JOHNNIAC Room

JOHNNIAC FLOATING-POINT INTERPRETIVE SYSTEM

by John I. Derr

August 31, 1955

JOHNNIAC FLOATING-POINT INTERPRETIVE SYSTEM

INTRODUCTION

This system is a floating-point interpretive system, and as such its primary function is to facilitate the execution of the elementary arithmetic operations (add, subtract, multiply, and divide) and, in addition, some of the more frequently used elementary mathematical function operations (square root, sine cosine, arc tangent, exponential, and logarithm). Entry into the system is effected by basic linkage under stored-program control. Upon entry into the system, all succeeding operations are carried out as pseudo-orders under the control of the interpretive system until one of a unique pair of exit orders is encountered.

The system is, relatively speaking, almost complete within itself. In addition to the arithmetic operations mentioned, the system recognizes input and output orders for floating-point data, conditional and unconditional transfer orders, indexing orders for the modification of addresses and the execution of repetitive loops, and indications (from both external and stored-program control) to print out the results of specified operations.

The logical form of the orders, which will be recognized by the interpreter, is very similar to that of the JOHNNIAC itself. (This system is basically a two-operation, two-address per instruction word system with an interpretation cycle of fetch, left, right. The same points hold true for the JOHNNIAC, and as a result, both are essentially single-address.) Another analogy is the agreement of

both numeric and mnemonic order codes for most of the operations which exist in both the JOHNNIAC and the floating-point system. The extent to which the over-all logic of this system conforms with that of the JOHNNIAC makes possible the integration of this system into the entire system of utility programs and sub-programs written or to be written for the JOHNNIAC. For example, that section of a problem code which is to be executed under interpreter control can be modified in the same way as machine-language code. The same assembly program can be used to process floating-point orders as that which is used to process machine-language code including the floating-point system itself, all of which implies that within a program we can conveniently intersperse fixed-point arithmetic, floating-point arithmetic, and logical operations in the ratio required by the problem being solved.

I. WORD FORM FOR FLOATING POINT DATA

The JOHNNIAC is a high-speed computer with an approximate add-time of 80 μs and 4096 words of high-speed magnetic core storage. A JOHNNIAC word contains 40 binary bits with the binary point to the right of the left-most bit. When data words are being considered, the left-most bit functions as a sign indicator and the remaining 39 bits represent the magnitudes of the data. Negative numbers are represented in complement form.

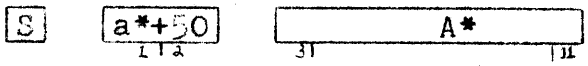
At all times the numerical data, which are operated upon or which are being transmitted to or from the high-speed storage under interpreter control, are in single-precision, floating-decimal form: i.e.,every piece of data can be expressed in the form $\pm M \cdot 10^m$ where M is a positive nine decimal digit number with a fixed decimal point and m is a two decimal digit integer. M is called the mantissa and m is the exponent of the number. The sign ($\pm$) is associated with the mantissa.

A. External Form of Data

By the external form of a piece of numerical data X we mean either the form in which X is punched into a card for input to high-speed storage or the form in which X is printed or punched as output from high-speed storage. First of all, X can be uniquely represented as follows:

(F) $X = \pm A^* \cdot 10^{a^*}$, where A^* is a positive 9-digit decimal fraction or zero, i.e. $10^{-9} \leq A^* < 1$ or $A^* = 0$, and a^* is a decimal integer in the range $-50 \leq a^* < 50$.

In order to eliminate the sign ($\pm$) from the exponent, we can increase a^* by 50. Then $0 \leq a^* + 50 < 100$. We shall at times refer to (F) as the implicit fractional form of X, and call a^* the true exponent or the implicit exponent of X. In the same way, let us call $a^* + 50$ the explicit exponent of X.

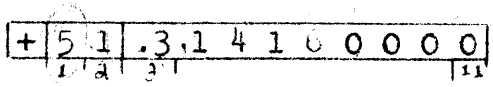


External Form of Data

A^* is a positive 9-digit decimal fraction, a^*+50 is the true exponent increased by 50, and S is the sign associated with A^*. In summary, the external form of X is that of sign, fractional mantissa, and true exponent +50.

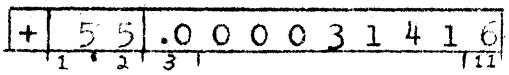
Examples: Prepare $\pi \doteq 3.1416$ for input in 1) and 2) below.

1) $3.1416 = +.3141600000 \cdot 10^{1}$ (implicit fractional form)



(external form)

2) $3.1416 = +.0000\ 31416 \cdot 10^{5}$ (implicit fractional form)



(external form)

Note: The mantissa A^* need not be normalized. (See Section D below for definition of "normalized".)

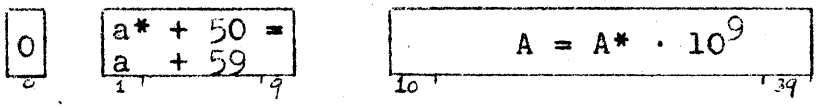
B. Internal (Packed) Form of Data

From (F) above, it follows that X can also be represented

as follows:

(I) $X = \pm A \cdot 10^{a}$, where $A = A^* \cdot 10^9$ is a positive 9-digit decimal integer or zero, i.e. $1 \leq A < 10^9$ or $A = 0$, and $a = a^* - 9$ is a decimal integer in the range $-59 \leq a < 41$.

We shall refer to (I) as the implicit integer form of X. By combining certain aspects of all three of the mentioned representations of X, we have as a result the explicit representation of X in its internal form.



Internal Packed Form of Data

The above representation holds true if X is non-negative. If X is negative, the entire word is complemented.

Note that in order to transform A* into A, A* is not explicitly multiplied by 10^9, since when A* is read from a card it is simply converted as an integer. Note also that a* + 50 = a + 59. As a result, no extra arithmetic is necessary when converting the external form of X into the internal form, since the only difference between the two is the location of the decimal point of the mantissa.

Example: Consider again $\pi \doteq 3.1416$.

$3.1416 = + \ . \ 3\ 1\ 4\ 1\ 6\ 0\ 0\ 0\ 0 \cdot 10^{1}$ (implicit fractional form)

$3.1416 = + \ \ 3\ 1\ 4\ 1\ 6\ 0\ 0\ 0\ 0 \cdot 10^{-8}$ (implicit integer form)

+	5 1	. 3 1 4 1 6 0 0 0 0	external form

+	5 1	3 1 4 1 6 0 0 0 0 .	internal form

Note: The only difference between the last two forms is the location of the decimal point.

Three reasons for choosing this system for the representation of data are enumerated below:

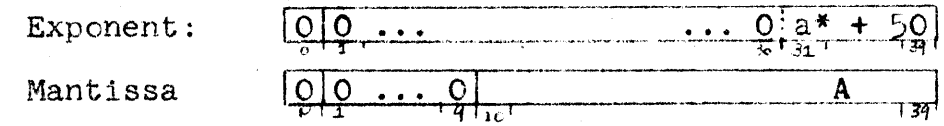
1) Exact reconversion of input data is made possible. For example, any number X is exactly the same after having been read into the machine and then printed as it was before either of these operations was performed. Therefore, under certain limitations to be specified later, this system can simulate a true decimal computer.

2) The form of packing data internally makes possible algebraic and magnitude comparisons of data while outside of the interpreter control.

3) Nine digit mantissas were chosen because $2^{29} < 10^9 < 2^{30}$ and $100 < 2^7 < 2^9$.

C. Internal (Unpacked) Form of Data

All arithmetic operations executed under interpreter control involve an arithmetic pseudo-register called the AMQ. All floating-point numbers placed in the AMQ are in their unpacked form, which is as follows:



Internal Unpacked Form of Data

The above representation holds true if the mantissa is non-negative. Otherwise, the mantissa only is complemented.

Then the AMQ consists of two adjacent full words of storage, and its location is permanent, relative to the location of the interpretive program. The function of the

AMQ is closely tied to the one-address nature of the system. Some of its properties are listed below:

1) The AMQ receives floating-decimal numbers from storage to initiate a sequence of operations. In the process of being transmitted from storage to the AMQ, all numbers are converted from their packed form to their unpacked form.

2) Conversely, at any stage of a sequence of operations the number retained in the AMQ may be transmitted to storage, and in this process it will be converted from its unpacked form to its packed form.

3) All "binary" arithmetic operations executed under interpreter control involve two operands, one of which is in the AMQ, with the other in the location specified by the address part of the operation. Furthermore, the result of the operation will be placed in the AMQ. For example, consider the operation X+Y. One of the operands (say X) must originally be in the AMQ, and the other must be at a specified location in storage. The result (X+Y) of the operation will be found in the AMQ upon completion of the operation. All "unary" arithmetic operations executed under interpreter control involve only the contents of the AMQ, and the result of any such operation will be placed in the AMQ. For example, consider the operation sin X; X must originally be in the AMQ,

and the result (sin X) of the operation will be found in the AMQ upon completion of the operation.

D. Normalizing and Significant Digit Modes

Two mutually exclusive modes of executing arithmetic operations are available. The interpreter is said to be in the normalizing mode (the N mode) if the results of all elementary arithmetic operations (+, -, x, ÷) and all elementary mathematical function operations are normalized prior to being placed in the AMQ. A number is normalized if $10^8 \leq A < 10^9$ or $A = 0$. If $A = 0$, then $a^* + 50 = 0$ also. Note that if $A \neq 0$, this amounts to saying that the most significant (left-most) position of the mantissa contains a non-zero digit.

Conversely, the interpreter is said to be in the significant digits mode (the SD mode) if for the results of all elementary arithmetic operations and mathematical function operations the following statement holds true: Roughly speaking, carry only as many significant digits as would be justified by the theory of error analysis with the possible exception of guarding figures.

The system is in the N mode if console switch T_1 is off.

The system is in the SD mode if console switch T_1 is on.

Examples: X, Y are in the packed internal form.

1)	X ⟷	59.1234 00000	
	Y ⟷	59.1201 00000	
	X-Y ⟷	59.0033 00000	(SD mode)
	X-Y ⟷	57.3300 00000	(N mode)
2)	X ⟷	53.0030 00000	
	Y ⟷	51.1000 00000	
	X·Y ⟷	53.0030 00000	(SD mode)
	X·Y ⟷	51.3000 00000	(N mode)

Note that the mantissa of X is less significant than that of Y. Therefore, the number of significant digits of the mantissa of X·Y depends upon the number of significant digits of X. (See Section III for a more detailed discussion of significant digits.)

E. Summary of Section I

The external form for representing data is that of sign, fractional mantissa, and true exponent + 50, and the user is required to know only this form of representation. The internal form for representing data is that of integer mantissa, true exponent + 59, and complementation for sign. This form has been described for the primary purpose of giving the user an insight into the system so that he may take full advantage of the opportunities available to him. It might be pointed out here that the user might need to know the internal form for representing data if he desires to convert fixed point numbers into floating point numbers,

and vice versa. However, one way of performing the conversion has been described in detail in Section V.

A primary advantage of treating floating point mantissas as integers has already been mentioned in paragraph B of this section, namely, the exact reconversion of input data. This is made possible by the fact that decimal integers and binary integers convert exactly one into the other.

But this system simulates a true decimal machine in a broader class of operations than just Input-Output operations. The results of all arithmetic operations are truncated decimally and without rounding. Then the results of any sequence of operations, with the exception of the elementary mathematical function operations, can be simulated exactly on a desk calculator by using nine digit operands. However, the algorithm for carrying out the simulation becomes more complicated when operations are carried out with unnormalized operands. A systematic way of accomplishing this simulation will be described for each order in Section IV.) Once more, we can say that this simulation is possible because no binary truncation is involved. But the simulation of decimal numbers is not closed under the mathematical function operations, because for these operations it is most desirable to transform at least the mantissa into a proper binary fraction. This transformation, as well as the ensuing calculations necessary to compute the resulting functional value, involves binary truncation,

and it is a well-known fact that in general decimal fractions do not convert exactly into binary fractions, and vice versa.

II. INSTRUCTION WORD FORMS FOR FLOATING POINT OPERATIONS

It was asserted in the Introduction that the interpretation of floating point orders is similar in many respects to the interpretation of JOHNNIAC machine language orders. In what follows we shall describe explicitly the logical form of the floating-point orders. However, we might take as our point of departure a brief description of the way in which the JOHNNIAC interprets instruction words.

0 1	7	18 19 20	21 22	27 28	39
Left Operation	Left Address	Not Used	Right Operation	Right Address	
Left Order			Right Order		

JOHNNIAC Instruction Word Form (J)

The octal operation codes are restricted to the range 000 - 177_8 (128 possibilities). The octal representations of addresses are restricted to the range 0000 - 7777_8 (4096 possibilities).

It has been mentioned in the Introduction that the basic interpretation cycle of the JOHNNIAC is fetch, left, right; i.e., first a word of the form (J) is fetched from storage, then the left order is executed, and finally the right order is executed. As is usual in the execution of an order the operation part of the order takes precedence over the address part of the order. The JOHNNIAC operation list has been arranged so that operations which are similar to each other in some respect are grouped in the same class, where the class is defined by the two most significant

octal digits of the octal operation code. As a result, the classes range from 00_8 to 17_8. See p. 83 for a list of JOHNNIAC operations.

A. Operation Types and Classes

Perhaps it will be convenient at this point to break up the list of floating point operations into three basic types: Logical-Control type operations, Arithmetic type operations, and Input-Output type operations. The Logical-Control type operations include the conditional and unconditional transfer operations, the Indexing operations, the operations effecting exit from the interpreter control, and also the "No Operation" operation. The Arithmetic type operations include the operations necessary for the transmission of floating-point data between the high-speed storage and the AMQ as well as the elementary arithmetic operations and the elementary mathematical function operations. Those floating-point operations which are used to transmit floating-point data between the high-speed storage and any of the mechanisms used for reading cards, punching cards, or printing, constitute the Input-Output type operations.

With the exception of orders executed in the Indexing mode and the Input-Output type operations this system is, like the JOHNNIAC, a two-operation, two-address per instruction word system with an interpretation cycle of fetch, left, right. Under the same restrictions the instruction word form is unchanged except that the fields for the opera-

tion codes have been defined as follows:

0	1 6	7 18	19 20	21	22 27	28 39
Left Con.	Left Oper.	Left Address	Not Used	Rt Con.	Right Oper.	Right Address
Left Order				Right Order		

Floating-Point Word Form (F)

The control fields are used for special control indications to the interpreter (with the exception of the Input-Output type orders). For example, the presence of a "1" in the Left Control field can cause the breakpoint printing of the order immediately after the order is executed. The octal operation codes are now restricted to lie in the range $00\text{-}77_8$ (64 possibilities). Recall that the operation part of an order is that part which takes precedence over all others. Note that the operation parts in (F) coincide with the least significant 6 binary (2 octal) digits of the corresponding operation parts in (J), and that the control parts in (F) coincide with the most significant binary (octal) digits of the corresponding operation parts in (J). For convenience of exposition in what follows, the term "operation" will be used interchangeably in either the sense of (J) or that of (F). However, after having noted the distinction between the two meanings, the reader should experience no difficulty from this direction.

In analogy with the concept of classes of operations for the JOHNNIAC, it is reasonable for us to group the floating-point operations according to eight classes (a zero

class, a one class, ..., a seven class), where the class is defined by the most significant octal digit of the octal operation code. (See p. 84 for a complete list of the floating-point operations.) Note that the Logical-Control type operations are in classes 0,1, and 7, the Arithmetic type operations are in classes 2, 3, 4, and 5, and the Input-Output type operations are in classes 0 and 1.

B. Indexing Mode

An Indexing mode (X mode) for interpretation of floating-point orders and a corresponding class (7) of Indexing orders have been incorporated into the interpretive system in order to facilitate the address modification and the counting involved in the execution of the repetitive loops which occur in a program. Immediately following the execution of the Enter Indexing operation, the Interpreter will be in the Indexing Mode. Then all succeeding orders will be interpreted in the Indexing Mode until a "1" is encountered in the Right Control field. The presence of a "1" in the Right Control field will always cause the Interpreter to exit the Indexing Mode. For this reason the Right Control field will be referred to as the Exit Indicator field while the Interpreter is in the X Mode.

When interpreting orders in the X Mode this system becomes a one operation per instruction word system with an interpretation cycle of fetch, left. The system also remains a single-address one, since one high-speed storage cell at most can be referred to in a single order. The in-

struction word form for orders executed in the Indexing Mode is as follows:

0	1 6	7 18	19 20	21	2 3 4 5 6 7	28 39
Con	Oper.	Left Address	Not used	X Ind.	X T A G	Right Address

Floating-Point Word Form (X Mode)

Only the Left Operation functions as an operation. The Right Operation field contains the Indexing Tag (X Tag) which can be used to specify uniquely any of the 64 possible combinations (including the combination where none of the indexing registers is involved) of six Indexing Registers which are involved in a given operation. Each of the Indexing Registers contains two quantities $X_{(\ )}$ and $\Delta X_{(\ )}$. The primary function of $X_{(\ )}$ is at execution time only to modify the addresses of arithmetic type orders which are executed in the Indexing Mode, and which have X Tags referring to $X_{(\ )}$. The principal use of $\Delta X_{(\ )}$ is to modify the corresponding $X_{(\ )}$ upon execution of a Transfer on Positive Index order or a transfer on Negative Index order. Each of the six Indexing Registers occupies one permanent full-word of storage within the Interpretive System as follows:

0 6	7 18	19 27	28 39
Zero	$X_{(\ )}$	Zero	$\Delta X_{(\ )}$

Indexing Register Layout

Obviously, both $X_{(\ )}$ and $\Delta X_{(\ )}$ contain numbers in the range 0000 - 7777_8. Negative numbers are represented in

complement form. For example, if $X_{(\ )} = -7$ and $\Delta X_{(\ )} = -1$, then the corresponding Indexing register would contain:

Zero	$7771_8 = 4089_{10}$	Zero	$7777_8 = 4095_{10}$

Thus far, we have made no attempt to assign to each of the Indexing Registers a unique name. Consider the figure below:

22	23	24	25	26	27
A	B	C	D	E	F

If we agree to use $X_{(\ )}$ to designate the Index Register containing $X_{(\ )}$, as well as $X_{(\ )}$, then we can enumerate the Indexing Registers as X_A, X_B, ..., X_F. We can, in a one-to-one manner, associate X_A with binary position 22 and likewise for the other Indexing Registers.

Now we can adopt the convention that the presence of a "1" in any of the binary tag positions means that the corresponding Indexing Register is involved in the execution of the given order, and that the presence of a "0" in the same position implies the opposite condition. Since it is natural to express JOHNNIAC operation codes in octal and since the Tag part of our orders coincides with the Right Operation of form (F), then it will be convenient to associate the octal representation of the Tags for each of the Indexing Registers as follows:

A $\longleftrightarrow$ 40, B $\longleftrightarrow$ 20, C $\longleftrightarrow$ 10

D $\longleftrightarrow$ 04, E $\longleftrightarrow$ 02, F $\longleftrightarrow$ 01

Example: Assume that the X Tag field contains 65_8.

$65_8 = 110\ 101_2$.

$= 40_8 + 20_8 + 04_8 + 01_8$.

According to either of the right-hand members above, Indexing Registers X_A, X_B, X_D, and X_F are involved in this order.

Given two modes of interpretation, the X Mode and the NX Mode, and two categories of operations, the Indexing operations (7 class operations) and the Non-Indexing operations, we have four logical possibilities:

1) Indexing orders executed in the X Mode,

2) Indexing orders executed in the NX Mode,

3) Non-Indexing orders executed in the X Mode, and

4) Non-Indexing orders executed in the NX Mode.

Condition 4 represents the standard situation, and the instruction word form is that of (F). The only order which satisfies Condition 2 is the Enter Indexing order. The instruction word form is also that of (F). Therefore, we can say that all orders executed in the NX Mode have the instruction word form of (F).

Conversely, all orders which are executed in the X Mode have the basic instruction word form of (X Mode). These orders, of course, satisfy Conditions 1 and 3. Although both Conditions 1 and 3 have the basic word form of (X Mode), there is a fundamental difference between the two conditions insofar as the contents of the Address fields are concerned.

Under Condition 3 we exclude the Input-Output orders because the Input-Output type of operations cannot be

executed in the X Mode. We do explicitly include all of the operations which are neither Indexing operations nor Input-Output type operations. Then under Condition 3 we have the instruction word form of (X Mode) modified to be:

0	1 6	7 18	19 20	21	2 3 4 5 6 7	28 33	4 5 6 7 8 9
Con	Oper.	Left Address	Not Used		X T A G	Blank	C L U E

Floating-Point Word Form (Condition 3)

Note that this form differs from the form X Mode only in the Right Address.

The reader can skip the remainder of Paragraph B on the first reading without affecting the continuity of thought. The six least significant bits of the word correspond in a one-to-one fashion with the six bits of the X Tag. Ordinarily, the Clue field will be left blank. However, if the X Tag field corresponding to X_A contains a zero and if some other X Tag field contains a "1", then if the user will place the numerical representation of the first $X_{(\ )}$, which has a "1" in its corresponding X Tag field, in the Clue field, the time required for executing the order will be decreased. (The saving of time results from the fact that the bits of the X Tag are examined from left to right, ordinarily beginning with A.)

Note that if the information contained in the Address fields is being processed by an assembly program as decimal information and if the information contained in the Operation fields is being processed as octal information, then the user must convert the numerical representation of the

$X_{(\ )}$ from octal to decimal. Perhaps the following table and an example will help to fix the idea:

X Register Tag	Octal Equiv.	Dec. Equiv.
A	40	32
B	20	16
C	10	08
D	04	04
E	02	02
F	01	01

Examples: 1. X TAG = 26_8.

CLUE = $20_8 = 16_{10}$.

(or = Zero).

2. X TAG = 07_8

CLUE = $04_8 = 04_{10}$.

(or = Zero).

We shall now emphasize the function of the Indexing Registers under Condition 3. For any Non-indexing order under Condition 3 we define the "Effective Address" of that order to be the sum of the Left Address plus all of the $X_{(\ )}$ which have a "1" in the corresponding X Tag position. The Effective Address is computed at interpretation time, and it is the address associated with the execution of the order. It is important to note that the Left Address of the instruction word as it was stored in high-speed storage is left unchanged by the execution of the order

Examples: Assume $X_A = 10_{10}$, $X_B = 5_{10}$, $X_C = 20_{10}$.

1. Consider the following Reset and Add order which is being executed in the X Mode:

0 6	7 18	21 27	28 39
0 2 0	0 9 0 0	0 5 0	0 0 0 0

The Effective Address = 900 + 10 + 20 = 930.

Then the result of this order is to place the contents of storage cell 0930 into the AMQ in unpacked form.

2. Consider the following Multiply order which is stored in storage cell 1000 under the same conditions as in example 1:

0 6	7 18	21 27	28 29
0 3 2	2 8 0 0	1 1 0	0 0 0 8

The Effective Address = 2800 + 20 = 2820. There are two significant results of this operation:

1) The contents of the AMQ will be multiplied by the contents of memory cell 2820, and the floating-point product will be placed into the AMQ in unpacked form.

2) The Interpreter will exit from the Indexing Mode; i.e., the instruction word stored in storage cell 1001 will be executed in the NX Mode.

We have seen that the application of the Indexing Registers for the purpose of modifying addresses is performed by orders executed under Condition 3. However, the operations performed upon the Indexing Registers themselves lie strictly in the domain of the Indexing orders executed under Condition 1. The precise operations which can be performed on the Indexing Registers will be discussed in Section IV. Let it suffice here to say that for each of the Indexing

Registers there exist operations for 1) setting $X_{(\ )}$ and $\Delta X_{(\ )}$ = to given values, 2) increasing $X_{(\ )}$ and $\Delta X_{(\ )}$ by given values, and 3) increasing $X_{(\ )}$ by $\Delta X_{(\ )}$ and then testing $X_{(\ )} + \Delta X_{(\ )}$ = a given value.

C. Input-Output Word Form

The Input-Output type of operations have been grouped together under this one type primarily because they have a word form which differs from both of the forms (F) and (X Mode). However, this word form is quite similar to the form (X Mode). Once more we have only one operation per instruction word, and hence at the same time an interpretation cycle of fetch, left. All addresses both between and including the ones specified are involved in the execution of the order. Below is the basic instruction word form for all Input-Output type operations:

0	1 6	7 18	19 20	21	22-24	25-27	28 39
0	Oper.	First Address	Not Used	A	B	C	Last Address

Input-Output Word Form

The left Operation field functions as the operation field for the order.

The First Address field contains the address of the first floating-point number to be transmitted between the high-speed storage and the input-output mechanism specified by the operation field. Similarly, the Last Address field contains the address of the last floating-point number. Of course, these fields coincide with the address fields of the other instruction word forms.

The fields A, B, and C coincide with the octal digits of the Right Operation field in the sense of (J). The function of the A, B, C fields is to specify to the interpreter the form which the data will take at the specified input-output mechanism. With respect to reading and punching cards, for instance, the user can specify the number of data words per card.

In the conclusion of Section II, we should like to point out to the reader that in this section we have emphasized the differences existing among all of the various instruction word forms. In doing so, we have presupposed that the similarities would speak for themselves. Most of the differences consist in calling the same fields by different names. All of the forms are similar in one very important respect; they are, with minor exceptions to be pointed out, compatible with all existing and proposed assembly programs.

III. SYSTEM PHILOSOPHY FOR THE ARITHMETIC TYPE OPERATIONS

A. Significant Digits

We shall define the significant digits of any decimal number to be that set of digits which consists of all of the non-zero digits (1, 2, ..., 9) and in addition all of the zero digits which lie to the right of some non-zero digit. The most significant digit is defined to the the first non-zero digit from the left. The least significant digit is defined to be the right-most significant digit. We shall denote by S_N the number of significant digits of N.

Examples:

1) N = 0 0 9 8 0 0 1 0 0 .

The digits 9800100 are significant. The digit 9 is the most significant digit. The digit 0, which occupies the first position from the right of N, is the least significant digit. $S_N = 7$.

2) N = 0 0 0 0 0 0 0 0 0 .

There are no significant digits. $S_N = 0$.

B. Approximate and Exact Numbers

It is a well-known fact that certain classes of real numbers cannot be represented exactly by a finite number of digits in the decimal system. Examples of this phenomena include the transcendental numbers, the irrational numbers and many of the rational numbers. Explicit examples are π, $\sqrt{2}$, and 1/3, respectively. There also, of course, exist rational numbers (including integers) which can be repre-

sented by a finite number of decimal digits, but such that this finite number exceeds some preassigned number. For example, the number 1 2 3 4 5 6 7 8 9 1 cannot be represented exactly by nine decimal digits. In either of these cases we have examples of what we shall call approximate numbers. Under the class of approximate numbers we shall also include the computed results of operations performed upon either approximate numbers or, in some cases, exact numbers. By exact numbers we shall mean only those quantities which can be represented exactly by a given number (nine in our case) of decimal digits. Observe that exact numbers can arise as the result of arithmetic operations. However, note that the arithmetic operations performed by this interpretive system are pseudo-operations. (Decimal truncations are performed in accordance with the algorithms to be specified for each operation.) It is important for the reader to understand that, in order for the result of an operation to be exact, two necessary conditions must be satisfied:

1) The operands must have been exact.
2) The pseudo-operation performed on these exact operands must give the same result as the true operation; i.e., no information can be lost because of approximations, truncations, etc.

For extensive calculations carried out in the floating-point system, however, the class of exact numbers should in general be restricted to include only constants and data

which do not change during a calculation. Furthermore, these exact numbers should always be normalized. Extreme caution is advised when considering the results of arithmetic operations to be exact. Recall that in order for the result of an operation to be exact, both of the operands must have been exact. In addition the actual pseudo-operation performed must be thoroughly understood.

Examples: 1) Assume X = 1 0 0 0 0 0 0 0 1,
and Y = 1 0 0 0 0 0 0 0 1 to be exact.
Then X+Y = 2 0 0 0 0 0 0 0 2 is exact.

Also X x Y = 1 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 1 is exact. Denote by X o Y the result of X x Y after truncation to 9 digits. Then X o Y = 1 0 0 0 0 0 0 0 2 is not exact.

2) Assume X = 0 0 0 0 0 0 0 0 1 is exact. Then arc tan X = . 7 8 5 3 9 8 1 6 3 is not exact, since .785398163 is only an approximation to $\pi/4$.

C. Absolute Error and Relative Error

If we denote by N* the approximate number representing an exact number N, then we shall define the absolute error of N* to be N* - N. Let us denote the absolute error by ΔN. Then we shall define the relative error to be $(\Delta N) \div N$. (Ordinarily, $\Delta N \div N$ can be approximated by $\Delta N \div N^*$.)

In what follows now let us assume that our numbers N* are 9 digit decimal integers, some of the digits of which might not be significant. We shall also assume that these approximate numbers have been truncated decimally without

rounding, and that $|\Delta N| < 1$. This fact means that we are considering only the absolute error introduced by this last truncation, or that we are not including the accumulated error which has been propagated from previous pseudo-operations.

Example: $N = 0\,0\,0\,0\,0\,1\,2\,3\,4\,.\,9\,9\,9\cdots$

$N^* = 0\,0\,0\,0\,0\,1\,2\,3\,4\,.$

Therefore, $\Delta N = -\,.\,9\,9\,9\cdots$ and $|\Delta N| < 1$.

RULE I. The absolute value of the absolute error of the sum of two approximate numbers cannot exceed the sum of the absolute values of the absolute errors of the given numbers.

Example: Let $X = 0\,0\,0\,0\,1\,2\,3\,4\,5\,.\,9$ and

$Y = 0\,0\,0\,0\,5\,4\,3\,2\,1\,.\,9$. Then

$X^* = 0\,0\,0\,0\,1\,2\,3\,4\,5$ and $Y^* = 0\,0\,0\,0\,5\,4\,3\,2\,1$.

Therefore, $|\Delta X| = .\,9$ and $|\Delta Y| = .\,9$.

Now $X + Y = 0\,0\,0\,0\,6\,6\,6\,6\,7\,.\,8$ and

$X^* + Y^* = 0\,0\,0\,0\,6\,6\,6\,6\,6$. Therefore,

$|\Delta(X+Y)| = 1\,.\,8$, and so $|\Delta(X+Y)| \leq |\Delta X| + |\Delta Y|$.

Observe that the worst case is approached when ΔX and ΔY both approach one and are of the same sign.

RULE II. The absolute value of the relative error of the product or quotient of two approximate numbers cannot exceed the sum of the absolute values of the relative errors of the given numbers.

Example: Let $X = 0\,0\,0\,0\,0\,0\,0\,0\,1\,.\,9$

and $Y = 0\,0\,0\,0\,0\,0\,0\,0\,2\,.\,9$.

Then $\left|\frac{\Delta X}{X}\right| = \frac{.9}{1.9}$ and $\left|\frac{\Delta Y}{Y}\right| = \frac{.9}{2.9}$.

$X \cdot Y = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 5\ .\ 5\ 1$ and

$X^* \cdot Y^* = 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 2.$

Therefore $\left|\frac{\Delta(X \cdot Y)}{X \cdot Y}\right| = \frac{3.51}{5.51}$, and $\left|\frac{\Delta(X \cdot Y)}{X \cdot Y}\right| \leq \left|\frac{\Delta X}{X}\right| + \left|\frac{\Delta Y}{Y}\right|$.

These two rules give us a convenient way for positing an upper bound for the error introduced by truncation in any one of the pseudo-operations (+), (x), (÷). We can, however, rephrase the second rule into a more useful form. We are able to derive this alternate form from the original because of the close connection between the concepts of significant digits and relative error.

RULE III. The number of significant digits carried in the product or the quotient of two approximate numbers cannot in general be justified beyond the number of significant digits carried in the least significant of the two operands.

Proof: We shall outline a proof for the product.

Let X and Y be the two operands and let X be the more significant of X and Y; i.e., $S_X \geq S_Y$. Also, assume X and Y are both positive, and $\Delta X \doteq 1$, and $\Delta Y \doteq 1$.

We know that $(X + \Delta X)(Y + \Delta Y) = XY + Y\Delta X + X\Delta Y + \Delta Y\,\Delta X$, and that either $S_{XY} = S_X + S_Y$ or $S_{XY} = S_X + S_Y - 1$.

Since $\Delta Y \doteq 1$, $S_{X\Delta Y} = S_X$. Therefore, the S_X least significant positions of $(X + \Delta X)(Y + \Delta Y)$ cannot be justified.

Then $S_{XY} - S_X = S_Y$ if $S_{XY} = S_X + S_Y$, and $S_{XY} - S_X = S_Y - 1$ if $S_{XY} = S_X + S_Y - 1$. In either case, $S_{X^*Y^*} \leq S_Y$. (Recall that $X^* = X + \Delta X$.)

A similar argument exists for the quotient.

D. Guarding Figures

Rule III holds true in general; i.e., even in the worst possible case when ΔX and $\Delta Y = 1$, X and Y are of the same sign, and $S_X = S_Y$. However, if we assume a random distribution of truncated digits, we can say that the average value of ΔX or ΔY is 1/2. Furthermore, in many cases the relative errors approach zero, in which case we would be justified in keeping all or most of the generated digits of the product. As a compromise we can always retain some additional digits in the product or quotient. These additional digits we shall call guarding figures. In the floating-point system we have adopted the convention of keeping at most one guarding figure in the results of the elementary arithmetic type orders. Referring back to the preceding proof, we see that for the product $S_{XY} - (S_X - 1) = S_Y + 1$ or S_Y. The product is actually computed in an analogous manner so that S_Y or $S_Y + 1$ significant digits are carried in the final result; and the same method is applied for the quotient.

The decision to keep zero or one guarding figure in the results of the elementary arithmetic operations was influenced by several factors. First and foremost, the same computational algorithm is used for both of the cases of zero and one guarding figures. Whether or not an extra digit is retained in a result is determined only by the distribution of the digits in the operands. In order to make the computational algorithm independent of the distribution of the digits, it is necessary to retain a variable number

of guarding figures, and the domain of this number must be over two consecutive integers. Secondly, if a "negative number of guarding figures" were allowed, information would be needlessly lost. Thirdly, if more than zero or one guarding figure were retained, we would fast lose the primary advantage of the SD Mode.

This advantage of the SD Mode is to indicate the number of "good" justifiable digits which result from a calculation. During the course of a calculation the leading significant digits of numbers can be lost when performing the Add type operations, and significant digits can be gained by inserting guarding figures into the low order position of the result when performing any of the elementary arithmetic operations. Consequently the quality of the indication given by the number of significant digits of a result depends upon the particular sequence of calculations required to produce the result. For example, several significant digits may be lost in computing an intermediate sum, but subsequent operations can conceal this fact by retaining an extra guarding figure at each of several steps in the problem. Another advantage of the SD Mode is that the execution time for all arithmetic orders is shortened since the results are not normalized.

E. Application to the Normalizing Mode

The advantage of using the N Mode is that the maximum number of guarding figures is kept in the results of the Multiply and Divide operations, provided that the operands

are always normalized. If the operands are normalized, the non-zero products and quotients are computed so as to contain nine or ten significant digits. If the operands are not normalized, the products and quotients are computed according to Rule III modified to insert at most one guarding figure, and under normalization only zeros can be inserted into the least significant position of the mantissa. Note that this operation of effectively shifting the result to the left does not change the relative error, since the relative error is independent of the decimal point.

The advantage just mentioned for using the N Mode becomes important in the same ratio as the relative errors of the operands approach zero. This fact implies that all exact numbers should be kept normalized at all times since for exact numbers the relative errors equal zero.

Example: Input the number 2.

1) If the form is + 5 1 2 0 0 0 0 0 0 0 0, then the relative error is considered to be approximately $1 + (2 \cdot 10^{8})$.

2) If the form is + 5 9 0 0 0 0 0 0 0 0 2, then the relative error is considered to be approximately 1 + 2.

F. Summary of Section III

The conventions adopted for carrying out the "binary" elementary arithmetic operations were chosen so as to permit these operations to be performed on numbers which may or may not be normalized and to be carried out in either the N Mode or the SD Mode. It is more efficient from the stand-

point of execution time to post-normalize the results of these operations than to pre-normalize them. Storage space is saved by using the same set of internal instructions to carry out these operations (excluding post-normalization) independent of the mode of operation and the state of normalization of the operands.

The conventions which were adopted for these operations are:

1) The operands are not pre-normalized prior to the execution of the operation.
2) The calculation (prior to the final normalization) is always carried out as if the operands were not normalized and the mode of operation were the SD Mode. This means that decimal truncations might have been performed at some point during the calculation with the result that less than 9 significant digits are present in the resulting mantissa.
3) The result is normalized only if the system is in the N Mode. Note that only zeros can be inserted into the least significant positions of the mantissa during the normalization process.

The operands are always normalized prior to the execution of the mathematical function operations. The resulting functional values are normalized only if the system is in the N Mode.

Decimal truncation without rounding and the use of

integer mantissas were decided upon in order to permit this system to simulate a true decimal computer (except for the mathematical function operations). Decimal simulation and significant digits control have not been included in this system without certain accompanying disadvantages. The disadvantages involve basically a loss of efficiency in regard to space, time,and accuracy as described below:

1. The incorporation of a SD Mode and the use of integer mantissas result in more internal machine-language instructions than would be necessary if only the N Mode were available and fractional mantissas were used.
2. The admittance of unnormalized numbers and the use of integer mantissas increase the execution time for the elementary arithmetic operations.
3. Unrounded truncation and the significant digits method of computing products and quotients can result in a faster accumulation of truncation error.

Only the practical application of the system will determine whether or not the advantages of decimal simulation and significant digits control outweigh the accruing disadvantages just enumerated. The notions of decimal simulation and significant digits control are admittedly novel ones (at least relative to interpretive systems for high-speed binary computers), but they have been incorporated into this system anyway in the spirit of experiment.

IV FLOATING-POINT OPERATIONS

The discussion of the floating-point operations will be broken down into paragraphs which parallel the types of operations discussed in Section II. The only exception is that the Logical-Control type operations will be split into two parts. One part consists of the Indexing orders, and the other part consists of the Non-indexing orders. See p. 84 for a complete list of floating-point operations.

A. Arithmetic Type Operations

The data operated upon by the floating-point operations are assumed to be floating-point numbers of the internal form as discussed in Section I. Any of the Arithmetic type operations can be executed in either the Indexing Mode or the Non-indexing Mode.

The Arithmetic type of operations consists of the 2, 3, 4, and 5 classes of floating-point operations, and the discussion of the operations will be by classes.

1. The Two Class (Add Class) of Operations

OPERATION	Octal	Mnemonic	OPERATION	Octal	Mnemonic
Reset Add	20	RA	Add	24	A
Reset Subtract	21	RS	Subtract	25	S
Reset Add Absolute Value	22	RAV	Add Absolute Value	26	AV
Reset Subtract Absolute Value	23	RSV	Subtract Absolute Value	27	SV

List of Two Class Operations

RESET ADD RA Y 20

The contents of cell Y (a packed floating-point number) is first unpacked, and then the unpacked number replaces the contents of the AMQ.

RESET SUBTRACT RS Y 21

1) The contents of cell Y (a packed floating-point number) is complemented.
2) The result of Step 1 is unpacked.
3) The result of Step 2 replaces the contents of the AMQ.

RESET ADD ABSOLUTE VALUE RAV Y 22

1) Take the absolute value of the packed floating-point contents of Y.
2) The result of Step 1 is unpacked.
3) The result of Step 2 replaces the contents of the AMQ.

RESET SUBTRACT ABSOLUTE VALUE RSV Y 23

1) Take the negative absolute value of the packed floating point contents of Y.
2) The result of Step 1 is unpacked.
3) The result of Step 2 replaces the contents of the AMQ.

ADD A Y 24

1) The packed floating-point number in Y is first

unpacked, and then the unpacked number replaces the contents of the floating-point Number Register (NR). The Number Register corresponds to the JOHNNIAC Number Register in the same way that the AMQ corresponds to the JOHNNIAC Accumulator and Multiplier Quotient registers.

2) Compare the exponents of the NR and the AMQ. (Note that we use NR and AMQ here to mean the contents of the NR and AMQ respectively.) If the exponent of the NR exceeds that of the AMQ, then interchange the contents of the NR and the AMQ.

3) Compute the positive difference of the exponents of the NR and the AMQ. If the difference exceeds 8 or if the mantissa of the NR equals 0, then proceed to Step 7.

4) Divide (unrounded) the mantissa of the NR by 10 raised to a power equal to the difference of the exponents.

5) Add algebraically the result of Step 4 to the mantissa of the AMQ. Retain the ten-digit sum in the mantissa of the AMQ.

6)a. If the number of significant digits of the mantissa of the AMQ = 10, then replace the mantissa by the mantissa divided (unrounded) by 10, and increase the exponent of the AMQ by 1. If the resulting exponent exceeds 99, then we shall say that the Exponent Overflow condition exists. The machine

will halt at the Error Halt location, and if the Go button is pressed, then the results of this operation will be printed and the control will go to execute the next interpretation cycle. If the exponent does not exceed 99, then this operation is completed.

b. If the number of significant digits of the mantissa of the AMQ is less than 10, then proceed to Step 7.

7)a. If the interpreter is in the SD Mode, then this operation is completed.

b. If the interpreter is in the N Mode, find the number of significant digits in the mantissa of the AMQ. (We shall designate this number as S_{AMQ}.)

8)a. If $S_{AMQ} < 9$, then compute the positive difference $(9 - S_{AMQ})$. Proceed to Step 9.

b. If $S_{AMQ} = 9$, then this operation is completed.

9) Multiply the mantissa of the AMQ by $10^{(9 - S_{AMQ})}$. (Note that only zeros are inserted into the least significant positions of the mantissa.)

10. Place the product in the mantissa part of the AMQ. Decrease the exponent of the AMQ by $(9 - S_{AMQ})$. If the resulting exponent is negative, then we shall say that the Exponent Underflow condition exists. The machine will halt at the Error Halt location, and if the Go button is pressed, then the results of this operation will be printed

and the control will go to execute the next interpretation cycle. If the resulting exponent is positive, then this operation is completed.

Examples for the Add Operation:

1)	AMQ	+ 55999000000	
	NR	+ 51505006011	
		999000000	
		50500	(Step 4)
		999050500	(Step 5)
	AMQ	+ 55999050500	(Resulting Sum)
2)	AMQ	+ 51999000000	
	NR	- 51997005000	
		+ 999000000	
		- 997005000	(Step 4)
		+ 001995000	(Step 5)
	AMQ	+ 51001995000	(Sum if SD Mode)
	AMQ	+ 49199500000	(Step 9 if N Mode)
3)	AMQ	+ 50999999999	
	NR	+ 50000000002	
		+ 999999999	
		+ 000000002	(Step 4)
		+ 1000000001	(Step 5)
	AMQ	+ 51100000000	(Step 6a)
4)	AMQ	+ 50123456789	
	NR	- 50123456789	
	AMQ	+ 50000000000	(Sum if SD Mode)
	AMQ	+ 00000000000	(Sum if N Mode)
	NR	+ 41876543210	
	AMQ	+ 50000000000	(Sum if SD Mode)
	AMQ	+ 41876543210	(Sum if N Mode)

5)

AMQ	+ 50123456789	
NR	+ 41876543210	
AMQ	+ 50123456789	(Sum in either Mode)
NR	- 50123456789	
AMQ	+ 50000000000	(Sum if SD Mode)
AMQ	+ 00000000000	(Sum if N Mode)

In examples 4 and 5, the same three numbers are added in different order. These examples emphasize the relative importance of the exponents as compared with the mantissas for the Add orders executed in a floating-point system. The number of significant digits of the mantissas plays an equally strong role for the Multiply and Divide orders.

6)

AMQ	59000000001	(exact number)
NR	51123456789	(exact number)
AMQ	59000000002	(Sum is an approximate number)

The reader should verify the sum if the original number in the AMQ were normalized; i.e., 51100000000. This example illustrates the necessity for keeping exact numbers normalized.

According to Rule I of Section III, the maximum absolute error which can be introduced into the ten-digit sum in Step 5 of this operation is equal to the sum of the absolute errors of the addends. Observe that normalization will not produce any additional guarding figures for the Add operations. Normalization increases the absolute error corresponding to the amount of shifting required, while it leaves the relative error unchanged.

SUBTRACT S Y 25

1)a. The contents of cell Y (a packed floating-point number) is complemented.

b. The result of a. is unpacked.

c. The result of b. replaces the contents of the NR. Steps 2, 3, ..., 9 are the same as for the ADD operation.

ADD ABSOLUTE VALUE AV Y 26

1)a. Take the absolute value of the packed floating-point contents of Y.

b. The result of a. is unpacked.

c. The result of b. replaces the contents of the NR. Steps 2, 3, ..., 9 are the same as for the ADD operation.

SUBTRACT ABSOLUTE VALUE SV Y 27

1)a. Take the negative absolute value of the packed floating-point contents of Y.

b. The result of a. is unpacked.

c. The result of b. replaces the contents of the NR. Steps 2, 3, ..., 9 are the same as for the ADD operation.

2. The Three Class (Multiply Class) of Operations

OPERATION	Octal	Mnemonic
MULTIPLY	32	M
MULTIPLY NEGATIVELY	33	MN

List of Three Class Operations

MULTIPLY M Y 32

1) If the mantissa of the AMQ equals 0, then the exponent of the AMQ is set equal to 0, and this operation is completed.
2) The packed floating-point number in Y is first unpacked, and then the unpacked number replaces the contents of the NR.
3) If the mantissa of the NR equals 0, then the mantissa and the exponent of the AMQ are set equal to 0, and this operation is completed.
4) Find the number of significant digits (S_M) of the most significant of the mantissas of the AMQ and the NR.
5) Replace the mantissa of the AMQ by the double-length product of the mantissas of the AMQ and the NR, all divided (unrounded) by $10^{(S_M-1)}$.
6) Replace the exponent of the AMQ by the sum of the exponents of the AMQ and the NR increased by (S_M-1) and diminished by 59.
7) If the exponent of the AMQ is negative, then the Exponent Overflow condition exists. See Step 9 of the ADD operation.
8) If the exponent of the AMQ exceeds 99, then the Exponent Overflow condition exists. See Step 7 a. of the ADD operation.

Steps 9, 10, 11, and 12 are the same as Steps 6, 7, 8, and 9 for the ADD operation.

If we denote the original contents of the AMQ by $A \cdot 10^a$, the original contents of the NR by $B \cdot 10^b$, and the resulting contents of the AMQ by $C \cdot 10^c$, then we can summarize symbolically the MULTIPLY operation as follows:

$$C = (A \cdot B) + 10^{S_M - 1}, \quad c+59 = a+59 + b+59 + (S_M - 1) - 59.$$

Denote by $S_{A \cdot B}$ the number of significant digits of the double-length product $A \cdot B$. If $S_{A \cdot B} = S_A + S_B$, then if $S_{A \cdot B} \neq 18$ there will be one more significant digit in the resulting mantissa (C) than there were in the least significant of the Mantissas (A and B) of the AMQ and the NR. If $S_{A \cdot B} = S_A + S_B - 1$, then there will be as many significant digits in C as there were in the least significant of A and B. Assuming a uniform distribution of digits for A and B, the former condition (one more significant digit) will occur about 83% of the time. In either event one additional significant digit is obtained so far as the relative error is concerned.

If the user desires to simulate the Multiply operation on a desk calculator, he can combine the decision concerning whether or not to retain one extra significant digit in the product mantissa along with the calculation of the double-length product. Just perform the multiplication on the desk calculator by using normalized mantissas. If the lead digit of the product is zero, write the product mantissa with as many leading zeros as the least significant of A and B. Otherwise, write the product mantissa with one less leading zero than the least significant of A and B.

Note: In the N Mode, the result left in the AMQ is always normalized. Observe that only zeros are introduced into the least significant position.

Examples for the Multiply Operation

1)

AMQ	+ 50000000000	
NR	+ 51100000000	
AMQ	+ 00000000000	(Step 1)

2)

AMQ	+ 52040000040	$(A \cdot 10^a)$
NR	+ 55000050000	$(B \cdot 10^b)$

$S_{AMQ} = 8$ and $S_{NR} = 5$. Therefore $S_M = 8$ and $S_M - 1 = 7$.

$\frac{0000020000020000000}{10000000} = 000200000 = C.$

$c = 52 + 55 + 7 - 59 = 55.$

AMQ	+ 55000200000	(Product if SD Mode)
AMQ	+ 52200000000	(Product if N Mode)

Note that $S_{A \cdot B} = S_A + S_B$. For this reason, $S_{A \cdot B} = S_B + 1$.

3)

AMQ	+ 51400000400	
NR	+ 51500000000	

C = 2000002000 (Step 5)

c = 51

AMQ	+52200000200	(Product)

MULTIPLY NEGATIVELY MN Y 33

1) (Same as Step 1 of Multiply operation.)

2)a. The contents of cell Y (a packed floating-point number) is complemented.

b. The result of a. is unpacked.

c. The result of b. replaces the contents of the NR.

Steps 3 through 12 are the same as the same steps for the Multiply Operation.

3. The Four Class (Divide Class) of Operations

OPERATION	Octal	Mnemonic
DIVIDE	40	DS
DIVIDE NEGATIVELY	41	DNS

List of Four Class Operations

DIVIDE DS Y 40

1) The packed floating-point number in Y is first unpacked, and then the unpacked number replaces the contents of the NR.

2) If the mantissa of the NR equals zero, then we shall say that the Divide Check condition exists. The machine will halt at the Error Halt location, and if the Go button is pressed, then the status of this operation will be printed and the control will go to execute the next interpretation cycle.

3) If the mantissa of the AMQ equals zero, then the exponent of the AMQ is set equal to zero, and this operation is completed.

4)a. If the absolute value of the mantissa of the AMQ is less than that of the NR, then replace the mantissa of the AMQ by the double-length product of the mantissa of the AMQ and $10^{S_{NR}}$, all divided (unrounded) by the mantissa of the NR. Also re-

place the exponent of the AMQ by the difference of the exponents of the AMQ and NR increased by 59 and diminished by S_{NR}.

b. If the absolute value of the mantissa of the AMQ is equal to or greater than that of the NR and if $2S_{NR} \geq S_{AMQ}$, then replace the mantissa of the AMQ by the double-length product of the mantissa of the AMQ and $10^{(2S_{NR} - S_{AMQ})}$, all divided unrounded by the mantissa of the NR. Also replace the exponent of the AMQ by the difference of the exponents of the AMQ and NR increased by 59 and diminished by $(2S_{NR} - S_{AMQ})$.

c. If the absolute value of the mantissa of the AMQ is equal to or greater than that of the NR and if $2S_{NR} < S_{AMQ}$, then replace the mantissa of the NR by the full-length product of the mantissa of the NR and $10^{(S_{AMQ} - 2S_{NR})}$. Then replace the contents of the mantissa of the AMQ by the unrounded quotient of the mantissa of the AMQ divided by the mantissa of the NR. Also replace the exponent of the AMQ by the difference of the exponents of the AMQ and NR increased by 59 and $(S_{AMQ} - 2S_{NR})$.

5) The remaining steps are the same as Steps 7 through 12 for the Multiply Operation.

SYMBOLIC SUMMARY OF DIVIDE OPERATION

a. If $|A| < |B|$, then

$$C = \frac{A \cdot 10^{S_B}}{B} \quad \text{and} \quad c+59 = a+59 - (b+59) + 59 - S_B.$$

b. If $|A| \geq |B|$ and $2S_B \geq S_A$, then

$$C = \frac{A \cdot 10^{(2S_B - S_A)}}{B} \quad \text{and} \quad c+59 = a+59 - (b+59) + 59 - (2S_B - S_A).$$

c. If $|A| \geq |B|$ and $S_A > 2S_B$, then

$$C = \frac{A}{B \cdot 10^{S_A - 2S_B}} \quad \text{and} \quad c+59 = a+59 - (b+59) + 59 + (S_A - 2S_B)$$

Note: If the user desires to simulate the Divide Operation on a desk calculator, he can combine the decision concerning whether or not to retain one extra significant digit in the quotient mantissa along with the calculation of the unrounded quotient. Just perform the division on the desk calculator by using normalized mantissas. If the quotient is less than one, write the quotient mantissa with as many leading zeros as the least significant of A and B. Otherwise, write the quotient mantissa with one less lead zero than the least significant of A and B. Assuming a uniform distribution of digits for A and B, both of these cases are equally likely. In either event one additional significant digit is obtained so far as the relative error is concerned. In the N Mode the quotient left in the AMQ is always normal-

ized. Observe that only zeros are introduced into the least significant positions.

Examples for the Divide Operation:

1)

AMQ	+ 58000000050	$(A \cdot 10^a$
NR	+ 57000010000	$B \cdot 10^b)$

$|A| < |B|$ and $S_B = 5$.

$$C = \frac{50 \cdot 10^5}{10000} = 500.$$

$c = 58 - 57 + 59 - 5 = 55.$

AMQ	+55000000500	(Quotient if SD Mode)
AMQ	+49500000000	(Quotient if N Mode)

2)

AMQ	+ 57000010000	$(A \cdot 10^a$
NR	+ 58000000900	$B \cdot 10^b)$

$|A| \geq |B|$ and $2S_B \geq S_A$. $S_A = 5$ and $S_B = 3$.

$2S_B - S_A = 1.$

$$C = \frac{10000 \cdot 10^1}{900} = 111.$$

$c = 57 - 58 + 59 - 1 = 57.$

AMQ	+ 57000000111	(Quotient if SD Mode)
AMQ	+ 51111000000	(Quotient if N Mode)

3)

AMQ	+ 56009000000
NR	+ 57000000500

$|A| \geq |B|$ and $S_A > 2S_B$. $S_A = 7$ and $S_B = 3$.

$S_A - 2S_B = 1.$

$$C = \frac{9000000}{500 \cdot 10^1} = 1800.$$

$c = 56 - 57 + 59 + 1 = 59.$

AMQ	+ 59000001800	(Quotient if SD Mode)
AMQ	+ 54180000000	(Quotient if N Mode)

DIVIDE NEGATIVELY DNS Y 41

1) a.The contents of cell Y (a packed floating-point number) is complemented.

b. The result of a. is unpacked.

c. The result of b. replaces the contents of the NR.

Steps 2 through 10 are the same as for the Divide Operation.

4. The Five Class of Operations

OPERATION	OCTAL	MNEMONIC	OPERATION	OCTAL	MNEMONIC
Store	50	ST	Arc Tangent	54	ART
Square Root	51	SQR	Exponential	55	EXP
Sine	52	SIN	Logarithm	56	LOG
Cosine	53	COS			

List of Five Class Operations

STORE ST Y 50

The unpacked floating-point number in the AMQ is packed and then the packed number replaces the contents of Y.

SQUARE ROOT SQR 51

1) If the mantissa of the AMQ equals zero, then this operation is completed.
2) Normalize the contents of the AMQ. (Note that normalization can cause an Exponent Underflow condition.)
3) If the mantissa of the AMQ is negative, the machine will halt at the Error Halt location. If the Go

button is pressed, then the present state of the operation will be printed. Then the mantissa of the AMQ will be complemented. Proceed to Step 4.

4) Replace the mantissa of the AMQ by the unrounded quotient of the mantissa divided by 10^9 (the contents of the AMQ are now in the internal fractional form). Denote the mantissa by A.

5) If $A < 1/2$, set $y_0 = 1/4 + A$. If $A > 1/2$, set $y_0 = 1/2 + 1/2\ A$. With y_0 as a "first guess" compute $\sqrt{A}$ using a Newton iteration process. The $\sqrt{A}$ is computed using 39 binary bit arithmetic, but ordinarily the test for the completion of the iteration (this test can be modified by altering a shift order) is made upon only 34 most significant positions. At most 3 iterations are required for convergence.

6) The resulting square root of the original contents of the AMQ replaces the contents of the AMQ in integer form. The operation is completed.

SINE (Radian Arguments) SIN 52

1) Normalize the contents of the AMQ.

2) If the exponent of the AMQ exceeds 58, the machine will halt at the Error Halt location. If the Go button is pressed, then the status of this operation will be printed, and the control will proceed to execute the next interpretation cycle.

3)a. If the operation is SINE and if the true exponent ≤ -3, then the operation is completed.

b. If the operation is COSINE and if the true exponent ≤ -5, then replace the mantissa of the AMQ by 10^8 and the exponent by 51.

c. Otherwise, proceed to Step 4.

4) Convert the AMQ into fractional form.

5) Divide the absolute value of the AMQ by $\pi/4$. Call the integral part of this quotient I and the fractional part F. Ordinarily, the series test word is 2^{-39}, but if $I > 0$, then the test word is replaced by I for this operation. I (mod 8) determines the octant into which our argument is resolved. We compute the sine of F or $(\pi/4-F)$ or the cosine of F or $(\pi/4-F)$, depending upon the octant. F does not exceed $\pi/4$. Sine F or cosine F is computed using 39 binary bit arithmetic. The calculation is carried out by using a MacLaurin series expansion, and first and second differences are used to compute the coefficients. The test for convergence is ordinarily made on the leading 34 positions diminished by the series test word. A maximum of seven terms of the series is required for convergence when the argument equals $\pi/4$.

6) The resulting Sine (Cosine) of the original contents of the AMQ replaces the contents of the AMQ in the integer form.

7) a. If SD Mode, the operation is completed.

b. If N Mode, normalize the AMQ.

COSINE (Radian Arguments) COS 53

The operation is performed in almost the same manner as the Sine. The main difference between the two is that I + 2 (mod 8) determines the octant into which the argument is resolved in Step 5.

ARC TANGENT (Radians) ART 54

1) Normalize the contents of the AMQ.
2) If the true exponent < -3, then the operation is completed.
3) Convert the AMQ into fractional form.
4) a. If the AMQ exceeds "1" in absolute value, set y equal to the quotient of "1" divided by the AMQ.

b. Otherwise, set y equal to the AMQ.

5) If $y < \tan \pi/8$, compute $\tan^{-1}\left(\frac{y - \tan \pi/16}{1 + y \tan \pi/16}\right)$.

If $y \geq \tan \pi/8$, compute $\tan^{-1}\left(\frac{y - \tan 3\pi/16}{1 + y \tan 3\pi/16}\right)$.

In either case the absolute value of the argument does not exceed $\tan \pi/16$. The calculation is carried out with a MacLaurin's series expansion using 39 binary bit arithmetic. First differences are used to compute the coefficients. The test for convergence is ordinarily made on the leading 34 positions, although this test may be modified.

A maximum of eight terms of the series is required for convergence, when the absolute value of the argument equals tan $\pi/16$.

6) The resulting Arc Tangent of the original AMQ replaces the AMQ in the integer form.

7) a. If SD Mode, the operation is completed.

b. If N Mode, normalize the AMQ.

EXPONENTIAL (e^X) EXP 55

1) Normalize the AMQ.

2) Convert the AMQ into fractional form.

3) If the true exponent is less than -9, then replace the mantissa of the AMQ by 10^8 and the corresponding exponent by 51. The operation is completed.

4) If the true exponent exceeds 3, and if the mantissa is negative, then the Exponent Underflow condition exists; but if the mantissa is positive, the Exponent Overflow condition exists.

5) Denoting the contents of the AMQ by x, one may write $e^x = 10^{Mx}$, where $M = \log_{10} e$. $10^{Mx} = 10^{I+F} = 10^I \cdot 10^F$, where I is the integral part of 10^{Mx} and F is the corresponding fractional part. Set the exponent of the AMQ = I. Now, $10^F = e^{F+M}$. With $(F+M)\cdot 2^{-2} = y$ as the argument, compute $e^y - 1$ with a MacLaurin series expansion using 39 binary bit arithmetic. First differences are used to compute the coefficients. The test for convergence

is ordinarily made on the first 34 binary bits. A maximum of 12 terms is required for convergence when the argument equals $(1/4)\cdot 2^{-2}$ which is approximately equal to .576.

6) The resulting value of e^X replaces the AMQ in the integer form.

7) The remaining steps are the same as Steps 6 through 10 for the ADD operation.

LOGARITHM (Log_e) LOG 56

1) Normalize the AMQ.

2) If the mantissa of the AMQ is negative, the machine will halt at the Error Halt location. If the Go button is pressed, then the present state of the operation will be printed. Then the mantissa of the AMQ will be complemented. Proceed to Step 3.

3) Convert the AMQ into fractional form.

4) Denoting the fractional mantissa by A, find $X = 2^q \cdot A$ such that $1/\sqrt{2} < X < \sqrt{2}$. With $\frac{X-1}{X+1}$ as the argument, compute 1/2 ln X with a Taylor series expansion using 39 binary bit arithmetic. First differences are used to compute the series coefficients. The test for convergence is ordinarily made on the first 34 binary bits. A maximum of 7 terms of the series is required for convergence when the argument equals $\frac{\sqrt{2}-1}{\sqrt{2}+1}$.

5) The resulting $\log_e$ of the original contents of the AMQ replaces the AMQ in the integer form.

6) If the number of significant digits of the mantissa of the AMQ = 10, then replace the mantissa by the mantissa divided (unrounded) by 10, and increase the exponent of the AMQ by 1.

7) The remaining steps are the same as Steps 7 through 10 for the ADD operation.

B. Logical-Control Type Operations (Non-indexing)

Any of the operations to be discussed in this paragraph can be executed in either the Indexing Mode or the Non-indexing Mode. These operations constitute most of the "0" class and the "1" class of operations. The Input-Output type of operations which occur in these two classes will be discussed in a later paragraph.

1. The Zero Class of Operations

OPERATION	Oc-tal	Mne-monic	OPERATION	Oc-tal	Mne-monic
No Operation	00	---	Punch Data Cards	04	PCH
Transfer Neg. to Left	01	TNL	Transfer Neg. to Right	05	TNR
Transfer Plus to Left	02	TPL	Transfer Plus to Right	06	TPR
Transfer to the Left	03	TL	Transfer to the Right	07	TR

List of Zero Class Operations

NO OPERATION --- -- 00

Proceed to the next operation.

TRANSFER NEGATIVE TO LEFT TNL Y 01

TRANSFER NEGATIVE TO RIGHT TNR Y 05

If the mantissa of the AMQ is negative, then the left (right) operation of the instruction word stored in Y will be executed next. Otherwise, proceed to the next operation.

TRANSFER PLUS TO LEFT TPL Y 02

TRANSFER PLUS TO RIGHT TPR Y 06

If the mantissa of the AMQ is non-negative, then the left (right) operation of the instruction word stored in Y will be executed next. Otherwise, proceed to the next operation.

TRANSFER TO THE LEFT TL Y 03

TRANSFER TO THE RIGHT TR Y 07

The left (right) operation of the instruction word stored in Y will be executed next. (Note that this operation is the same as the JOHNNIAC operation 010 (014) with mnemonic code TRL (TRR).)

2. The One Class of Operations

OPERATION	Octal	Mnemonic		Octal	Mnemonic
Exit Interpreter Left	10	EXL	Exit Interpreter Right	14	EXR
Transfer Zero to Left	11	TZL	Transfer Zero to Right	15	TZR
	12			16	
Input Data Cards	13	INP	Print Data	17	PNT

List of One Class Operations

EXIT INTERPRETER TO LEFT	EXL	Y	10
EXIT INTERPRETER TO RIGHT	EXR	Y	14

The left (right) operation of the instruction word stored in Y, and all operations following this one and preceding re-entry into interpreter control will be executed as JOHNNIAC machine-language operations. (Note that the octal operation code for this operation is the same as that for the transfer to the left (right) JOHNNIAC operation.)

This operation cannot be executed in the Indexing Mode if the Exit Indicator is "1". In other words, the control cannot exit the Interpreter and the X Mode simultaneously.

TRANSFER ZERO TO LEFT	TZL	Y	11
TRANSFER ZERO TO RIGHT	TZR	Y	15

If the mantissa of the AMQ is 0, then the left (right) operation of the instruction word stored in Y will be executed next. Otherwise, proceed to the next operation.

C. Logical-Control Type Operations (Indexing)

All of the operations to be discussed in this paragraph must be executed in the Indexing Mode with the exception of the Enter Indexing Mode operation. These operations constitute the "7" class of operations.

OPERATION	Oc-tal	Mne-monic	OPERATION	Oc-tal	Mne-monic
Reset Add Index	70	RAX	Add Index	74	AX
Transfer on Negative Index	71	TNX		75	
Transfer on Positive Index	72	TPX		76	
Enter Indexing Mode	73	ENX		77	

List of Seven Class Operations

ENTER INDEXING MODE ENX 73

The operation in the next instruction word and all operations succeeding this one will be executed in the X Mode , until a "1" is encountered in the Exit Indicator field.

RESET ADD INDEX RAX 70

0	1 6	7 18	19 20	21	22 27	28 39
CON	7 0	X	Not Used	X Ind	X TAG	Δ X

The contents of the $X_{(\ )}$ and $\Delta X_{(\ )}$ of the Indexing Register specified by the X Tag are replaced by X and ΔX respectively. If either X or ΔX is negative, then the values must be expressed in complement form, i.e., $2^{12} - |X|$.

For example, if X_C is specified, then $X \longrightarrow X_C$ and $\Delta X \longrightarrow \Delta X_C$.

ADD INDEX AX 74

0	1 6	7 18	19 20	21	22 27	28 39
CON	7 4	X	Not Used	X Ind	X TAG	ΔX

The contents of the $X_{(\ )}$ and $\Delta X_{(\ )}$ for the Indexing

Registers specified by the X Tag are increased by X and ΔX respectively. If either X or ΔX is negative, then their values must be expressed in complement form, i.e., $2^{12} - |X|$. Also, the resulting values of $X_{(\ )}$, and $\Delta X_{(\ )}$ are retained modulo 2^{12}.

For example, if X_E is specified, then $X + X_E \pmod{2^{12}} \longrightarrow X_E$ and $\Delta X + \Delta X_E \pmod{2^{12}} \longrightarrow \Delta X_E$.

TRANSFER ON NEGATIVE INDEX TNX 71

0	1 6	7 18	19 20	21	22 27	28 39
CON	7 1	Y	Not Used	X Ind	X TAG	X_{MAX}

1) Denote the Indexing Register specified in the X Tag field by X_α. Then compute $X'_\alpha = X_\alpha + \Delta X_\alpha$.

2)a. If $X_{MAX} - X'_\alpha$ is negative, replace X_α by $X'_\alpha \pmod{2^{12}}$. Then execute the Left Operation of the instruction word stored in Y.

b. If $X_{MAX} - X'_\alpha$ is non-negative, then execute the left operation of the next instruction word.

This operation is designed for use with Indexing Registers which have negative $X_{(\ )}$ and $\Delta X_{(\ )}$ (in this case X_{MAX} must be in complement form). Observe that Y refers to a location in storage, whereas X_{MAX} does not. This fact should not be overlooked when the code is to be translated by an assembly program.

TRANSFER ON POSITIVE INDEX TPX 72

0	1 6	7 18	19 20	21	22 27	28 39
CON	7 2	Y	Not Used	X Ind	X TAG	X_{MAX}

1) Compute $X'_\alpha = X_\alpha + \Delta X_\alpha$.

2)a. If $X'_\alpha - X_{MAX}$ is negative, replace X_α by X'_α (mod 2^{12}). Then execute the Left Operation of the instruction word stored in Y.

b. If $X'_\alpha - X_{MAX}$ is non-negative, then execute the Left Operation of the next instruction word.

Observe that Y refers to a location in storage, whereas X_{MAX} does not.

The formation of loops can be accomplished by the use of the RAX operation, followed later in the code by the use of either the TPX or the TNX operation. The RAX operation sets the specified Indexing Register, say X_α, to the values X and ΔX. The TPX or TNX operation increases or decreases X_α by ΔX_α, and then tests the result against X_{MAX}. If $X_{MAX} = X + n \cdot \Delta X$, then the loop will be executed exactly n times. Ordinarily X = 0. In this case $X_{MAX} = n \cdot \Delta X$.

The addresses of the operations executed between the RAX and the TPX or the TNX can be modified at execution time (that is, the Effective address is computed and used) by the Indexing Registers specified in the X Tag. If the Effective addresses result in selecting particular elements of an array of data, then we can conceive of the X Tag positions which contain a "1" as being subscripts. For the

NX operations executed in the X Mode, if none of the X Tag positions contains a "1", then we shall say that these operations are non-subscripted. On the other hand if at least one of the X Tag positions contains a "1", we shall call these operations subscripted.

Example: Given the matrices A and B, compute the product matrix C = A x B. A is a 10x20 array which is stored by rows. The element A_{ij} is in storage location 1000+20i+j.

B is a 20x10 array which is stored by columns. The element b_{jk} is in storage location 2000+j+20k.

C is a 10x10 array which is to be computed and stored by rows. The element c_{ik} is to be in storage location 3000+10i+k, where $c_{ik} = \sum_{j} a_{ij} \times b_{jk}$.

A code for producing C is as follows:

LOC	LOP	LA	ROP	RA	COMMENTS
M.00	RA	M.00	010	F.00	Basic link to interpreter
M.01	ENX	--	--	--	Enter X Mode.
M.02	RAX	0000	040	0020	i = 0, Δi = 20
M.03	RAX	0000	004	0010	i'= 0,(Δi)' = 10
M.04	RAX	0000	010	0020	k = 0, Δk = 20
M.05	RAX	0000	001	0001	k' = 0, (Δk)' = 1
M.06	RAX	0000	020	0001	j = 0, Δj = 1
M.07	RA	K.00	000	0000	
M.08	ST	T.00	--	--	Set $C_{i'k'}$ = 0.
M.09	RA	(1000)	060	--	A_{ij} 1000 + 20 i+j
M.10	M	(2000)	030	0016	$A_{ij} \times b_{jk}$ 2000 + j + 20k
M.11	A	T.00	--	--	$+C_{i'k'}$
M.12	ST	T.00	--	--	$\longrightarrow C_{i'k'}$
M.13	TPX	M.09	020	0020	j < 20, j+1 $\longrightarrow$ j
M.14	RA	T.00	--	--	
M.15	ST	(3000)	005	0004	Ci'k' 3000 + 10i' + k'
M.16	AX	0001	001	0000	k' + 1 $\longrightarrow$ k'
M.17	TPX	M.06	010	0200	k < 200, k + 20 $\longrightarrow$ k
M.18	AX	0010	004	0000	i'+1 $\longrightarrow$ i'
M.19	TPX	M.04	040	0200	i < 200, i + 20 $\longrightarrow$ i
M.20	--	--	100	--	Exit X Mode
M.21	EXR	M.21	HTL	M.00	Exit Interpreter End
K.00	000	0000	000	0000	Constant - Zero
T.00	--	--	--	--	$C_{i'k'}$ Temporary

The interpretation time required to execute Non-indexing operations in the Non-indexing Mode is less (by approximately 10% for the "average" operation) than if the operations were executed in the X Mode. In addition, the storage requirement for the instruction words, which contain the operations, is doubled if the operations are executed in the X Mode. Then from the standpoint of both space and time, the operations can be executed more efficiently in the NX Mode. Therefore, if a reasonably long sequence of operations occurs after a RAX operation and before a TPX or TNX operation, and if all of these operations are non-subscripted, it may be desirable to exit the X Mode prior to this sequence of operations and re-enter the X Mode following this sequence of operations.

D. Input-Output Type Operations

The operations to be discussed in this paragraph cannot be executed in the Indexing Mode, nor can the operations be printed (traced). These operations occur among the "0" class and the "1" class of operations.

PUNCH DATA CARDS PCH 04

0	1 6	7 18	19 20	21	22-24	25-27	28 39
0	0 4	First Address	Not Used	A	0	C	Last Address

The packed floating-point numbers in the consecutive storage cells between the First Address and the Last Address inclusive will be punched, C words per card, into

IBM punched cards. If A = 1, a 12 punch will be placed in column 80 of the last card punched. The numbers will appear in consecutive positions of the cards in their explicit external form, beginning with column "9". A maximum of six words can be punched in a card, i.e., C $\leq$ 6. If a number is positive, the corresponding S column will contain a 12 punch. The cards will be punched according to the Floating-Point Data Card Format described below.

1 8	9	10-11	12 20	~	69	70-71	72 79	80
I D	S	EXP	MANTISSA	~	S	EXP		E F
							MANTISSA	

Floating-Point Data Card Format

The above format holds for both the PCH and the INP operations. Card columns 1-8 are not examined by the interpreter. If a number is negative, the corresponding S column must contain an "11" punch. If the number is positive, the corresponding S column may contain a 12 punch or it may be left blank. The EXP columns contain the true exponent increased by 50, and the columns labeled MANTISSA contain the mantissa as a proper fraction with the decimal point at the extreme left.

INPUT DATA CARDS INP 13

0	1 6	7 18	19-20	21	22-24	25-27	28 39
0	1 3	First Address	Not Used	A	0	C	Last Address

Floating-point numbers will be read from either the primary or secondary feed of the collator, C words per card, beginning with column "9", and will be stored in packed form in the consecutive storage cells between the First Address and the Last Address inclusive, unless a 12 punch in column 80 is encountered on a card before the Last Address is reached. In the latter event the address into which the last word (Cth word) from the card with the 12 punch in column 80 mark is stored will replace the Last Address of the INP instruction word in high-speed storage.

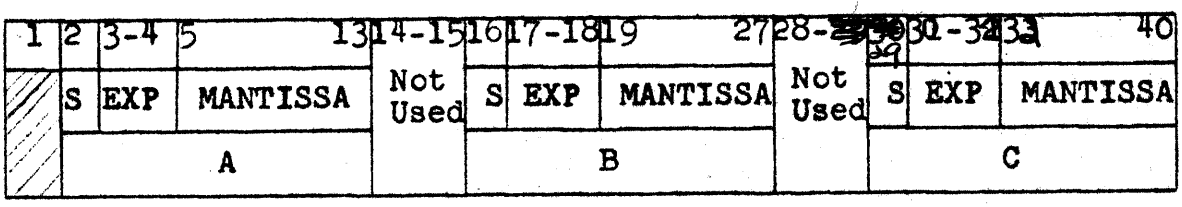
If A = 1, the secondary feed of the collator will be read. Otherwise, the primary feed will be read. A maximum of six words can be read from a card, i.e., $C \leq 6$. The cards to be read must be punched according to the Floating Point Data Card Format described above. A blank or a zero punch is interpreted as zero in the EXP and MANTISSA fields.

PRINT **PNT** 17

0	1 6	7 18	19-20	21	22-24	25-27	28 39
0	1 7	First Address	Not Used	A	B	C	Last Address

The packed floating-point numbers in the consecutive storage cells between the First Address and the Last Address inclusive will be printed by the ANelex Printer. The format for printing depends upon the information contained in the A, B, and C fields. A maximum of three numbers can be printed on one line. The positions on the paper into which these numbers can be printed line up vertically from line to

line and are called positions A, B, and C, starting from the left-hand side of the paper. Now, the presence of an octal digit "1" in any of the fields, A, B, and/or C, means that floating-point numbers can be printed in the corresponding positions A, B, and/or C on the paper. Any combination of these three positions can be specified by the INP instruction word. If all of the fields A,B, and C are zero, then the paper will be spaced as many lines as the number appearing in the Last Address field. In this event, the First Address is not interpreted, and no printing occurs.



40 Column ANelex Printer Print Format

S is represented by a "0" and "1" over-printed if the number is negative (⓪). Otherwise, S is represented by a blank. The EXP position contains the true exponent increased by 50, and the MANTISSA position contains a proper fraction with the decimal point at the extreme left.

V OPERATION

A. Tracing

Tracing is the name by which we shall call the process of selectively printing the results of the operations executed under interpreter control and the operations themselves. See p. 85 for the tracing formats.

Tracing is controlled first and foremost by the settings of the T_2 and T_3 switches. There are four mutually exclusive settings for these two switches, and corresponding to each there is a distinct type of tracing.

SWITCH SETTING	TYPE OF TRACING
T_2 Off T_3 Off	No Tracing
T_2 On T_3 Off	Breakpoint Tracing
T_2 Off T_3 On	Transfer Tracing
T_2 On T_3 On	All Orders Tracing

If T_2 is off and T_3 is off, then no orders will be traced unless the calculator has stopped at the Error Halt location.

If T_2 is on and T_3 is off, then no orders can be traced unless the Left Control field contains a "1" or the calculator has stopped at the Error Halt location. Note that the "1" in the Left Control field can cause either a Left Order or a Right Order or both to be traced; i.e. Breakpoint tracing can only be specified for an entire instruction word, and not for Left and Right orders separately.

If T_2 is off and T_3 is on, then no orders can be traced except those Transfer orders which result in a jump to the location specified in the address field (these orders include the TPX and TNX orders), unless the calculator has stopped at the Error Halt location.

If T_2 is on and T_3 is on, then all floating point orders can be traced upon execution except the Input-Output orders and the EXL and EXR orders.

If an order which is executed in the X Mode is traced, then two lines of information are printed. The first line of printing is the same as that for a Left operation executed in the NX Mode, except that the Left Address positions contain the Effective Address, if the order is a NX order.

One extra line is spaced on the paper just prior to tracing a Left Order.

1	4 5-6	7-9	10 11	14 15	16	17-18	19	27 28	29	30-31	32	40
LOC		LEFT OPER	LEFT ADDR.		S	EX	MANTISSA		S	EX	MANTISSA	
					CONTENTS AMQ				CONTENTS LEFT ADDR.			

Tracing Format for Tracing Left Order

The LOC, LEFT OPER., and LEFT ADDR. positions are printed as octal integers. The CONTENTS AMQ and CONTENTS LEFT ADDR. positions are printed in decimal in exactly the same manner as positions B and C for the PRINT operation. However, the CONTENTS LEFT ADDR. positions will be left blank for the "0", "1", and "7" classes of operations.

1 4	5	6	7 9	10 11 14	15	16 17-18	19 27	28	29	30-31	32 40
LOC	2		RT. OPER	RIGHT ADDR.		S	EX MANTISSA		S	EX	MANTISSA
						CONTENTS AMQ			CONTENTS RT. ADDR.		

Tracing Format for Tracing Right Order

The above description for tracing the Left order applies for tracing the Right order if, everywhere it occurs, "Left" is replaced by "Right". Note that this format holds true only for the NX Mode.

1 4	5	6	7	8-9	10	11-14	15 23	24 27	28 36	37 40
LOC	2		X IND	X TAG		RIGHT ADDR.	Not Used	CONTENTS $X_{(\ )}$	Not Used	CONTENTS $\Delta X_{(\ )}$

Tracing Format for Second Line (X Mode)

The LOC, X IND and TAG, and RIGHT ADDR. positions are printed as octal integers. The CONTENTS $X_{(\ )}$ and CONTENTS $\Delta X_{(\ )}$ positions are printed as decimal integers. If more than one Indexing Register is specified by the X Tag, only the contents of the Indexing Register corresponding to the right-most "1" in the X Tag will be printed.

In order to allow the user to make more complicated decisions concerning which orders shall be traced, traps may be set manually or by manchine-language code. There are three independent traps, one for each type of tracing, i.e., Breakpoint, All Orders, and Transfer types of tracing. One storage register (call it a Trap Register) is set aside within the interpreter for each of the three traps. Ordinarily, each of these cells contains a negative number. In this case, orders stored in all high-speed storage locations

can be traced if the other conditions are satisfied. In order to set a trap for a more restricted range of storage locations, the user must modify the contents of the Trap Register to suit his purpose. The Registers must be set in the following form:

0 6	7 18	19-20	21 27	28 39
000	LOWER BOUND	0	000	UPPER BOUND

Trap Register Layout

Denote the Lower Bound by LB and the Upper Bound by UB where LB and UB represent addresses. Then an order stored in location Y can be traced only if $LB \leq Y \leq UB$. In order to return the Trap Register to the condition where all locations can be traced, place any negative number into the register, or set LB = 0000 and UB = 4095_{10}.

If a trap has been set for a particular type of tracing and if at least one operation has been traced since the trap was set, then if H_3 is on, the calculator will halt upon execution of the first order which could be traced under this type of tracing and which has a location outside of the trap range. Essentially the user can (by placing H_3 on) effect an automatic halt following the completion of the tracing over a trap region.

Three levels of exclusion exist in which orders can be removed from the class of traceable orders. The first level uses the mutually exclusive settings of the T_2 and T_3 switches to reduce the class of traceable orders. The

second level consists of the Left Control indication. This level, of course, holds only for the Breakpoint type of tracing. The third level makes use of traps. The order in which the levels have been described is the same order in which the levels are examined by the interpreter for each individual operation. (The user need not be concerned with the order, however, except from the standpoint of efficiency.) The effective class of traceable orders is at any time just the logical intersection of the classes of traceable orders at each level.

B. Error Halt

The Error Halt Location is the location within the Interpreter to which control is transferred when an arithmetical inconsistency occurs during the execution of one of the Arithmetical types of operations. These inconsistencies have been discussed in Section IV, but we shall explain what will happen in the event that an Error Halt occurs with either the Exponent Overflow or the Exponent Underflow condition.

The Exponent Overflow condition is said to exist if the true exponent increased by 50 exceeds 99. Whenever the exponent exceeds 99, the exponent will be replaced by 99 and the calculator will halt at the Error Halt Location. If the Go button is pressed, the status of the operation causing the Exponent Overflow condition will be traced. Then the control will go to execute the next interpretation cycle.

The Exponent Underflow condition is said to exist if the true exponent increased by 50 becomes negative. Whenever the exponent becomes negative, the exponent will be replaced by zero and the calculator will halt at the Error Halt Location. If the Go button is pressed, the status of the operation causing the Exponent Underflow condition will be traced. Then the control will go to execute the next interpretation cycle.

C. Conversion of Data

Let X be any fixed-point number in the range $10^{-9} \leq X < 10^{9}$ which is to be converted into a floating-point number and stored in the packed internal form. Assume that $X \cdot 2^{q}$ is stored in L(X) as a proper fraction, where $-30 < q < 30$. Also assume that one of the integers 0, 1, ..., 9 has been stored in L (p) and that $X < 10^{p}$. Then $X \cdot 10^{9-p} < 10^{9}$.

The following machine-language code will result in placing X in L(F) in the packed floating-point form:

LOC	LOP	LAD	ROP	RAD	COMMENTS
α	LM	[L(X)]	M	$[L(10^{9-p})]$	$(X \cdot 2^{q}) \cdot 10^{9-p}$
α+1	LLH / LRH	[|q|] / [q]	ST	L(MAN)	LLH if q<0. LRH if q>0. $[(X) \cdot 2^{q}) \cdot 10^{9-p}] \cdot 2^{-q} \longrightarrow$ MAN
α+2	RA	L(50)	A	L(p)	PACK: 50+p = EXP.
α+3	LRC	0009	020	L(MAN)	
α+4	TPR	α+5	065	L(F)	MANTISSA < 0
α+5	TRL	α+6	064	L(F)	MANTISSA $\geq$ 0
α+6					

The integers 0, 1, ..., 9 and the integers 10^0, 10^1, ..., 10^9 are stored in consecutive locations within the interpreter. The integer 50 is also stored within the interpreter. The resulting number in L(F) can be normalized by adding the number to zero under interpreter control if the system is in the N Mode. Note that information can be lost by shifting if $X < 1$. In order to achieve a minimum loss of information a more sophisticated conversion method will be needed.

Let F be any floating-point number in the packed internal form which has the explicit exponent in the range $41 \leq EXP \leq 59$. Let us consider the problem of converting F into a fixed-point number.

The following machine-language code will result in placing $F \cdot 2^{-q}$ in L(X) as a fixed-point number if $EXP \geq 50$ and $F \cdot 2^{-q} < 1$. If $EXP < 50$ and $F \cdot 2^{q} < 1$, the code will result in placing $F \cdot 2^{q}$ in L(X) as a fixed-point number. In either case $0 \leq q < 30$.

LOC	LOP	LAD	ROP	RAD	COMMENTS
α+1	RAV	L(F)	CLC	10	UNPACK F
α+1	SRH	10	ST	L(MAN)	
α+2	STQ	L(EXP)	RA	L(F)	
α+3	TPR	α+4	021	L(MAN)	
α+4	ST	L(MAN)	RA	L(EXP)	
α+5	S	L(50)	TPL	α+13	EXP - 50 EXP ≥ 50
α+6	ST	TEMP	RS	TEMP	EXP < 50 50 - EXP
α+7	A	$L(10^0)$	SAR	α+16	$L(10^{50-EX})$
α+8	RA	L(MAN)	DS	$L(10^9)$	MAN + 10^9
α+9	STQ	TEMP	TPL	α+11	
α+10	RS	L(1)	TRR	α+11	
α+11	ZTA	---	LLH	q	$(MAN + 10^9)\cdot 2^q$
α+12	TRR	α+16			
α+13	ST	TEMP	RA	L(9)	
α+14	S	TEMP	A	$L(10^0)$	9-(EXP-50)
α+15	SAR	α+16	RA	L(MAN)	$L(10^{9-(EXP-50)})$
α+16	LRH	|q|	D	[ ]	$MAN\cdot 2^{-q}$ + 10^K
α+17	STQ	L(X)			

Note that information can be lost by truncation if EXP < 50. A more involved conversion method will be required to control the loss of information.

VI APPENDIX

A. Data Forms

0	1 2	3 11
S	a* + 50	A*

External Form of Data

A* is a positive 9-digit decimal fraction with the decimal point at the extreme left, a* is the true exponent, and S is the sign associated with the mantissa.

0	1 9	10 39
0	a* + 50	$A^* \cdot 10^9$

Internal Packed Form of Data

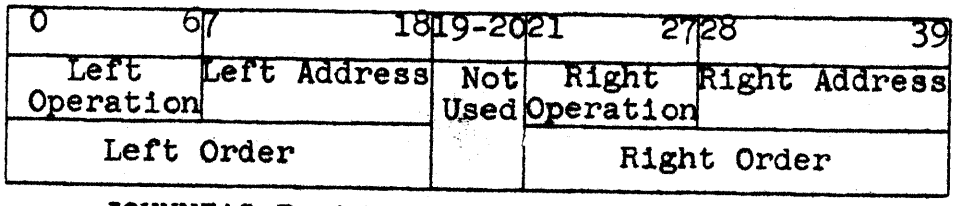
The above representation holds true if the mantissa is non-negative. If the mantissa is negative, the entire word is complemented.

1 8	9	10-11	12 20	21	22-23	24 32	69	70-71	72 80
ID	S	EXP	MANTISSA	S	EXP	MANTISSA	S	EXP	MANTISSA

Floating-Point Data Card Format

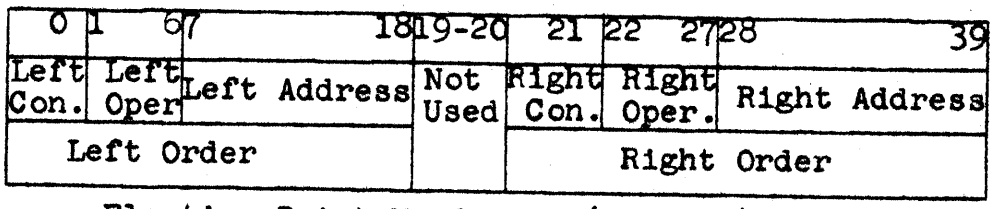
Columns 1 - 9 are not interpreted. The S positions must contain an "11" punch if the corresponding mantissa is negative. If the mantissa is positive the S position may contain a "12" punch or be left blank. A "12" punch in column 80 indicates an end-of-file condition for the INP operation. From one to six consecutive data words can be punched into one card beginning with column 9.

B. Instruction Word Forms



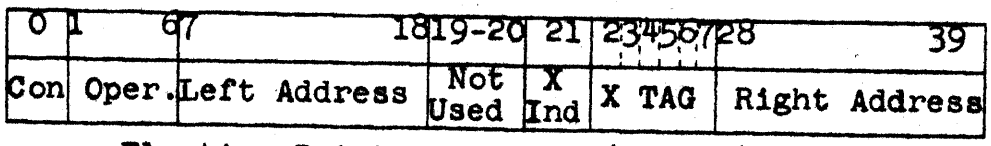
JOHNNIAC Instruction Word Form

The interpretation cycle is Fetch, Left Right.



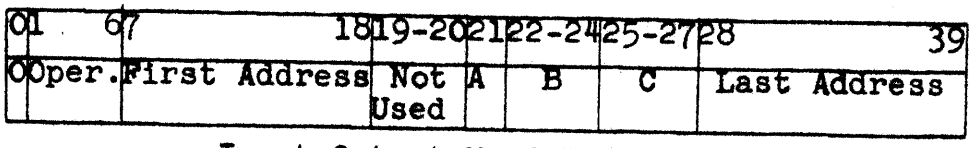
Floating-Point Word Form (NX Mode)

This form holds for all floating-point operations which are executed in the NX Mode except for the Input-Output type of operations. The interpretation cycle is Fetch, Left, Right.



Floating-Point Word Form (X Mode)

The interpretation cycle is Fetch, Left.



Input-Output Word Form

The interpretation cycle is Fetch, Left.

C. Switch Settings

T_1 Off		N MODE
T_1 On		SD MODE
T_2 Off	T_3 Off	NO TRACING
T_2 On	T_3 Off	BREAKPOINT TRACING
T_2 Off	T_3 On	TRANSFER TRACING
T_2 On	T_3 On	ALL ORDERS TRACING
H_3 On		POSSIBLE HALT DURING TRAP TRACING
H_3 Off		NO HALT DURING TRAP TRACING

D. Operations Which Differ Significantly from JOHNNIAC Operations

PUNCH DATA CARDS PCH 04

01	67	18	19-20	21	22-24	25-27	28 39
0	04	First Address	Not Used	A	0	C	Last Address

The floating point numbers stored in the locations between the First Address and the Last Address inclusive will be punched into cards, C words per card. The cards are punched in the Floating-Point Data Card form. If the A field contains a "1", a 12 punch will be punched into column 80 of the last card punched for the order. If the punch in column 80 is not desired, A should be zero. See p. 62 for a more exhaustive discussion of the PCH order.

TRANSFER TO THE LEFT	TL	Y	03
TRANSFER TO THE RIGHT	TR	Y	07

These operations are executed in a manner analogous to the TRL and TRR (010 and 014) JOHNNIAC operations. See p. 55 for a more exhaustive discussion of the TL and TR orders.

EXIT INTERPRETER TO LEFT	EXL	Y	10
EXIT INTERPRETER TO RIGHT	EXR	Y	14

The Left (Right) order of location Y will be executed next as a JOHNNIAC operation. All succeeding orders will be executed outside the Interpreter control until the control is reentered by the standard basic linkage. (Note that the octal operation code for this operation is the same as that for the transfer to the left (right) JOHNNIAC operation). These orders cannot be traced. See p. 56 for a further discussion of the EXL and EXR orders.

TRANSFER ZERO TO LEFT	TZL	Y	11
TRANSFER ZERO TO RIGHT	TZR	Y	15

If the mantissa of the AMQ is zero, the Left (Right) order of location Y will be executed next. Otherwise, proceed to the next operation. See p. 56 for a more exhaustive discussion of the TZL and TZR orders.

INPUT DATA CARDS	INP		13

0	1 6	7 18	19-20	21	22-24	25-27	28 39
0	13	First Address	Not Used	A	0	C	Last Address

Floating-point numbers are read from cards punched in the Floating-Point Data Card form, C words per card, and stored in locations between the First Address and the Last Address inclusive. However, if a 12 punch in column 80 appears on a card before the Last Address is reached, then no more cards will be read and the Last Address field of the instruction word will be replaced by the last address into which a number has been stored. If the A field contains "0", the primary feed will be selected. Otherwise, the secondary field will be selected. See p. 63 for a more exhaustive discussion of the INP order.

PRINT PNT 17

0	1 6	7 18	19-20	21	22-24	25-27	28 39
0	17	First Address	Not Used	A	B	C	Last Address

The floating-point numbers stored in the locations between the First Address and the Last Address inclusive will be printed. The presence of a "1" in any of the fields A, B, or C indicates that numbers can be printed in the corresponding print positions. If all of the fields A, B, and C are zero, then the paper will be spaced as many times as the number appearing in the Last Address field. See p. 64 for a more exhaustive discussion of the PNT order.

SQUARE ROOT SQR -- 51

Denote the contents of the AMQ by X. This operation computes $\sqrt{X}$ for $0 \leq X < 10^{50}$. If $X < 0$, the calculator will halt at the Error Halt location. If the Go button is pressed, the operation will be printed and $\sqrt{|X|}$ will be computed. See p. 48 for a more exhaustive discussion of the SQR order.

SINE SIN -- 52

Denote the contents of the AMQ by X, where X is in radians. This operation computes sin X for $0 \leq |X| \ddot{<} 10^{9}$. If $|X| \ddot{>} 10^{9}$, the calculator will halt at the Error Halt location. If the Go button is pressed, the operation will be printed. See p. 49 for a more exhaustive discussion of the SIN order.

COSINE COS -- 53

Denote the contents of the AMQ by X, where X is in radians. This operation computes cos X for $0 \leq |X| \ddot{<} 10^{9}$. If $|X| \ddot{>} 10^{9}$, the calculator will halt at the Error Halt location. If the Go button is pressed, the operation will be printed. See p. 51 for a more exhaustive discussion of the COS order.

ARC TANGENT ART -- 54

Denote the contents of the AMQ by X. This operation computes arc tan X in radians for $X \leq |X| < 10^{50}$. See p.51.

for a more exhaustive discussion of the ART order.

EXPONENTIAL EXP -- 55

Denote the contents of the AMQ by x. This operation computes e^x for all x such that $10^{-50} < e^x < 10^{50}$.

If $e^x > 10^{50}$ or $e^x < 10^{-50}$, then the calculator will halt at the Error Halt location. If the Go button is pressed, the operation will be printed.

See p. 52 for a more exhaustive discussion of the EXP order.

LOGARITHM LOG -- 56

Denote the contents of the AMQ by X. This operation computes $\log_e X$ for $0 < X < 10^{50}$. If $X \leq 0$, the calculator will halt at the Error Halt location. If the Go button is pressed, the operation will be printed. See p. 53 for a more exhaustive discussion of the LOG order.

RESET ADD INDEX RAX 70

0	1 6	7 18	19-20	21	23 4 5 6 7	28 39
CON	70	X	Not Used	X Ind	X TAG	ΔX

$X \longrightarrow X_{(\ )}$ and $\Delta X \longrightarrow \Delta X_{(\ )}$.

The control must be in the X Mode when executing this order. If the X IND field contains a "1", the control will exit from the X Mode following the execution of the order. See p. 57 for a more exhaustive discussion of the RAX order.

TRANSFER NEGATIVE INDEX TNX Y 71

0	1 6	7 18	19-20	21	23456 7	28 39
CON	71	Y	Not Used	X Ind	X TAG	X MAX

Compute $X_{(\ )} + \Delta X = X'_{(\ )}$. If $X_{MAX} - X'_{(\ )} < 0$, $X'_{(\ )} \longrightarrow X$ and the control will jump to Y. If $X_{MAX} - X'_{(\ )} \geq 0$, proceed to the next operation.

The control must be in the X Mode when executing this operation. If the X IND field contains a "1", the control will exit from the X Mode following the execution of this order. See p. 58 for a more exhaustive discussion of the TNX order.

TRANSFER POSITIVE INDEX TPX Y 72

0	1 6	7 18	19-20	21	23456 7	28 39
CON	72	Y	Not Used	X Ind	X TAG	X_{MAX}

Compute $X_{(\ )} + \Delta X = X'_{(\ )}$. If $X'_{(\ )} - X_{MAX} < 0$, $X'_{(\ )} \longrightarrow X_{(\ )}$ and the control will jump to Y. If $X'_{(\ )} - X_{MAX} \geq 0$, proceed to the next operation.

The control must be in the X Mode when executing this operation. If the X IND field contains a "1", the control will exit from the X Mode following the execution of this order. See p. 59 for a more exhaustive discussion of the TPX order.

ENTER INDEXING MODE ENX -- 73

Upon execution of this operation all succeeding operations will be executed in the X Mode until a "1" is encountered in the X IND field. This operation can be executed in the X Mode or the NX Mode. See p. ↶ for a more exhaustive discussion of the ENX order.

ADD INDEX AX -- 74

0	1 6	7 18	19-20	21	23456728 39
CON	74	X	Not Used	X Ind	X TAG ΔX

$X + X_{(\ )} \longrightarrow X_{(\ )}$ and $\Delta X + \Delta X_{(\ )} \longrightarrow \Delta X_{(\ )}$.

The control must be in the X Mode when executing this order. If the X IND field contains a "1", the control will exit from the X Mode following execution of this order. See p. ↶ for a more exhaustive discussion of the AX order.

E.

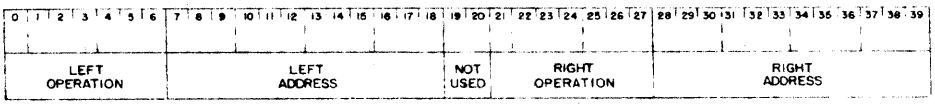
LIST OF JOHNNIAC OPERATIONS – MARCH 16, 1955

Operation Codes are Base Eight Numbers. Notes are Base Ten

000		Proceed to next order in sequence	004	LM	Clear MQ, M → MQ
001	TNL	If A < 0, c → left command in M	005	TNR	If A < 0, c → right command in M
002	TPL	If A ≥ 0, c → left command in M	006	TPR	If A ≥ 0, c → right command in M
003	TFL	If overflow, c → left command in M	007	TFR	If overflow, c → right command in M
010	TRL	c → left command in M	014	TRR	c → right command in M
011	T1L	If T_1 on, c → left command in M	015	T1R	If T_1 on, c → right command in M
012	T2L	If T_2 on, c → left command in M	016	T2R	If T_2 on, c → right command in M
013	T3L	If T_3 on, c → left command in M	017	T3R	If T_3 on, c → right command in M
020	RA	Clear A, M → A	024	A	M + A → A
021	RS	Clear A, − M → A	025	S	− M + A → A
022	RAV	Clear A, | M | → A	026	AV	| M | + A → A
023	RSV	Clear A, −| M | → A	027	SV	−| M | + A → A
030	MR	Clear A, M · MQ rounded → A	034	MB	M · MQ + 2^{-39} [A + ½(1−$_1$ A $_1$)] → A and MQ
031	MNR	Clear A, −M · MQ rounded → A	035	MNB	−M · MQ + 2^{-39} [A + ½(1−$_1$ A $_1$)] → A and MQ
032	M	Clear A, M · MQ → A and MQ	036	MA	M · MQ + 2^{-39} A → A and MQ
033	MN	Clear A, −M · MQ → A and MQ	037	MNA	−M · MQ + 2^{-39} A → A and MQ
040	DS	A ÷ M → MQ, r → A	044	D	(A + 2^{-39} MQ) ÷ M → MQ, r → A
041	DNS	A ÷(− M)→ MQ, r → A	045	DN	(A + 2^{-39} MQ) ÷(− M)→ MQ, r → A
050	ST	A → M	054	SAB	$_7$ A $_{19}$ and $_{28}$ A $_{39}$ → $_7$ M $_{19}$ and $_{28}$ M $_{39}$
051	SOL	$_0$ A $_8$ → $_0$ M $_8$	055	SOR	$_{20}$ A $_{27}$ → $_{20}$ M $_{27}$
052	SAL	$_7$ A $_{19}$ → $_7$ M $_{19}$	056	SAR	$_{28}$ A $_{39}$ → $_{28}$ M $_{39}$
053	SHL	$_0$ A $_{19}$ → $_0$ M $_{19}$	057	SHR	$_{20}$ A $_{39}$ → $_{20}$ M $_{39}$
060	STQ	Clear A, MQ → A and M	064	AQS	MQ + A → A and M
061	SNQ	Clear A, − MQ → A and M	065	SQS	− MQ + A → A and M
062	SVQ	Clear A, | MQ | → A and M	066	AVS	| MQ | + A → A and M
063	SNV	Clear A, −| MQ | → A and M	067	SVS	−| MQ | + A → A and M
070	SRC	Clear MQ, shift A right n places Zeros into A_0.	074	SRH	Shift A right n places. Zeros into A_0.
071	CLC	Clear MQ, circular shift of A and MQ left n places. Couple MQ_0 to A_{39}, A_0 to MQ_{39}.	075	CLH	Circular shift of A and MQ left n places. Couple MQ_0 to A_{39}; A_0 to MQ_{39}.
072	LRC	Clear MQ, power shift A and MQ right n places. Couple A_{39} to MQ_1. $_0$ A $_0$ to → $_0$ MQ $_0$.	076	LRH	Power shift A and MQ right n places. Couple A_{39} to MQ_1. $_0$ A $_0$ → $_0$ MQ $_0$.
073	LLC	Clear MQ, power shift A and MQ left n places. Couple zeros into MQ_{39}; MQ_1 to A_{39}.	077	LLH	Power shift A and MQ left n places. Couple zeros into MQ_{39}, MQ_1 to A_{39}.
100	SEL	Select input outout Address part of 100 XXX0 Pri. Feed Reader XXX1 Sec. Feed Reader XXX2 Feed Punch XXX3 Feed Punch and echo XXX4 Select Printer and space XXX5 Select Printer	104	DIS	Display
			105	HUT	Hoot
			106	EJ	Eject page
101	C	Copy			
110	RD	Read drum words to M and memory addresses following numerically. Denoting MQ as xxx $f_1f_2f_3f_4$ dpb $l_1l_2l_3l_4$, the f's determine the first drum address and the l's the last drum address. d selects the drum; p, the position of the heads; and b, the band to be read.			
111	WD	Read M and words in memory addresses following numerically to drum. MQ has the same significance as in 110.			
120	ZTA	Clear A to Zero	124	PI	M I A → A
121		Clear A	125	NI	− M I A → A (− denotes digit inversion of M).
122		Clear A	126	PMI	| M |I A → A
123		Clear A	127	NMI	−| M |I A → A
130	HTL	Halt c → left command in M	134	HTR	Halt c → right command in M
131	H1L	Halt if H_1 on; c → left command in M	135	H1R	Halt if H_1 on; c → right command in M
132	H2L	Halt if H_2 on; c → left command in M	136	H2R	Halt if H_2 on; c → right command in M
133	H3L	Halt if H_3 on; c → left command in M	137	H3R	Halt if H_3 on; c → right command in M
140					
150					
160					
170					

DEFINITIONS

A	Accumulator
MQ	Multiplier Quotient Register
M	Word in the M^{th} address in Internal Storage
$_{20}A_{39}$	Digits in position 2^{-20} through 2^{-39} of the word in A.
c →	Control goes to
I	Logical (digit by digit) product or Intersection.



JOHNNIAC INSTRUCTION WORD LAYOUT

F. List of Floating-Point Operations

OPERATION	Octal	Mnemonic	OPERATION	Octal	Mnemonic
No Operation	00	---	Punch Data Cards	04	PCH
Transfer Neg. to Left	01	TNL	Transfer Neg. to Right	05	TNR
Transfer Plus to Left	02	TPL	Transfer Plus to Right	06	TPR
Transfer to the Left	03	TL	Transfer to the Right	07	TR
Exit Interpreter Left	10	EXL	Exit Interpreter Right	14	EXR
Transfer Zero to Left	11	TZL	Transfer Zero to Right	15	TZR
	12			16	
Input Data Cards	13	INP	Print Data	17	PNT
Reset Add	20	RA	Add	24	A
Reset Subtract	21	RS	Subtract	25	S
Reset Add Absolute Value	22	RAV	Add Absolute Value	26	AV
Reset Sub. Absolute Value	23	RSV	Subtract Absolute Value	27	SV
	30			34	
	31			35	
Multiply	32	M		36	
Multiply Negatively	33	MN		37	
Divide	40	DS		44	
Divide Negatively	41	DNS		45	
	42			46	
	43			47	
Store	50	ST	Arc Tangent	54	ART
Square Root	51	SQR	Exponential	55	EXP
Sine	52	SIN	Logarithm	56	LOG
Cosine	53	COS		57	
	60			64	
	61			65	
	62			66	
	63			67	
Reset Add Index	70	RAX	Add Index	74	AX
Transfer on Neg. Index	71	TNX		75	
Transfer on Pos. Index	72	TPX		76	
Enter Indexing Mode	73	ENX		77	

40 COL. ANELEX PRINTER

JOHNNIAC FLOATING POINT SYSTEM

X - DECIMAL DIGIT
Y - OCTAL DIGIT
S { BLANK - PLUS ; ⊖ - MINUS

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40

		FORMAT FOR:
L O C	L O P L A D C (AMQ) C (MEM)	
Y Y Y Y	Y Y Y Y Y Y Y S X X X X X X X X X X S X X X X X X X X X X	TRACING LEFT ORDER
L O C	R O P R A D C (AMQ) C (MEM)	
Y Y Y Y 2	Y Y Y Y Y Y Y S X X X X X X X X X X S X X X X X X X X X X	TRACING RIGHT ORDER NON-INDEXING
L O C	T A G R A D X( ) ΔX( )	
Y Y Y Y 2	Y Y Y Y Y Y Y X X X X X X X X	TRACING RIGHT ORDER INDEXING

ONE EXTRA LINE IS SPACED PRIOR TO PRINTING INFORMATION FOR THE LEFT ORDER.

	A	B	C	
	C (α)	C (α + 1)	C (α + 2)	
S X X	X X X X X X X X X X	S X X X X X X X X X X	S X X X X X X X X X X	PRINTING WITH TAG OR ROP = $l_A l_B l_C$

SENSE SWITCH SETTINGS FOR TRACING

T_2 Off	T_3 Off	NO TRACING
T_2 On	T_3 Off	BREAKPOINT
T_2 Off	T_3 On	TRANSFER
T_2 On	T_3 On	FLOATING POINT (All Orders)

Left Margin of Paper
Column Marker
Manually Set
Column Marker
Manually Set
Column Marker
Manually Set
Right Margin of Paper