{1-x_v^2}$	
	109	SUB	+	2	[ ]	201	000	$x_{v+1} - x_v$	
4	110	MPY	+	3	000	302	000		
	111	ADD	+	1	000	199	199	Partial sum of (7)	
	112	CONS	+	0	401	000	000	Should partial sum be punched?	
	113	TRSGN	+	0	201	115	114		
	114	PCH	+	0	410	199	201	Punch partial sum with x_v (and T_o)	
5	115	PCH	+	0	410	201	210	Punch $T_1 = x_v$, T_2, ..., T_{10}	
	116	ADD B	+	0	060	099	001	Increase v by 1	Outer mod.
	117	ADD A	+	0	600	109	001		
	118	TR B	+	7	099	451	099	Is v = 50?	Outer test
	119	PCH	+	0	410	199	199	Punch (7) unconditionally	End
6	120	UNC STOP	+	0	000	201	000	End; Display x_{50}	

A number of remarks are called for, many of them of general applicability:

(a) The C-address of instruction 101 will, during execution, run through the values 000-008, but the result of the instruction is always called out from 000 on step 102. This trick makes it possible to use the LOOP BC operation instead of address change, which is normally required if different sets of addresses are to be modified during a loop.

(b) The instruction numbering was arrived at by starting the preparation of the program at instruction 101 with the intention of later adding an unknown number of initialization steps preceding it. This speaks in favor of not starting a program at 001.

(c) The stop which would normally occur each time the CONS instruction is reached may be bypassed when found superfluous without any sacrifice by turning the PROGRAMMED STOP switch to RUN, since no COND STOP, SIN or LOG operations (the only other ones involving a conditional stop) are used. The operator decision regarding punching of partial sums is made using only the sign switch of the STORAGE ENTRY switches. This switch does not influence LOADING.

(d) The quantity x_v is used so frequently that it was more economical to MOVE it into a fixed location than to apply address modification. The converse applies to x_{v+1}, which is used only once.

(e) The constant 1 appears in three locations merely in order to simplify bookkeeping and loading, as well as changing the number of points x_v in a later run.

(f) An invaluable aid in determining whether the results of a calculation are correct is a mathematical identity which they must satisfy, and the programming of such checks is strongly recommended whenever it is possible. In the present problem, the identity

$$\text{(iii)} \quad \int_{-1}^{1} \frac{\left[T_{10}(x)\right]^2}{\sqrt{1-x^2}}\, dx \equiv \frac{\pi}{2}$$

is closely connected with the computation of (7) if the x_v are distributed over the interval (-1, 1).

(g) An alternative method of programming the outer loop, which would eliminate the address change operations at the expense of somewhat increased card preparation, would be to key punch the x_v one to a card and give a READ order entering one x_v at a time into a fixed location during the execution of the program. The difficulty arising from the need for x_{v+1} on step 109 is not insurmountable.

XI INTERNAL STRUCTURE OF THE SYSTEM

XI. 1. DETAILED DESIGN CONSIDERATIONS

An expert examining the program at the end of this report will ask a number of questions about apparent duplication, about tight optimization in one routine in contrast to a lack of it in another, about the choice of operations and of methods of implementing them, etc. This section will attempt to answer some of these questions and also suggest a number of changes and additions that could be considered for a second version of the system. Additional questions and suggestions from readers will be genuinely appreciated.

In the early stages of system design, the following requirements were among those agreed upon, in addition to the general principles discussed in Sec. I. 2:

(a) The arithmetic operations and those logical operations most likely to occur in inner loops (LOOP and certain TRANSFER operations) must be as fast as we know how to make them, regardless of the expense in storage.

(b) The system must occupy at most 1000 memory locations.

(c) The special functions must have full accuracy and unlimited range and most of them should be as fast as these requirements and available storage permit.

(d) Optimum programming, (see the 650 Manual of June, 1955) in addition to being necessary for the attainment of (a) and (c), should be used locally in any program where the gain is significant but not at the expense of extensive rewriting of previously completed programs.

(e) The programs must be so written that if the machine stops on any program step in a subroutine and control is transferred elsewhere before restarting, the subroutine, where the stop occurred, is left in a condition which assures correct operation the next time that subroutine is used. This implies that if a subroutine is used in more than one program, it must be initialized by each program rather than having a normal form used in one program and temporarily being changed at the beginning of other programs when needed there and then restored to normal at the end.

(f) To facilitate changes, the individual programs (or "decks", 1-20, see Sec. XI. 7) that make up the system should be as independent of one another as they can be without excessive waste of storage. This requirement was not fully adhered to near the end of the programming task.

As a result of these requirements and of some oversights in programming, there are a number of storage registers which could be made available without any loss in system performance and a number which could be freed at some sacrifice.

A brief guide for finding such registers will now be given followed by a number of suggestions for their possible use in a revised version of the system.

The 6 vacant 0_2-code locations and the 11 vacant registers listed in deck 5 are, of course, available. The only distinction between them is one of mnemonics in connection with the choice of operation codes. In addition, it appears possible to salvage 22 registers essentially without loss by the following substitutions, but a careful check followed by machine testing is advisable:

Deck	Card	Loc.	Replace by
20	113	1801	1848
16	48	1240	1138
18	103	1896	1138
6	70	1360	1160
16	6	1230	1338
18	77	1887	1137
2	32	1058	1955
12	60	1639	1289
16	30	1244	1245
19	103	1702	1103
17	63	1480	1980
10	36	1331	1358
8	32	1166	1241
12	56	1423	1674
17	22	1485	1285
18	97	1842	1504
17	29	1495	1297
5	11	1252	1952
5	12	1255	1955
5	13	1260	1960
5	21	1277	1977
5	23	1283	1983

It is, of course, necessary to determine, by sorting on instruction and data address, all places where the locations listed are referred to.

Registers that may be freed at a price in speed include, above all, nearly 40 extra registers used in the arithematic routines in calling out A and B, splitting them up and storing the parts. This is done separately in each of decks 12, 13 and 14 to accommodate minor differences that facilitate optimization. To combine these steps without any loss of time is a task which, if possible, would require re-optimization of a substantial part of the system. At the expense of one revolution, they may be combined easily. Similarly, making the dissection of $\bar{B}$ common to all 0_2-routines would result in a substantial saving at the expense of lost time in cases (such as LOOP and TREXP) where $\bar{B}$ is irrelevant. To make this dissection common only to those routines where it is needed would be less profitable.

At some sacrifice in external characteristics, registers may, of course, be freed in any number of ways. If, in tracing, the modified instruction (redundant but convenient) is omitted, seven steps are eliminated. The MOVE operation for $A \neq 000$ is easily programmed in terms of LOOP BC and MOVE 000 and could be omitted, as could the special functions in degrees and to base 10 (or radians and base e, respectively).

A number of suggestions for changes and additional operations will now be listed. Suggestions (1) - (3) use only the vacant registers and operation codes listed in deck 5 and can consequently be added to the system without difficulty at the option of any installation or individual programmer. For temporary use, they may be punched on separate cards and loaded after the system deck, in the case of (1) and (2) and after the punching deck, in the case of (3). Such cards should have an x-punch in column 5 and the four-digit location in columns 6 - 9.

(1) Add an 0_2-operation defined as follows:

0_2 = 453	SWITCH	Transfer on switch. Control is transferred to C if the Storage Entry Sign Switch is set to minus, to B if it is set to plus.

This operation bypasses the stop that would occur if the same function were programmed by a CONS and a TR SGN order. It might be particularly useful in connection with tracing when it is desired to start tracing after a certain amount of running time has elapsed or for following the progress of a calculation by occasional punching of intermediate results at the discretion of the operator.

The coding for SWITCH consists of the instruction:

1453	10	8000	1015	Read console. Go to TR SGN routine.

The execution time is 19.2 ms.

(2) Add an 0_2-operation called COUNT having the same counting and testing properties as the LOOP orders but using a counter independent of the loop box and not capable of modifying instructions. Its function can be duplicated, e.g., by a SET A, an ADD A and a TR A instruction. Its advantage lies in its speed and simplicity. The execution time is 24.0 ms. when control is transferred to C and 33.6 the last time when control proceeds ahead and the counter is reset. A formal definition follows:

0_2 = 800	COUNT	The number standing in the counter is increased by 1. Its new value is compared to B. If B is greater, control is transferred to C. Other-

wise, the counter is reset to zero and control proceeds to the next instruction. The counter is also reset in loading.

If COUNT is used extensively, an expansion of the tracing program to punch out the contents of the counter, e.g., in place of the problem number, would seem desirable.

The coding for COUNT reads as follows:

Loc.	Op.	Data	Instr.	Remarks
1800	10	1356	1314	Call out and increase
1314	10	1317	1323	the contents, N, of the counter.
1323	11	8002	1381	Test N+ 1− B.
1381	46	1337	1391	
1337	10	8001	1396	On −, store N + 1 in the counter; go to C
1396	21	1356	1120	(in TR SGN program).
1391	16	8002	1066	On +, reset the counter, go to General
1066	20	1356	1095	Interpretation.
1356	00	0000	0000	The Counter
1317	00	1000	0000	Constant
1194	20	0000	1378	Change in LOADING to reset
1378	24	1356	1178	the counter.

Note: If the COUNT program is loaded separately, the card loading zero into 1356 (step 9 in the program) must be included.

(3) Include in the punching program (deck 9) a routine that prevents unwanted numbers from being punched out when the word count is less than the normal maximum. This can be done on a tabulator with sufficient selector capacity (see Sec. XI. 5). In the 650, it requires five locations and increases the execution time of the PCH order by 24. 0 ms.when the word count is less than the normal maximum. The program, which may be punched on three cards, reads as follows:

1949	44	1306	1095
1306	20	1980	1307
1307	20	1981	1308
1308	20	1982	1309
1309	20	1983	1044
1044	20	1984	1973

(4) Make room for the tracing program to be on the drum in parallel with the punch program, replacing the mode-of-operation deck (see LOADING) by an

x-punch on the transfer card or a setting of the storage entry sign switch. An expansion of the loading program (about 7 steps) or of general interpretation would be needed, and the present overlap between tracing and punching is 34 registers.

(5) If MOVE is omitted, except for A = 000, make this an 0_2-operation and use the vacant 0_1-code for NGDIV. Alternatively, add a fast 0_2-operation, "NEG", identical with MOVE 000 except that it changes the sign of $\overline{B}$. If NEG were available, however, it might be used in cases where, by slight reprogramming, a better program using NGMPY could be written.

(6) Increase the number of logical operations, adding to the flexibility of the system and to the confusion of the beginner: Have a register called the "address counter", addressed, e.g., by 000 or by special operations and SET instructions referring to the address counter (as in 701 Speedcoding) where the present SET instructions refer to their own C-address. Have a set of TR A, TR B, TR C orders which automatically increase the address referred to by 1. These would have to be alternative to the address transfer orders in present use unless vacant 0_1-codes are produced.

(7) Make use of addresses now ignored in some operations. For example, in CONS, use the B-address to call out a number $\overline{B}$ for console display when the machine stops. In TR OUT, make B a "return address" similar to that in TR SUBR. In START TR, or a new tracing order supplementing it, let B (or C) designate the number of steps to be traced before an automatic discontinuation of tracing.

(8) Make Program Loading reset the registers below 1000 to zeros, unless told not to by an x-punch on the first card being loaded.

(9) Have a conditional stop, or an operation effecting such a stop, on loss of accuracy in ADD and SUB, analogous to those in SIN and LOG. In many problems, particularly in connection with tests, such loss is legitimate, however, and a stop undesirable.

(10) Replace or supplement the present error stops by the punching of an "error card".

(11) Introduce an operation similar to ST TR ERAS, perhaps replacing it, which will cause the machine to trace the first, second and last repetition in a loop.

(12) Add another LD-STD pair (at no loss in time) to General Interpretation (see cards 26 and 27) making ADD and SUB, as well as MPY and DIV available as internal subroutines.

(13) Cut the execution time of several subroutines, such as, the arc tangent program, by making minor rearrangements, usually involving the expenditure of a few additional registers.

(14) Add an 0_2-operation, SPACE, which causes an x-punch to appear on the next card punched.

(15) Interchange the functions of registers 1002 and 1702, causing the machine to stop sooner if a programmer accidentally attempts to continue upward from instruction 999.

(16) Investigate whether a carry can ever occur on card 78, deck 18. If not, put the registers used on cards 78-82 to better use.

(17) Replace or supplement the arc tan operation by an 0_1-operation, ARG, which gives the argument (angle) of the point whose coordinates are $(\bar{A}, \bar{B})$.

XI. 2. RELATED SYSTEMS

Several systems supplementing the present one suggest themselves: (a) A system of symbolic or regional programming where the machine assigns absolute addresses in connection with loading; (b) A system externally identical with the present one, or very nearly so, operating on complex numbers, probably with real and imaginary parts in 8-2 floating-decimal form; (c) A system externally identical with the present one, or very nearly so, operating on double precision floating-decimal numbers, e.g., 16-4; (d) A system of "formula translation" or "automatic coding" (such as, the IBM Fortran for the 704) putting on the machine as much as possible of the burden of translation from a set of mathematical formulas to a program.

XI. 3. NUMERICAL METHODS

The study of numerical methods for calculation of the special functions included in the system was not nearly as exhaustive as would have been desirable and no claim to an optimal choice is made.

The square root is computed by Newton's iteration method,

(1) $$x_{n+1} = \frac{1}{2}\left(\frac{B_1}{x_n} + x_n\right)$$

where $1 \leq B_1 \leq 10$, using the initial approximation

(2) $$x_0 = 1 + .22\, B_1 \ .$$

The evaluation of the trigonometric and exponential functions is based on RAND approximations (see Approximations for Digital Computers by Cecil Hastings, Jr., Princeton University Press) to $\sin \frac{\pi}{2} x$ and 10^x for $0 \leq x \leq 1$. Resembling the approximations obtainable by expansion in terms of orthogonal polynomials, these approximations are in general somewhat more efficient than partial sums of Taylor series for a prescribed interval and accuracy, but it is not obvious that a further reduction of the argument followed by the use of a Taylor expansion could not have been better in the present case. For small x, as stated previously, the formula sin x = x (in radians) is used in order to retain significant figures.

The logarithm and arc tangent are evaluated from fixed-length partial sums of power series after preliminary reductions of the argument, since eight-digit RAND approximations were not available. For the logarithm of B_1, $1 \leq B_1 < 10$, the substitutions

$$(3) \quad u = \frac{B_1}{\sqrt{e}}, \; V = \frac{1}{2}, \; \text{for } B_1 < e,$$

$$(4) \quad u = \frac{B_1}{e^{1.65}}, \; V = 1.65, \; \text{for } B_1 \geq e,$$

$$(5) \quad t = \frac{u-1}{u+1},$$

are followed by the evaluation of

$$(6) \quad \log_e B_1 = \log_e \frac{1+t}{1-t} + V = 2t\left[1 + \frac{t^2}{3} + \cdots + \frac{t^{10}}{11}\right] + V.$$

The constants, $\sqrt{e}$ and $e^{1.65}$ were arbitrarily chosen within the intervals that would lead to a minimal number of terms in (6). For x near 1, the logarithm is inherently less accurate than x since

$$(7) \quad d \log x = \frac{dx}{x} \approx dx$$

and $\log x \approx 0$ whereas $x \approx 1$. No substitution comparable to sin x = x can alleviate this difficulty. The use of a second 0_2-operation for $\log(1 + \bar{B})$ was considered but was rejected due to space limitations. This second logarithm could not be used to replace the present one for all values of the argument, since if the logarithm of a small number, say 10^{-10} is desired, the substitution,

$$(8) \quad 1 + \bar{B} = 10^{-10}$$

forced upon the programmer, yields $\bar{B} = -1$ exactly (in the eight-digit system used) with all digits of the input lost.

For the arc tangent, the reduction (after the argument is restricted to $0 \leq \bar{B} \leq 1$ by the obvious properties of the function) is based on the formula

$$\text{(9)} \quad \arc \tan x = \arc \tan y + \arc \tan \frac{x-y}{1+xy}$$

which is merely the addition theorem for the tangent rewritten. With $y = .6$, the use of (9) gives the desired accuracy in

$$\text{(10)} \quad \arc \tan z = z\left[1 - \frac{z^2}{3} + \frac{z^4}{5} - \cdots - \frac{z^{10}}{11}\right]$$

with $z = \frac{x-y}{1+xy}$ for $x > .28$ and $z = x$ for $x \leq .28$. There is again some leeway in the choice of these constants. For small x, the substitution arc tan x = x is used to preserve significant digits.

XI. 4. CONTROL PANEL WIRING FOR THE READ-PUNCH UNIT

The control panel for the 533 Read-Punch Unit associated with the 650 is wired as follows:

Col. 1, 1st Reading, to LOAD.
R + Sign, jackplugged.
P + Sign, jackplugged.

Col. 5, 1st Reading, to Pilot Sel. 1 X PU.
Rd. Hold to PS1 Hold.
Read Card C, Col. 6, to PS1 T.
Read Impulse 0 to PS1 N.
PS1 C to Storage Entry C, Word 1, pos. 3 (from the left).

Read Card C, Col. to Storage Entry C.

7-9	Wd. 1, pos. 4-6
10	Wd. 2, pos. 6
11	Wd. 3, Sign
12-21	Wd. 3, pos. 1-10
22	Wd. 4, Sign
----	-----
67-76	Wd. 8, pos. 1-10
77-79	Wd. 9, pos. 4-6

Read Impulse 12	Wds. 1, 2, 9, 10, Sign
Read Impulse 0	Wd. 1, pos. 7-10 Wd. 2, pos. 7-10 Wd. 9, pos. 7-10

Word Size Emitter	to	Word Size Entry C
10		Wds. 3-8, 10
8		Wd. 1
7		Wd. 9
5		Wd. 2

Storage Exit C	to	Punch Card C, Col.
Wd. 10, pos. 3-6		1-4
Wd. 1, pos. 3-6		6-9
Wd. 2, pos. 6		10
Wd. 3, Sign		11
Wd. 3, pos. 1-10		12-21
Wd. 4, Sign		22
------		---
Wd. 8, pos. 1-10		67-76
Wd. 9, pos. 4-6		77-79
Wd. 2, pos. 10		80

Double Punch and Blank Column Detection as available and desired.

XI. 5. CONTROL PANEL WIRING FOR THE TABULATOR

The IBM accounting machine or tabulator used for printing from the cards associated with the interpretive system may be expected to perform some or all of the following tasks:

(1) Automatic selection of different printing forms (i. e., zero control and spacing between items on a line) for data cards and trace cards.

(2) Selective spacing between lines.

(3) Suppression of unwanted words from cards with word count less than 5 (or 6).

Since there are many tabulator models, it is not feasible to provide a detailed wiring diagram in this report. Instead, suggestions of general applicability will be given.

Exact selector requirements depend on the characteristics of each machine. As an example, requirements on a 416 will be given: The printing of signs requires 6 single-position selectors with X-pickup and 6 positions of 11-12 separation (either special attachments or 6 positions of a selector transferred by an 11-1/2 impulse). Task (1) requires a digit selector (which may be put to duplicate use in task (3)) and 34 selector positions with digit pickup (delayed pickup). Task (2) requires only 3 one-position selectors on the 416. Task (3) requires 55 selector positions with delayed pickup and some comparing units or a five-position selector and a digit emitter for control.

The problem in connection with task (1) is to get the desired zero control and spacing in the two cases with the same setting of the hammersplit levers (also called zero suppression levers) and hammerlocks, on machines where zero control is not performed on the control panel. On a tabulator with 89 type bars, this may be done as follows:

Type bar	Direct Wiring	Data Cards	Trace Cards
Alpha. 1-4	Col. 1-4, II*.		
5	---		
6-8	Col. 7-9, II.		
9	---		
10		Col. 5, II.	Col. 11, I*.
11		Col. 6, II.	Col. 12, II.
12		--	Emit 10.
13		--	Col. 13, II.
14		--	Col. 14, II.
15		Col. 11, I.	Col. 15, II.
16			Emit 10.
17			Col. 16, II.
18			Col. 17, II.
19		--	Col. 18, II.
20		Col. 12-21, II.	Emit 10.
21			Col. 19, II.
22			Col. 20, II.
23			Col. 21, II.
24			Emit 10.
25			Col. 23, II.
26			Emit 10.
27		Col. 22, I.	Col. 24, II.
28			Col. 25, II.
29			Col. 26, II.
30		Col. 23-29, II.	Emit 10.
31			Col. 27, II.

Type bar	Direct Wiring	Data Cards	Trace Cards
32			Col. 28, II.
33			Col. 29, II.
34			Emit 10.
35	Col. 30, II.		
36	Col. 31, II.		
37	Col. 32, II.		
38		---	Emit 10.
39		Col. 33, I.	Col. 35, II.
40			Col. 36, II.
41			Col. 37, II.
42			Emit 10.
43		Col. 34-40, II.	Col. 38, II.
44			Col. 39, II.
Num. 1			Col. 40, II.
2			Emit 10.
3	Col. 41, II.		
4	Col. 42, II.		
5	Col. 43, II.		
6	---		
7	Col. 44, I.		
8-17	Col. 45-54, II.		
18	---		
19	Col. 55, I.		
20-29	Col. 56-65, II.		
30	---		
31	Col. 66, I.		
32-41	Col. 67-76, II.		
42	---		
43-45	Col. 77-79, II.		

(*) The symbol "II" denotes wiring from the second brushes (on some machines called "third reading") whereas "I" denotes wiring from the first brushes ("second reading") through a selector that separates 11's from 12's to the X-PU of an X-distributor ("pilot selector") through the transfer point of which an emitted 10 goes to the type bar in question.

The hammersplit (zero suppression) levers alpha. 4, 11, 25, 37 and num. 5, 17, 29, 41 and the long hammerlocks alpha. 12, 16, 20, 24, 26, 30, 34, 38, 42 and num. 2 are raised. Left zero carry clips of width 3 are attached to hammersplit levers alpha. 6-8 and num. 43-45.

Trace cards are distinguished by the presence of an 8 in column 80. This impulse is wired through a digit selector to the digit pickup of a row of selectors ("class selectors" or "co-selectors with a controlling pilot selector") with a total of 34 sets of points and to the hammerlock control hub.

For performing task (2) in the manner specified in Sec. V. 2., four external switches are needed. On a 416, the single-double spacing lever and the minor, intermediate and major control switches can be used; on machines with four pluggable switches there is no problem.

Spacing every 10 lines (alternative (b), Sec. V. 2.) may be accomplished by adding 1 to a counter on card cycles and using the carry (which, on a 416, is automatically available at the counter total exit) to initiate a minor cycle during which spacing takes place and the counter is cleared.

Spacing alternatives (c) and (d) both involve inspecting a card column at two reading stations and taking certain action when the digit at the first brushes is less than the digit at the second brushes. This may be done by wiring from second brushes to a comparing entry and from the corresponding comparing exit to the immediate pickup ("ZFS P. U." on a 416) of a selector through the transfer point of which the digit at first brushes is passed. In case (d), this digit is taken directly to cause spacing; in case (c) it is wired to the digit pickup of a selector which initiates spacing on the next cycle.

Task (3) is easily accomplished if a sufficient number of selectors are available. Since the same task can be performed on the 650 at the expense of 5 locations, (see Sec. XI. 1.) the tabulator wiring will not be discussed here.

XI. 6. SYSTEM LOADING

The interpretive system deck, normally with 6 words to a card and an x-punch in 5 to get the 1 in column 6 picked up as a leading digit of the address, is loaded by a deck of six self-loading cards (12 in col. 1) with 7 words to a card and a card number in the eighth word. The first card serves the sole purpose of making a fixed console setting possible. The System Loading program on these cards operates as follows:

8000	70	1951	1333	
1951	70	0004	0152	
0004	70	0053	0152	Read in the loading program from six load cards (B into 0001, C + 1 into 0002).
0053	70	0106	0152	
0106	70	0153	0152	
0153	70	0204	0152	
0204	70	0251	0056	Read a system card (non-load).
0056	60	0001	0055	
0055	11	0251	0155	Go to stop if expected loc. B' $\neq$ loc. on card, L.
0155	44	0152	0102	
0102	60	0002	0007	

0007	11	0001	0205	Go to stop if C + 1 < B' + n.
0205	11	0252	0057	
0057	46	0152	0051	
0051	21	0080	0052	
0052	60	0006	0103	Prepare accumulator for move.
0103	10	0251	0156	
0156	15	0202	0206	
0206	10	0252	0054	Store test constant.
0054	21	0070	0203	
0203	11	0252	8002	
8002	69	[0253]	8003	Move one word.
8003	24	[L]	0151	
0151	15	0154	0201	Increase addresses by 1.
0201	10	8001	0207	
0207	11	0070	0101	Test for end of moving.
0101	44	0105	0107	
0105	10	0070	8002	Return to move another word.
0107	60	0080	0104	If C + 1 = B' + n, end of loading; go to console.
0104	44	0157	8000	
0157	60	0001	0005	Increase location by word count, go to read next card.
0005	10	0252	0003	
0003	21	0001	0204	
0152	69	7777	1333	Error stop.
0001	00	B	0000	Constants
0002	00	C 1	0000	
0006	24	0000	0151	
0202	69	0253	8003	
0154	00	0001	0000	

A similar program is used for punching out the system in condensed form in case extensive changes, entered on self-loading, single-instruction cards, have been made.

The Reset Memory Card, mentioned in Sec. V.3., is a load card with eight words. The program, essentially identical with one supplied by the IBM 650 Sales Research Group at Endicott, runs as follows:

8000	70	1951	1333
1951	69	8000	1953
1953	24	0000	1954
1954	69	1957	1955
1955	24	0999	1956
1956	61	1958	8003
8003	20	[0001]	0999

0999	{11	1952	8003
	00	0000	0000
0000	70	1951	1333
1952	00	0001	0000
1957	11	1952	8003
1958	20	0001	0999

XI. 7. PROGRAMS

The complete programs of the system are listed on the next 21 pages, (i) - (xxi). In many cases, but not always, a constant used in two programs is listed in both.

650 INTERPRETIVE SYSTEM.

DECK	CARD	LOC.	OP.	DATA	INSTR.	
						1. GENERAL INTERPRETATION.
1	1	1095	60	1098	1014	
1	2	1014	10	1024	1729	Increase i to i+1.
1	3	1043	21	1098	8001	(see cards 26 & 27 below)
1	4	8001	60	[i+1]	1107	Start dissection system instr.
1	5	1107	46	1112	1061	Shall the loop box be used?
1	6	1061	30	0003	1019	
1	7	1019	20	1023	1026	Store C.
1	8	1026	60	8003	1033	
1	9	1033	30	0003	1041	
1	10	1041	20	1045	1048	Store B.
1	11	1048	60	8003	1105	
1	12	1105	30	0003	1063	
1	13	1063	44	1067	1076	Is an O_2-operation called for?
1	14	1067	10	1020	8003	
1	15	8003	69	8002	1081	Go to an O_1-subroutine.
1	16	1112	11	1017	1046	Add the loop box to the instr.
1	17	1046	61	8003	1061	Remove the minus sign.
1	18	1076	60	8001	1034	
1	19	1034	10	1037	8003	
1	20	8003	65	1045	[1000+O_2]	Go to an O_2-subroutine.
1	21	1098	60	[i]	1107	Constants
1	22	1024		0001		
1	23	1020	69	8002	1081	
1	24	1037	65	1045	1000	
1	25	1017				Loop box (initially 0)
1	26	1729	69	1732	1735	Restore the multiplication
1	27	1735	24	1539	1043	routine (O_1=3) to normal → # 1.03 page i
						(The trig. program makes
1	28	1732	35	0002	1445	special use of it).
						2. TRANSFER OPERATIONS.
2	1	1000	30	0003	1119	O_2=000, UNC STOP
2	2	1119	15	1030	8002	Call out B for console display
2	3	8002	60	[B]	1096	
2	4	1096	69	9999	1120	Stop.
2	5	1200	30	0003	1059	O_2=200, COND STOP
2	6	1059	15	1162	8002	Call out B for console display
2	7	8002	60	[B]	1016	
2	8	1016	01		1120	Stop; Go to C on PROGRAM START
2	9	1201	10	1009	1015	O_2=201, TR SGN
2	10	1015	46	1120	1069	Test the sign of prev. result
2	11	1120	60	1023	1027	Go to C if −.
2	12	1069	60	8002	1027	Go to B if +.
2	13	1202	60	1009	1013	O_2=202, TR EXP

2	14	1013	30	0002	1021	
2	15	1021	67	8002	1029	Get the exp. of prev. result
2	16	1029	30	0001	1036	
2	17	1036	16	1045	1012	Compare it to B; Go ahead
2	18	1012	46	1095	1120	if exp. $<$ B, to C if exp. $\geq$ B.
2	19	1203	60	1023	1027	O_2=203, TR
2	20	1027	30	0003	1035	Replace i+1 by C in
2	21	1035	10	1038	1043	general interpretation.
2	22	1204	65	1023	1127	O_2=204, TR SUBR
2	23	1127	30	0003	1040	
2	24	1040	69	1145	1151	
2	25	1151	22	1058	1161	
2	26	1161	15	1114	8002	Set the C-address of the
2	27	8002	67	[C]	1075	instruction at C equal to B;
2	28	1075	30	0003	1133	go to TR to C.
2	29	1133	60	8002	1092	
2	30	1092	15	1045	1049	
2	31	1049	35	0003	1058	
2	32	1058	21	[C]	1203	
2	33	1205	65	1023	1128	O_2=205, TR OUT
2	34	1128	35	0003	8003	Go to C in machine language
2	35	8003			[C]	
2	36	1454			1095	O_2=454, NOOP
						Go to general interpretation.
2	37	1114	67		1075	
2	38	1145	21		1203	
2	39	1038	60		1107	Constants
2	40	1162	60		1016	
2	41	1030	60		1096	
						3. LOOP OPERATIONS
3	1	1001	30	0005	1064	O_2=001, LOOP C
3	2	1064	69	1017	1071	Move B to the C-address
3	3	1071	30	0002	1077	position; Call out and
3	4	1077	10	8001	1135	increase the loop box.
3	5	1135	10	1138	1093	
3	6	1093	11	8002	1051	Common LOOP steps
3	7	1051	46	1054	1155	Compare the loop box to B;
3	8	1054	10	8001	1062	if less, loop again (go to C).
3	9	1062	21	1017	1120	
3	10	1155	16	8002	1113	If equal, reset the loop box,
3	11	1113	20	1017	1095	go to general interpretation
3	12	1100	30	0001	1057	O_2=100, LOOP A
3	13	1057	10	1017	1025	
3	14	1025	10	1031	1093	
3	15	1101	30	0001	1008	O_2=101, LOOP AC
3	16	1008	10	1017	1074	
3	17	1074	10	1031	1135	
3	18	1110	69	1017	1121	O_2=110, LOOP AB
3	19	1121	30	0001	1028	

3	20	1028	10	8001	1136	
3	21	1136	10	1039	1093	
3	22	1111	69	1017	1171	O_2=111, LOOP ABC
3	23	1171	30	0001	1078	
3	24	1078	10	8001	1185	
3	25	1185	10	1139	1093	
3	26	1010	30	0004	1343	O_2=010, LOOP B
3	27	1343	10	1047	1003	
3	28	1003	10	1017	1093	
3	29	1011	30	0004	1180	O_2=011, LOOP BC
3	30	1180	10	1138	1343	
3	31	1138			0001	
3	32	1031		0100		
3	33	1047			1000	Constants.
3	34	1039		0100	1000	
3	35	1139		0100	1001	
						4. MOVE
4	1	1090	65	8002	1002	O_1=9, MOVE
4	2	1002	45	1109	1186	Test N (i.e., A); go to special move if N=0
4	3	1109	30	0003	1719	
4	4	1719	16	1024	1329	
4	5	1329	20	1283	1187	Store N-1
4	6	1187	35	0003	1097	
4	7	1097	15	1023	1327	
4	8	1327	44	1315	1132	Go to stop if N-1+C $\geq$ 1000.
4	9	1132	11	8001	1190	
4	10	1190	10	1045	1099	Test for upward or downward move.
4	11	1099	46	1367	1153	
4	12	1153	10	8002	1312	
4	13	1312	30	0003	1322	
4	14	1322	10	1325	1330	Initialize for downward move.
4	15	1330	16	1283	1388	
4	16	1388	15	1142	1197	
4	17	1367	10	1023	1328	
4	18	1328	66	8002	1340	
4	19	1340	11	1045	1363	Initialize for upward move.
4	20	1363	30	0003	1172	
4	21	1172	11	1325	1336	
4	22	1336	16	1142	1197	
4	23	1197	21	1255	1176	Store test constant * (see next p.)
4	24	1176	11	1283	8003	Complete initialization
4	25	8003	69	[**]	8002	
4	26	8002	24	[***]	1102	Move one word
4	27	1102	11	1255	1159	
4	28	1159	44	1366	1095	Test for completion
4	29	1366	10	8001	1321	
4	30	1321	10	1024	1179	Change addresses by 1,
4	31	1179	15	8001	8003	return to move next word
4	32	1186	65	1045	1052	If N=0, Get $\bar{B}$ and go
4	33	1052	30	0003	1362	to storing routine.

4	34	1362	15	1065	8002	
4	35	8002	60	[B]	1445	
4	36	1315	69	2222	1095	Error stop for MOVE, READ & PCH.
4	37	1325	69		8002	Constants
4	38	1142	24		1102	
4	39	1065	60		1445	

5. ERASABLE AND VACANT LOCATIONS.

5	1	1306	77	7777	7777	Vacant O_2-codes
5	2	1307	77	7777	7777	
5	3	1308	77	7777	7777	
5	4	1309	77	7777	7777	
5	5	1453	77	7777	7777	
5	6	1800	77	7777	7777	
5	7	1009	55	5555	5555	Inter-subroutine storage
5	8	1023	55	5555	5555	
5	9	1045	55	5555	5555	
5	10	1250	88	8888	8888	Erasable
5	11	1252	88	8888	8888	
5	12	1255	88	8888	8888	
5	13	1260	88	8888	8888	
5	14	1264	88	8888	8888	
5	15	1265	88	8888	8888	
5	16	1267	88	8888	8888	
5	17	1268	88	8888	8888	
5	18	1270	88	8888	8888	
5	19	1272	88	8888	8888	
5	20	1274	88	8888	8888	
5	21	1277	88	8888	8888	
5	22	1278	88	8888	8888	
5	23	1283	88	8888	8888	
5	24	1285	88	8888	8888	
5	25	1289	88	8888	8888	
5	26	1291	88	8888	8888	
5	27	1293	88	8888	8888	
5	28	1294	88	8888	8888	
5	29	1297	88	8888	8888	
5	30	1356	77	7777	7777	Vacant registers
5	31	1314	77	7777	7777	
5	32	1317	77	7777	7777	
5	33	1323	77	7777	7777	
5	34	1378	77	7777	7777	
5	35	1381	77	7777	7777	
5	36	1337	77	7777	7777	
5	37	1391	77	7777	7777	
5	38	1396	77	7777	7777	
5	39	1066	77	7777	7777	
5	40	1044	77	7777	7777	

INITIALIZATION CONSTANTS FOR MOVE

	Downward	Upward
*	B+N-1	-B
**	B	-(B+N-1)
***	C	-(C+N-1)

6. ADDRESS CHANGE OPERATIONS.

6	1	1087	69	1140	1195	O_1=6, TR A
6	2	1088	69	1341	1195	O_1=7, TR B

6	3	1089	69	1342	1195	O_1=8, TR C
						Common steps.
6	4	1195	24	1198	1053	Set amount of shift for TR A, B or C.
6	5	1053	65	8002	1311	
6	6	1311	30	0003	1369	Get the instruction ($\bar{A}$)
6	7	1369	15	1072	8002	located at A.
6	8	8002	67	[A]	1198	
6	9	1198	35	[1,4,7]	1359	
6	10	1359	65	8002	1117	Separate out its A-, B-,
6	11	1117	35	0003	1125	or C-address.
6	12	1125	60	8003	1183	
6	13	1183	30	0003	1392	
6	14	1392	16	1045	1199	Compare this to B.
6	15	1199	45	1203	1095	Go on if equal, to C if unequal.
6	16	1140	35	0001	1359	
6	17	1341	35	0004	1359	Constants
6	18	1342	35	0007	1359	
6	19	1072	67		1198	
6	20	1005	65	8003	1163	O_2=005, SET C
6	21	1050	65	1103	1163	O_2=050, SET B
6	22	1500	65	1104	1163	O_2=500, SET A
6	23	1163	69	1116	1269	Common steps
6	24	1269	22	1274	1177	
6	25	1177	69	1080	1184	Set amounts of shift.
6	26	1184	22	1289	1292	
6	27	1292	65	1045	1349	
6	28	1349	30	0003	1157	
6	29	1157	69	1160	1164	Set the address for storing
6	30	1164	22	1267	1170	the modified instr. at B.
6	31	1170	15	1073	8002	Get the instruction ($\bar{B}$)
6	32	8002	60	[B]	1289	located at B.
6	33	1289	30	[0,3,6]	1108	
6	34	1108	20	1265	1122	Store the right end of $\bar{B}$.
6	35	1122	60	8003	1032	
6	36	1032	30	0003	1144	Destroy the old A-, B- or
6	37	1144	60	8003	1158	C-address
6	38	1158	46	1115	1118	
6	39	1115	16	1023	1130	Replace it by C
6	40	1118	15	1023	1130	
6	41	1130	35	0003	1196	
6	42	1196	15	1265	1274	
6	43	1274	35	[0,3,6]	1267	Attach the right end;
6	44	1267	21	[B]	1095	store at B.
6	45	1103		0003		
6	46	1116	35		1267	
6	47	1080	30		1108	Constants
6	48	1160	21		1095	
6	49	1104		0006		
6	50	1006	65	1023	1079	O_2=006, ADD C
6	51	1007	66	1023	1079	O_2=007, SUB C

6	52	1079	30	0007	1004	Common to both
6	53	1060	65	1023	1129	O_2=060, ADD B
6	54	1070	66	1023	1129	O_2=070, SUB B
6	55	1129	30	0004	1004	Common to both
6	56	1600	65	1023	1189	O_2=600, ADD A
6	57	1700	66	1023	1189	O_2=700, SUB A
6	58	1189	30	0001	1004	Common to both
6	59	1004	20	1260	1068	Common to all ADD & SUB op.
6	60	1068	65	1045	1399	Store shifted ±C;
6	61	1399	30	0003	1357	Set address for storing
6	62	1357	69	1360	1313	modified instr. at B
6	63	1313	22	1267	1370	
6	64	1370	15	1123	8002	Get $\bar{B}$
6	65	8002	60	[B]	1348	
6	66	1348	46	1156	1106	
6	67	1106	10	1260	1267	Add ±C to $\lvert\bar{B}\rvert$; store
6	68	1156	11	1260	1267	(with original sign) at B.
6	69	1267	21	[B]	1095	
6	70	1360	21		1095	
6	71	1123	60		1348	Constants
6	72	1073	60		1289	
						7. PROGRAM LOADING.
7	1	8000	70	1951	1333	Read the first program card.
7	2	1333	60	1242	1124	Set an instruction for return
7	3	1124	24	1278	1194	from the READ routine
7	4	1194	20		1178	Reset 000
7	5	1178	24	1017	1175	Reset the Loop box
7	6	1175	24	1986	1344	Reset the card counter
7	7	1344	60	1952	1042	Test the word count,n. If ≠ 0,
7	8	1042	45	1364	1377	go to READ to move n words
7	9	1377	69	1959	1376	If n=0, put the problem no.
7	10	1376	24	1985	1397	into the punch band,
7	11	1397	69	1976	1345	decide if tracing is required,
7	12	1345	24	1061	1018	(dep. on contents of 1976*),
7	13	1018	60	1951	1126	store the address of the
7	14	1126	10	1038	1729	first programmer instruction, #1, 26 page
7						go to execute it (gen.int.)
7	15	1038	60		1107	*{30 0003 1019 if not tracing
						{21 1980 1386 if tracing
7	17	1242	70	1951	1344	Constant (return from READ)
						8. READING OPERATIONS
8	1	1401	01		1131	O_2=401, READ CONS
8	2	1131	60	8000	1445	Stop. On start, read console
						switches, go to storing routine.
8	3	1400	69	1318	1174	O_2=400, READ
8	4	1174	24	1278	1486	Set return instructions, go to

8	5	1486	69	1241	1168	Steps common with PCH
8	6	1272	21	1277	1736	Return from common steps.
8	7	1736	70	1951	1326	Read a card.
8	8	1326	11	1951	1365	Go to stop if loc. L on the card
8	9	1365	44	1315	1733	differs from progr. first loc.
8	10	1733	60	1291	1347	(the stop is in MOVE)
8	11	1347	11	1952	1372	Go to stop if B'+n > C+1
8	12	1372	11	1277	1382	(n=word count, B'=current
8	13	1382	46	1315	1740	first location)
8	14	1740	21	1285	1590	
8	15	1590	60	1952	1364	
8	16	1364	10	1324	1686	Calculate and store
8	17	1686	10	1951	1056	test constant
8	18	1056	21	1268	1231	
8	19	1231	15	1188	1094	Complete initialization
8	20	1094	11	1952	8002	
8	21	8002	69	[1953]	8003	Move one word from the read
8	22	8003	24	[L]	1320	band into place
8	23	1320	15	1024	1335	Increase addresses by 1.
8	24	1335	10	8001	1193	
8	25	1193	11	1268	1373	Test for end of moving.
8	26	1373	44	1339	1278	
8	27	1339	10	8001	8002	Return to move next word
8	28	1278	60	1285	1390	
8	29	1390	44	1346	1095	If B'+n = C+1, end of READ
8	30	1346	60	1277	1584	
8	31	1584	10	1952	1166	Add n to B' to get new B',
8	32	1166	21	1277	1736	go to read next card.
8	33	1188	69	1953	8003	
8	34	1318	60	1285	1390	Constants
8	35	1241	21	1277	1736	
8	36	1324	24		1320	

<u>9. PUNCHING.</u>

9	1	1410	69	1964	1168	O_2=410, PCH
9	2	1168	24	1272	1081	Common with READ
9	3	1081	65	1023	1232	
9	4	1232	10	1045	1361	Prepare for testing
9	5	1361	30	0003	1319	and initialization.
9	6	1319	15	1024	1134	
9	7	1134	20	1291	1272	
9	8	1272	21	1977	1989	PCH program only
9	9	1989	11	8002	1998	Go to stop (in MOVE) if
9	10	1998	46	1966	1315	B ≥ C+1.
9	11	1966	60	1969	1973	
9	12	1973	21	1978	1988	Set the word count
9	13	1988	60	1291	1972	Let B'=first loc. not yet
9	14	1972	11	1977	1962	punched. If B'+5 ≤ C+1,
9	15	1962	11	1978	1999	go to punch 5 to a card.
9	16	1999	46	1975	1961	If not, go to further testing.
9	17	1961	60	8001	1968	
9	18	1968	30	0004	1948	
9	19	1948	10	1963	1970	Prepare to move n words
9	20	1970	15	1974	1950	to the punch band

9	21	1950	15	1977	1987	
9	22	1987	10	1990	8002	
9	23	8002	69	[B']	8003	Move one word
9	24	8003	24	[1979]	[1990+n]	
9	25	1991	60	1986	1971	Go to punch
9	26	1992	15	1024	1987	
9	27	1993	15	1024	1987	
9	28	1994	15	1024	1987	Return to move
9	29	1995	15	1024	1987	another word
9	30	1996	15	1024	1987	
9	31	1971	10	1024	1967	Increase the card no.
9	32	1967	21	1986	1997	
9	33	1997	71	1977	1947	Punch a card
9	34	1947	60	1977	1946	
9	35	1946	10	1978	1965	Set the location no.
9	36	1965	21	1977	1966	for the next card
9	37	1975	10	8001	1949	If B'=C+1, punching
9	38	1949	44	1973	1095	is completed.
9	39	1976	30	0003	1019	This const. keeps trace orders
9	40	1964	21	1977	1989	inoperative during non-tracing.
9	41	1969		0005		This const. will be 00 0000 1095
9	42	1990			9999	during tracing (PCH inoperative)
9	43	1963	24	1978	1991	Constants
9	44	1974	69		8003	When the PCH progr. is loaded
9	45	1958	24	1978	1991	with program loading, this instr.
						will load into 1963 (!).
						10. TRACING
10	1	1061	21	1980	1386	Expansion of gen. int.
10	2	1386	30	0003	1946	Store modified instr. for trac.
10	3	1946	20	1023	1947	Store C
10	4	1947	60	8003	1948	
10	5	1948	30	0003	1949	
10	6	1949	20	1045	1950	Store B
10	7	1950	60	8003	1961	
10	8	1961	24	1260	1962	
10	9	1962	30	0003	1963	
10	10	1963	44	1965	1968	
10	11	1965	65	8002	1966	Call out $\bar{A}$ if $O_1 \neq 0$
10	12	1966	30	0003	1967	
10	13	1967	15	1998	8002	
10	14	8002	60	[A]	1968	
10	15	1968	21	1982	1969	Store $\bar{A}$ or 0 for tracing
10	16	1969	65	1045	1970	
10	17	1970	30	0003	1971	
10	18	1971	15	1999	8002	
10	19	8002	69	[B]	1972	Store $\bar{B}$ for tracing
10	20	1972	24	1983	1973	
10	21	1973	69		1974	Store prev. result for tracing
10	22	1974	24	1984	1975	
10	23	1975	69	1017	1987	Store loop box for tracing
10	24	1987	24	1981	1988	
10	25	1988	65	1098	1989	Store instr. no. for tracing
10	26	1989	24	1977	1990	
10	27	1990	15	1997	8002	
10	28	8002	69	[i+1]	1991	Store original instr. for tracing
10	29	1991	24	1979	1992	
10	30	1992	65	1986	1993	

10	31	1993	15	1024	1994	Store card no. for trace cards
10	32	1994	20	1986	1995	
10	33	1995	71	1977	1996	Punch a trace card
10	34	1996	60	1260	1105	Return to normal gen. int.
10	35	1450	69	1976	1331	O_2=450, START TR
10	36	1331	24	1061	1095	Modify gen. int. to include tracing expansion
10	37	1451	69	1154	1358	O_2=451, STOP TR
10	38	1358	24	1061	1095	Restore gen. int. to normal
10	39	1452	69	1976	1332	O_2=452, ST TR ERAS
10	40	1332	24	1061	1091	Modify gen. int. to include
10	41	1091	65	1098	1165	tracing expansion.
10	42	1165	16	1173	1334	
10	43	1334	69	1141	8002	Replace the present progr.
10	44	8002	24	[i+1]	1095	instr. by a NOOP
10	45	1997	09		0884	
10	46	1998	60		1968	
10	47	1999	69		1972	
10	48	1141	04	5400		Constants
10	49	1173	36		0012	
10	50	1154	30	0003	1019	
10	51	1978		0006	0008	Word count & trace identif.
10	52	1964			1095	Make PCH inoperative when tracing.
10	53	1976	21	1980	1386	Will be 30 0003 1019 when not trac.
10	54	1024		0001		When the trace program is
10	55	1958	44	1965	1968	loaded with program loading, this will load into 1963 (!).

11. STORING THE RESULT

11	1	1445	21		1404	(Common to all math. routines)
11	2	1404	21	1009	1414	Store $\bar{C}$ in 000 for prev. res. and
11	3	1414	65	1023	1427	in 1009 for cond. transfer ops.
11	4	1427	45	1430	1095	If C=000, go directly to gen. int.
11	5	1430	30	0003	1439	
11	6	1439	15	1443	1447	If C $\neq$ 000, store $\bar{C}$ in C,
11	7	1447	69		8002	go to gen. int.
11	8	8002	24	[C]	1095	
11	9	1443	24		1095	Constant

12. ADDITION AND SUBTRACTION

12	1	1082	24	1289	1492	O_1=1, ADD
12	2	1492	65	1045	1449	
12	3	1449	30	0003	1457	Get $\bar{B}$
12	4	1457	15	1460	8002	
12	5	8002	60	[B]	1473	
12	6	1083	24	1289	1542	O_1=2, SUB
12	7	1542	65	1045	1599	
12	8	1599	30	0003	1207	Get $-\bar{B}$
12	9	1207	15	1310	8002	
12	10	8002	61	[B]	1473	
12	11	1473	21	1278	1483	Common steps
12	12	1483	65	1289	1493	
12	13	1493	45	1496	1497	If A=000, get $\bar{A}$ directly.

12	14	1496	30	0003	1406	
12	15	1406	15	1409	8002	Get $\bar{A}$ if A $\neq$ 000
12	16	8002	60	[A]	1446	
12	17	1446	21		1405	
12	18	1497	60		1405	
12	19	1405	30	0002	1411	
12	20	1411	20	1265	1468	$\bar{A} = A_1,a$. Split up
12	21	1468	21	1272	1475	and store
12	22	1475	60	1278	1583	
12	23	1583	30	0002	1589	$\bar{B} = B_1,b$. Split up
12	24	1589	20	1293	1696	and store
12	25	1696	21	1250	1603	
12	26	1603	67	8002	1661	
12	27	1661	18	1265	1519	
12	28	1519	30	0004	1630	Put $\lvert b-a \rvert$ into shift instr.
12	29	1630	69	1433	1436	
12	30	1436	22	1639	1442	
12	31	1442	45	1697	1247	
12	32	1247	65	1250	1408	If a=b, form $A_1+B_1=C_1'$.
12	33	1408	15	1272	1429	
12	34	1429	35	0002	1435	
12	35	1435	44	1489	1440	If $\lvert C_1' \rvert \geq 10$, $C_1 = C_1'/10$.
12	36	1489	31	0003	1499	
12	37	1499	46	1652	1703	
12	38	1652	16	1456	1461	If $C_1 < 0$, get c=a+1,
12	39	1461	18	1265	1471	combine with C_1,
12	40	1471	35	0002	1577	go to store.
12	41	1577	10	8002	1445	
12	42	1703	15	1456	1611	Same if $C_1 > 0$
12	43	1611	17	1265	1471	
12	44	1440	45	1494	1445	If $C_1=0$, go to store machine 0.
12	45	1494	60	8002	1403	
12	46	1403	36		1521	
12	47	1521	11	8002	1479	If $\lvert C_1' \rvert < 10$, get C_1 by
12	48	1479	60	8003	1487	shifting; correct exp.
12	49	1487	30	0002	1544	
12	50	1544	46	1461	1611	
12	51	1697	69		1503	If a $\neq$ b, prepare to shift.
12	52	1503	30	0005	1415	
12	53	1415	45	1418	1469	
12	54	1418	46	1422	1423	If $\lvert b-a \rvert \geq 10$, get $\bar{C}=\bar{A}$ or
12	55	1422	60	8001	1445	$\bar{C}=\bar{B}$, go to store.
12	56	1423	60	1278	1445	
12	57	1469	46	1472	1623	
12	58	1472	65	1250	1412	
12	59	1412	69	1272	1639	If $b < a$, shift B_1.
12	60	1639	31	[ ]	1413	
12	61	1413	15	8001	1429	
12	62	1623	69	1293	1501	
12	63	1501	24	1265	1619	If $b > a$, interchange
12	64	1619	65	1272	1432	A_1 and B_1, replace a by b.
12	65	1432	69	1250	1639	
12	66	1310	61		1473	
12	67	1460	60		1473	
12	68	1409	60		1446	Constants
12	69	1433	31		1413	
12	70	1456	01			

13. MULTIPLICATION

13	1	1084	24	1289	1592	$O_1=3$, MPY
13	2	1592	65	1045	1549	
13	3	1549	30	0003	1557	Get $\bar{B}$
13	4	1557	15	1560	8002	
13	5	8002	60	[B]	1578	
13	6	1086	24	1289	1692	$O_1=5$, NGMPY
13	7	1692	65	1045	1649	
13	8	1649	30	0003	1657	Get $-\bar{B}$
13	9	1657	15	1660	8002	
13	10	8002	61	[B]	1578	
13	11	1578	21	1283	1586	Common steps
13	12	1586	65	1289	1593	
13	13	1593	45	1596	1597	
13	14	1596	30	0003	1605	Get $\bar{A}$
13	15	1605	15	1508	8002	
13	16	8002	60	[A]	1655	
13	17	1597	60		1655	
13	18	1655	30	0002	1511	
13	19	1511	44	1515	1316	If $A_1=0$, $\bar{C}=0$ directly.
13	20	1515	21	1270	1523	
13	21	1523	20	1277	1530	$\bar{A}=A_1,a$. Split up and store
13	22	1530	60	1283	1537	
13	23	1537	30	0002	1543	$\bar{B}=B_1,b$. Split up and store.
13	24	1543	44	1547	1445	
13	25	1547	21	1252	1556	
13	26	1556	68	8002	1565	
13	27	1565	15	1568	1573	Calculate and store
13	28	1573	18	1277	1531	49 - (a+b)
13	29	1531	20	1285	1588	
13	30	1588	46	1591	1595	Go to error stop if exp. of
13	31	1591	44	1595	1546	prod. would be out of range
13	32	1546	60	1252	1564	
13	33	1564	19	1270	1648	$C_1' = A_1 \cdot B_1$
13	34	1648	35	0004	1259	
13	35	1259	65	8003	1717	
13	36	1717	35	0001	1273	Common with DIV.
13	37	1273	44	1677	1628	
13	38	1677	69	1580	1533	If $C_1' \geq 10$, $C_1 = C_1'/10$.
13	39	1533	31	0003	1743	
13	40	1628	69	1281	1484	If $C_1' < 10$, prepare to
13	41.	1484	31	0002	1743	correct exponent
13	42	1743	35	0002	1299	
13	43	1299	44	1253	1554	If $C_1=10$ (due to carry in
13	44	1554	46	1507	1258	rounding), go to correct.
13	45	1507	15	8001	1563	
13	46	1258	16	8001	1563	
13	47	1253	35	0007	1528	
13	48	1563	60	8002	1571	Calculate c = a+b-49-corr.;
13	49	1571	30	0002	1528	attach it to C_1.
13	50	1528	46	1581	1532	
13	51	1581	18	1285	1539	
13	52	1532	17	1285	1539	
13	53	1539	35	0002	1445	

13	54	1595	69	5555	1338	Error stop. for MPY and DIV.
13	55	1660	61		1578	Constants
13	56	1560	60		1578	
13	57	1508	60		1655	
13	58	1568	49			
13	59	1580				
13	60	1281			0001	
13	61	1338	65	8002	1445	Go to store 0 after error stop.
13	62	1316	60	8003	1539	
						14. DIVISION.
14	1	1085	24	1289	1642	$O_1=4$, DIV
14	2	1642	65	1045	1699	Get $\overline{B}$
14	3	1699	30	0003	1607	
14	4	1607	15	1610	8002	
14	5	8002	60	[B]	1673	$\overline{B}=0$ tested below
14	6	1626	21	1283	1636	
14	7	1636	65	1289	1643	Get $\overline{A}$
14	8	1643	45	1646	1647	
14	9	1646	30	0003	1656	
14	10	1656	15	1659	8002	
14	11	8002	60	[A]	1606	
14	12	1647	60		1606	
14	13	1606	30	0002	1613	$\overline{A}=A_1$,a. Split up and store
14	14	1613	20	1267	1620	
14	15	1620	21	1274	1527	$A_1=0$ tested below
14	16	1680	60	1283	1637	$\overline{B}=B_1$,b. Split up and store
14	17	1637	30	0002	1693	
14	18	1693	20	1297	1650	
14	19	1650	60	8003	1407	
14	20	1407	35	0002	1663	
14	21	1663	21	1268	1671	
14	22	1671	60	1274	1629	Store shifted A_1.
14	23	1629	35	0001	1685	
14	24	1685	21	1291	1644	
14	25	1644	67	1297	1651	Calculate and store b-a-50.
14	26	1651	16	1654	1510	
14	27	1510	18	1267	1621	
14	28	1621	46	1624	1595	Go to stop (in MPY) if exp. of quotient would be out of range.
14	29	1624	44	1595	1678	
14	30	1678	20	1285	1638	
14	31	1638	60	1291	1645	C¦$=A_1/B_1$; go to exp. corr. (in MPY) and storing routines.
14	32	1645	64	1268	1717	
14	33	1610	60		1673	Constants
14	34	1659	60		1606	
14	35	1654	50			
14	36	1673	45	1626	1374	Go to stop if $\overline{B}=0$.
14	37	1527	45	1680	1445	Set $\overline{C}=0$ directly if $A_1=0$
14	38	1374	69	3333	1338	Error stop for $\overline{B}=0$.
						15. ABSOLUTE VALUE.
15	1	1350	30	0003	1169	$O_2=350$, ABS
15	2	1169	15	1022	8002	Get $\lvert\overline{B}\rvert=\overline{C}$, go to store it.
15	3	8002	67	[B]	1137	
15	4	1137	60	8002	1445	

15	5	1022	67		1137	Constant
						16. SQUARE ROOT
16	1	1300	30	0003	1214	O_2=300, SQRT
16	2	1214	15	1219	8002	
16	3	8002	60	[B]	1206	Get $\bar{B}$
16	4	1206	46	1670	1211	
16	5	1670	69	4444	1230	Stop if $\bar{B} < 0$.
16	6	1230	60	8002	1445	
16	7	1211	30	0002	1218	
16	8	1218	44	1223	1445	If $\bar{B}$=0, $\bar{C}$=0 directly.
16	9	1223	20	1278	1181	
16	10	1181	60	8003	1239	$\bar{B}$=B_1,b. Split up and store
16	11	1239	35	0001	1248	
16	12	1248	21	1255	1229	
16	13	1229	60	1234	1251	
16	14	1251	19	1255	1235	x_0=1+.22B_1
16	15	1235	10	1240	1245	
16	16	1245	30	0001	1152	
16	17	1152	10	1255	1209	
16	18	1209	16	8002	1217	
16	19	1217	24	1270	1227	
16	20	1227	64	8001	1238	Calc. and store B_1/x_n.
16	21	1238	20	1294	1216	
16	22	1216	16	1270	1225	
16	23	1225	30	0001	1182	If $\|B_1/x_n - x_n\| < 10^{-8}$, go to end.
16	24	1182	45	1236	1237	
16	25	1236	65	8001	1191	
16	26	1191	15	1294	1249	
16	27	1249	10	1055	1210	$x_{n+1} = 1/2\ (B_1/x_n + x_n)$
16	28	1210	16	8002	1228	
16	29	1228	19	8001	1244	
16	30	1244	30	0001	1152	
16	31	1237	60	1055	1222	
16	32	1222	19	1278	1208	
16	33	1208	35	0001	1215	Calc. 1/2 (b+50).
16	34	1215	10	1419	1224	
16	35	1224	21	1278	1233	
16	36	1233	60	8002	1192	If b is even, go to end.
16	37	1192	44	1246	1257	
16	38	1246	60	1150	1212	
16	39	1212	19	1270	1220	If b is odd, $C_1 = x_N \sqrt{10}$.
16	40	1220	31		1243	
16	41	1243	60	8002	1213	
16	42	1213	35	0002	1221	Equip with exp., go to store
16	43	1221	10	1278	1445	
16	44	1257	65	1270	1226	If b is even, C_1=x_N.
16	45	1226	31	0002	1243	
16	46	1219	60		1206	
16	47	1234			0022	
16	48	1240			0001	Constants
16	49	1055			0005	
16	50	1419			0025	
16	51	1150	03	1622	7766	

17. EXPONENTIAL

17	1	1301	69	1514	1467	O_2=301, EXP E
17	2	1351	69	1464	1467	O_2=351, EXP 10
17	3	1467	30	0003	1425	Common steps.
17	4	1425	24	1485	1488	Set an instr. for EXP E or EXP 10.
17	5	1488	15	1441	8002	
17	6	8002	60	[B]	1553	Get $\overline{B}$=B_1,b. Split up
17	7	1553	30	0002	1459	and store.
17	8	1459	21	1264	1417	
17	9	1417	46	1420	1421	
17	10	1420	69	1424	1477	Set an instr. distinguishing
17	11	1421	69	1524	1477	between pos. and neg. exp.
17	12	1477	24	1480	1633	
17	13	1633	67	8002	1541	
17	14	1541	16	1444	1550	
17	15	1550	35	0001	1458	Form b-49. Go to special
17	16	1458	44	1462	1512	routines if $\lvert b-49 \rvert \geq 10$.
17	17	1512	30	0005	1525	
17	18	1525	46	1478	1529	
17	19	1478	69	1481	1485	Set instr. for left or right
17	20	1529	69	1482	1485	shift of $\lvert b-49 \rvert$.
17	21	1485	22	1495	1551	Go to EXP E routine
17	22	1485	22	1495	1502	Go to EXP 10 routine
17	23	1551	60	1504	1509	
17	24	1509	19	1264	1631	For $e^{\overline{B}}=10^{m\overline{B}}$, calc. $\lvert mB_1 \rvert$.
17	25	1631	35	0002	1437	
17	26	1437	67	8003	1495	
17	27	1502	67	1264	1669	For $10^{\overline{B}}$, get $\lvert B_1 \rvert$.
17	28	1669	35	0002	1495	
17	29	1495	[ ]	[$\lvert b-49 \rvert$]	1465	Shift $\lvert \overline{B} \rvert = n+\lambda$. ($0 \leq \lambda < 1$).
17	30	1465	20	1270	1474	Store λ.
17	31	1474	21	1278	1681	Store n.
17	32	1681	11	1587	1491	
17	33	1491	46	1594	1393	If $\lvert \overline{B} \rvert \geq 50$, go to stop.
17	34	1594	60	1548	1516	
17	35	1516	19	1270	1695	
17	36	1695	60	8003	1653	
17	37	1653	10	1506	1466	
17	38	1466	19	1270	1694	
17	39	1694	60	8003	1254	
17	40	1254	10	1658	1416	
17	41	1416	19	1270	1448	
17	42	1448	60	8003	1608	
17	43	1608	10	1612	1667	
17	44	1667	19	1270	1598	
17	45	1598	60	8003	1558	$10^{\lambda} = [1+a_1\lambda+\ldots+a_7\lambda^7]^2$
17	46	1558	10	1562	1617	
17	47	1617	19	1270	1552	
17	48	1552	60	8003	1609	
17	49	1609	10	1513	1567	
17	50	1567	19	1270	1402	
17	51	1402	60	8003	1559	
17	52	1559	10	1463	1517	
17	53	1517	19	1270	1602	
17	54	1602	60	8003	1662	
17	55	1662	10	1665	1520	
17	56	1520	19	8003	1480	

17	57	1480	65	8003	1687	
17	58	1687	31	0001	1498	
17	59	1498	35	0002	1566	For $\overline{B} \geq 0$, round 10^{λ},
17	60	1566	60	8002	1625	equip with exponent n+50,
17	61	1625	10	1278	1683	go to store.
17	62	1683	10	1587	1445	
17	63	1480	21	1285	1438	
17	64	1438	60	1665	1522	
17	65	1522	30	0002	1679	For $\overline{B} < 0$, calculate $10^{-\lambda}$.
17	66	1679	64	1285	1470	
17	67	1470	35	0001	1428	
17	68	1428	44	1431	1632	
17	69	1632	31	0002	1641	
17	70	1641	15	1444	1604	Equip $10^{-\lambda}$ with exponent
17	71	1604	35	0002	1616	49-n (if $10^{-\lambda} < 1$) or
17	72	1616	10	8002	1675	50 (if $10^{-\lambda} = 1$); go to store.
17	73	1675	11	1278	1445	
17	74	1431	60	1434	1675	
17	75	1462	46	1615	1393	Go to stop if $b \geq 59$,
17	76	1615	60	1434	1445	set $\overline{C}=1$ if $b \leq 39$.
17	77	1514	22	1495	1551	
17	78	1464	22	1495	1502	
17	79	1441	60		1553	
17	80	1524	65	8003	1687	
17	81	1424	21	1285	1438	
17	82	1444	49			
17	83	1481	30		1465	
17	84	1482	35		1465	
17	85	1504	43	4294	4819	Constants
17	86	1587			0050	
17	87	1548		0093	2643	
17	88	1506		0255	4918	
17	89	1658		1742	1120	
17	90	1612		7295	1737	
17	91	1562	02	5439	3575	
17	92	1513	06	6273	0884	
17	93	1463	11	5129	2776	
17	94	1665	10			
17	95	1434	10		0050	
17	96	1393	69	6666	1338	Error stop for exp., log. & trig.
						18. LOGARITHM.
18	1	1302	69	1807	1818	O_2=302, LOG E
18	2	1352	69	1815	1818	O_2=352, LOG 10
18	3	1818	24	1821	1824	Common steps.
18	4	1824	30	0003	1833	
18	5	1833	15	1836	8002	
18	6	8002	60	[B]	1831	Get $\overline{B}=B_1$,b. Split up,
18	7	1831	30	0002	1837	store exponent
18	8	1837	20	1291	1844	
18	9	1844	44	1847	1393	Go to stop (in EXP) if $B_1 \leq 0$.
18	10	1847	46	1393	1751	
18	11	1751	60	8003	1809	
18	12	1809	11	1812	1817	

18	13	1817	46	1820	1871	If $B_1 < e$, set $u = B_1/\sqrt{e}$, $v=.5$.
18	14	1820	10	8001	1825	
18	15	1825	19	1828	1814	
18	16	1814	69	1867	1870	
18	17	1871	10	8001	1827	If $B_1 \geq e$, set $u = B_1/e^{1.65}$, $v=1.65$
18	18	1827	19	1830	1813	
18	19	1813	69	1816	1870	
18	20	1870	24	1274	1877	
18	21	1877	35	0002	1883	$t = \frac{u-1}{u+1}$
18	22	1883	10	1886	1841	
18	23	1841	21	1297	1802	
18	24	1802	11	1869	1891	
18	25	1891	64	1297	1876	
18	26	1876	60	8002	1885	$L = \log_e \frac{1+t}{1-t} + v = t[2 + \frac{2}{3}t^2 + \frac{2}{5}t^4 + \ldots + \frac{2}{11}t^{10}]+v.$
18	27	1885	24	1289	1892	
18	28	1892	19	8001	1860	
18	29	1860	21	1264	1767	
18	30	1767	60	8001	1823	
18	31	1823	19	1776	1894	
18	32	1894	60	8003	1851	
18	33	1851	10	1856	1861	
18	34	1861	19	1264	1890	
18	35	1890	60	8003	1850	
18	36	1850	10	1855	1859	
18	37	1859	19	1264	1792	
18	38	1792	60	8003	1899	
18	39	1899	10	1853	1858	
18	40	1858	19	1264	1889	
18	41	1889	60	8003	1898	
18	42	1898	10	1852	1857	
18	43	1857	19	1264	1805	
18	44	1805	30	0001	1866	
18	45	1866	10	1869	1873	
18	46	1873	60	8003	1884	
18	47	1884	19	1289	1821	
18	48	1821	10	1274	[ ]	End of common steps
18	49	1829	21	1285	1838	For LOG E, store L, calculate Mb_1. (b_1=b-50)
18	50	1838	60	1291	1845	
18	51	1845	11	1848	1803	
18	52	1803	19	1806	1882	
18	53	1881	60	8003	1839	For LOG 10, store mL, calculate b_1.
18	54	1839	19	1842	1826	
18	55	1826	21	1285	1888	
18	56	1888	60	1291	1895	
18	57	1895	11	1848	1822	
18	58	1822	30	0001	1882	
18	59	1882	69	1285	1840	<u>Common steps</u>
18	60	1840	30	0008	1810	
18	61	1810	15	8001	1819	
18	62	1819	35	0004	1880	
18	63	1880	44	1835	1893	Go to stop on loss of more than two digits of accuracy
18	64	1835	35	0004	1846	
18	65	1846	44	1849	1445	Go to store machine zero if C' has seven zeros.
18	66	1849	36		1862	
18	67	1862	69	1865	1868	
18	68	1868	23	1272	1875	

18	69	1875	65	8003	1834	$C' = \begin{cases} L+Mb_1, \text{ for LOG E} \\ mL+b_1, \text{ for LOG 10} \end{cases}$
18	70	1834	31	0002	1843	
18	71	1843	35	0002	1750	
18	72	1804	46	1808	1811	
18	73	1811	16	1272	1878	Normalize and round.
18	74	1878	15	1832	1887	
18	75	1808	15	1272	1879	
18	76	1879	16	1832	1887	
18	77	1887	60	8002	1445	
18	78	1750	44	1854	1804	
18	79	1854	30	0001	1863	In case a carry occurs
18	81	1874	15	1896	1804	in rounding, correct.*
18	82	1872	16	1896	1804	
18	83	1893	01		1835	Stop on loss of two digits, return to log. progr. on start.
18	84	1807	10	1274	1829	
18	85	1815	10	1274	1881	
18	86	1836	60		1831	
18	87	1812		2718	2818	
18	88	1828	60	6530	6597	
18	89	1867	05			
18	80	1863	46	1872	1874	* Card out of place.
18	90	1816	16	5000		
18	91	1830	19	2049	9086	
18	92	1886	10			
18	93	1869	20			Constants
18	94	1832			0052	
18	95	1848	50			
18	96	1806	23	0258	5093	
18	97	1842	43	4294	4819	
18	98	1776	18	1818	1818	
18	99	1856	22	2222	2222	
18	100	1855	28	5714	2857	
18	101	1853	40			
18	102	1852	66	6666	6666	
18	103	1896			0001	
18	104	1865				

19.. SINE AND COSINE.

19	1	1303	69	1256	1614	O_2=303, SIN R
19	2	1614	24	1267	1721	
19	3	1721	69	1526	1579	
19	4	1304	69	1707	1664	O_2=304, COS R
19	5	1664	24	1267	1622	
19	6	1622	69	1576	1579	
19	7	1579	24	1283	1536	Common steps.
19	8	1536	65	8002	1545	(this step needed in degree prog.)
19	9	1545	30	0003	1455	
19	10	1455	15	1561	8002	
19	11	8002	60	[B]	1574	Get $\overline{B}=B_1$,b. Split
19	12	1574	21	1278	1582	up and store.
19	13	1582	30	0002	1689	
19	14	1689	21	1252	1267	
19	15	1267	[ ]	[ ]	[ ]	

19	16	1575	68	8002	1538	Tests in sine routine:
19	17	1538	15	1691	1295	
19	18	1295	45	1298	1749	Test 47-b.
19	19	1298	46	1569	1674	
19	20	1749	67	1252	1708	
19	21	1708	15	1711	1666	If b=47, test $\lvert\overline{B}\rvert$-.0025
19	22	1666	44	1569	1674	
19	23	1674	60	1278	1445	If $\lvert\overline{B}\rvert < .0025$, go to store $\overline{C}=\overline{B}$
19	24	1569	68	1272	1627	
19	25	1627	15	1444	1601	If $\lvert\overline{B}\rvert \geq .0025$, test 49-b,
19	26	1601	35	0001	1709	go to special routines if
19	27	1709	44	1276	1714	$\lvert 49-b\rvert \geq 10$.
19	28	1714	30	0005	1727	
19	29	1727	46	1682	1282	
19	30	1282	69	1635	1688	For $\lvert 49-b\rvert < 10$, set
19	31	1688	22	1293	1715	shift instruction.
19	32	1682	69	1535	1738	
19	33	1738	22	1293	1746	
19	34	1746	15	1702	1710	If $b > 52$, stop to indicate
19	35	1710	46	1375	1715	loss of accuracy; continue
19	36	1375	01		1715	on start
19	37	1715	60	1618	1426	
19	38	1426	19	1252	1718	
19	39	1718	35	0002	1293	Compute the fractional
19	40	1293	[ ]	[$\lvert 49-b\rvert$]	1716	part F of $\overline{B}/2\pi$, then
19	41	1716	30	0001	1723	$G=F$ if $F \geq 0$, $G=F+1$ if $F < 0$.
19	42	1723	46	1476	1741	
19	43	1476	10	1886	1741	
19	44	1741	61	8003	1704	
19	45	1704	10	8003	1713	
19	46	1713	10	8003	1283	$\alpha = 4G$ for SIN, $1+4G$ for COS.
19	47	1283	[ ]	1886	1742	
19	48	1742	10	8001	1747	If $\alpha \leq 1$, $\alpha'=\alpha$
19	49	1747	46	1705	1706	
19	50	1705	10	8001	1262	If $1 < \alpha \leq 2$, $\alpha'=2-\alpha$
19	51	1262	46	1266	1748	
19	52	1266	10	8001	1271	If $2 < \alpha \leq 3$, $\alpha'=2-\alpha$
19	53	1271	46	1676	1634	
19	54	1676	10	8001	1540	If $3 < \alpha < 5$, $\alpha'=\alpha-4$.
19	55	1706	11	8001	1540	
19	56	1540	61	8003	1748	These angle reductions
19	57	1748	30	0009	1720	carried out.
19	58	1634	11	8001	1748	
19	59	1720	44	1383	1724	If $\alpha'=1$, set $S=1$.
19	60	1724	20	1285	1288	
19	61	1288	60	8001	1744	
19	62	1744	19	8001	1745	
19	63	1745	21	1250	1712	
19	64	1712	60	8003	1668	
19	65	1668	19	1572	1730	
19	66	1730	60	8003	1387	
19	67	1387	11	1490	1296	
19	68	1296	19	1250	1368	$S=C_1\alpha'+C_3\alpha'^3+\ldots+C_9\alpha'^9$.
19	69	1368	60	8003	1728	
19	70	1728	10	1534	1290	
19	71	1290	19	1250	1371	
19	72	1371	60	8003	1280	
19	73	1280	11	1684	1389	

19	74	1389	19	1250	1275	
19	75	1275	30	0001	1734	
19	76	1734	60	8003	1394	
19	77	1394	10	1398	1726	
19	78	1726	19	1285	1640	
19	79	1640	65	8003	1672	
19	80	1672	35	0001	1279	
19	81	1279	44	1383	1284	
19	82	1383	31	0003	1722	
19	83	1722	35	0002	1379	
19	84	1284	31	0002	1395	
19	85	1395	35	0001	1379	Round and normalize,
19	86	1379	60	8002	1737	go to store
19	87	1737	36		1263	
19	88	1263	11	8002	1725	
19	89	1725	46	1286	1287	
19	90	1287	10	1690	1445	
19	91	1286	11	1690	1445	
19	92	1276	46	1393	1380	Go to stop if b $\geq$ 59,
19	93	1380	60	1865	1283	set α=0 if b $\leq$ 39.
19	94	1256	20	1272	1575	
19	95	1526	69	1886	1742	
19	96	1707	68	8002	1627	
19	97	1576	11	1886	1742	
19	98	1561	60		1574	
19	99	1691	47			
19	100	1711	99	7500		
19	101	1635	30		1716	
19	102	1535	35		1716	Constants
19	103	1702		0003		
19	104	1618	15	9154	9430	
19	105	1572		0151	4842	
19	106	1490		4673	7656	
19	107	1534	07	9689	6793	
19	108	1684	64	5963	7111	
19	109	1398	15	7079	6318	
19	110	1690			0050	
19	111	1444	49			
19	112	1353	69	1384	1739	O_2=353, SIN D
19	113	1354	69	1385	1739	O_2=354, COS D
19	114	1739	24	1293	1505	Common steps
19	115	1505	69	1261	1585	Set an instr. at the end of
19	116	1585	24	1539	1698	MPY to get out to SIN R or COS R.
19	117	1698	20	1289	1555	Go to MPY with B in place of A
19	118	1555	60	1167	1578	and $2\pi/360$ in place of B.
19	119	1384	21		1303	
19	120	1385	21		1304	Constants
19	121	1261	35	0002	1293	
19	122	1167	17	4532	9348	

20. ARC TANGENT.

20	1	1305	69	1758	1761	O_2=305, ARC TAN R

20	2	1355	69	1908	1761	<u>O_2=355</u>, <u>ARC</u> <u>TAN</u> <u>D</u>
20	3	1761	24	1264	1917	<u>Common</u> <u>steps</u>
20	4	1917	30	0003	1925	
20	5	1925	15	1928	8002	
20	6	8002	60	[B]	1941	Get B=B_1,b; store B_1.
20	7	1941	21	1297	1900	
20	8	1900	30	0002	1757	
20	9	1757	21	1265	1768	
20	10	1768	46	1771	1772	Prepare to use the formula
20	11	1771	69	1774	1778	arctan (-x) = -arctan x
20	12	1772	69	1775	1778	at the end if $\overline{B} < 0$.
20	13	1778	24	1283	1786	
20	14	1786	68	8002	1795	
20	15	1795	15	1568	1773	Test 49-b; go to special
20	16	1773	35	0001	1779	routines if $\|49-b\| \geq 10$
20	17	1779	44	1783	1784	
20	18	1783	46	1937	1930	
20	19	1784	30	0005	1897	
20	20	1897	69	1901	1754	Prepare for shift of 49-b.
20	21	1754	22	1260	1763	
20	22	1763	46	1766	1918	If b > 49, go to arctan 1/x.
20	23	1918	16	1921	1926	If $b \leq 46$, set arctan x = x,
20	24	1926	46	1929	1930	go to end.
20	25	1930	60	1297	1264	
20	26	1929	65	1265	1769	If $46 < b \leq 49$, go to
20	27	1769	69	1922	1777	series evaluation
20	28	1766	60	1919	1923	
20	29	1923	64	1265	1770	If b > 49, calculate 1/B_1.
20	30	1770	69	1924	1777	
20	31	1777	24	1285	1788	
20	32	1788	35	0001	1260	Shift to get x=$\overline{B}$ or x=1/$\overline{B}$
20	33	1260	30	[49-b]	1781	into fixed decimal form
20	34	1781	67	8002	1789	
20	35	1789	20	1293	1796	If $\|x\| > .28$, go to calc.
20	36	1796	15	1799	1752	
20	37	1752	44	1755	1756	$z = \frac{\|x\|-y}{1+\|x\|y}$; y=.6
20	38	1755	60	1759	1913	
20	39	1913	19	1293	1927	
20	40	1927	35	0001	1933	
20	41	1933	10	1665	1920	
20	42	1920	21	1277	1780	
20	43	1780	60	1293	1797	$z = \frac{\|x\|-y}{1+\|x\|y}$.
20	44	1797	11	1759	1764	
20	45	1764	64	1277	1902	
20	46	1902	69	1905	1909	
20	47	1756	65	1293	1147	
20	48	1147	35	0001	1753	$z = \|x\|$. Prepare to
20	49	1753	69	1906	1909	add 0 or arctan y.
20	50	1909	24	1267	1570	
20	51	1570	20	1278	1931	
20	52	1931	60	8002	1939	
20	53	1939	19	8001	1945	
20	54	1945	21	1250	1903	
20	55	1903	61	8003	1911	
20	56	1911	19	1776	1940	
20	57	1940	60	8003	1149	
20	58	1149	10	1856	1762	

20	59	1762	19	1250	1790	
20	60	1790	60	8003	1798	
20	61	1798	11	1855	1914	
20	62	1914	19	1250	1938	arc tan z =
20	63	1938	60	8003	1148	
20	64	1148	10	1853	1760	$= z[1 - \frac{z^2}{3} + \frac{z^4}{5} - \ldots - \frac{z^{10}}{11}]$
20	65	1760	19	1250	1787	
20	66	1787	60	8003	1146	(coefficients from LOG)
20	67	1146	11	1852	1907	
20	68	1907	19	1250	1904	
20	69	1904	30	0001	1912	
20	70	1912	60	8003	1782	
20	71	1782	10	1785	1791	
20	72	1791	19	1278	1932	
20	73	1932	60	8003	1143	(correction at the end)
20	74	1267	10	[ ]	1285	arctan $\lvert x \rvert$ = arctan z + [0 or arctan y]
20	75	1285	[ ]	8003	[ ]	
20	76	1943	15	1701	1910	If b>49, arctan $\lvert x \rvert = \pi/2 -$ arctan $1/\lvert x \rvert$.
20	77	1910	35	0001	1934	
20	78	1934	44	1942	1793	
20	79	1942	31	0003	1915	If arctan $\lvert \bar{B} \rvert \geq 1$, round
20	80	1915	35	0002	1935	in 7th decimal place, if
20	81	1793	31	0002	1916	< 1, round in 8th.
20	82	1916	35	0001	1935	
20	83	1935	60	8002	1944	
20	84	1944	36		1765	Normalize arc tan $\lvert \bar{B} \rvert$
20	85	1765	11	8002	1864	
20	86	1864	10	1518	1283	
20	87	1283	[ ]	8003	1264	$\bar{C}$ = arc tan $\bar{B}$ in radians
20	88	1264	[ ]	[ ]	[ ]	For degrees, go to MPY
20	89	1936	60	1794	1655	by $360/2\pi$.
20	90	1937	65	1701	1910	For $b \geq 59$, arc tan $\lvert \bar{B} \rvert = \pi/2$.
20	91	1758			1445	
20	92	1908	21	1283	1936	
20	93	1928	60		1941	
20	94	1774	61	8003	1264	
20	95	1775	60	8003	1264	
20	96	1568	49			
20	97	1901	30		1781	
20	98	1921		0003		
20	99	1922	65	8003	1910	
20	100	1919		0100		
20	101	1924	66	8003	1943	Constants
20	102	1799	97	2000		
20	103	1759	06			
20	104	1665	10			
20	105	1905	10	1731	1285	
20	106	1906	10	1580	1285	
20	107	1785	20			
20	108	1731	05	4041	9500	
20	109	1580				
20	110	1701	15	7079	6327	
20	111	1518			0050	
20	112	1794	57	2957	8051	
20	113	1801	50			
20	114	1143	19	1801	1267	Correction in arc tan z.

Form No. 34-6822-0

IBM®

DATA PROCESSING

INTERNATIONAL BUSINESS MACHINES CORP.
590 MADISON AVENUE, NEW YORK 22, N. Y.

Litho. in U.S.A.