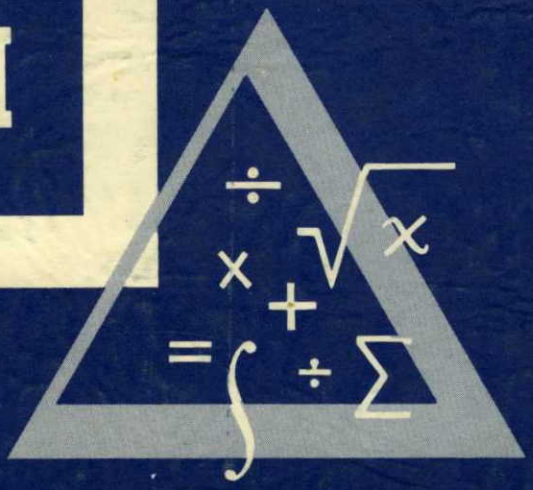
IBM



Technical Newsletter No. 8

APPLIED SCIENCE DIVISION

IBM

APPLIED SCIENCE DIVISION

Technical Newsletter No. 8

September 1954

This issue of the Newsletter is devoted solely to articles on the IBM Type 650 Magnetic Drum Data Processing Machine. Three of the articles were written by George R. Trimble, Jr., and Elmer C. Kubie who were members of the IBM Mathematical Planning Group at Endicott, N. Y. Mr. Trimble is also the author of the remaining three articles. All the routines and programs presented in this issue have been tested and checked out on the Type 650 in Endicott.

CONTENTS

PRINCIPLES OF OPTIMUM PROGRAMMING THE IBM TYPE 650

G. R. Trimble, Jr.
E. C. Kubie

Introduction

The IBM Type 650 is a parallel-serial calculator utilizing a magnetic drum for storage. It has been designed with "ease of use" as one of the primary considerations. The programmer is not burdened with timing restrictions which he must always keep in mind. Interlocks make it impossible to violate timing conditions in such a way as to cause the machine to give erroneous results. If, however, proper recognition and analysis of the sequence of events occurring within the machine is made, one can significantly increase the overall speed by properly locating data and instructions. Optimum programming is the technique by which data and instructions are located in such a manner as to minimize or eliminate, if possible, non-productive waiting or searching time.

The basic cycle of the 650 is the "word time" or time required to read one word. As there are 50 words written around the drum, each word time is equal to 1/50 of a drum revolution. Since the drum revolves at 12,500 revolutions per minute a word time is equal to .096 milliseconds. Each operation can be analyzed in terms of word times. The object here is to analyze each operation to see how many word times are required in its interpretation and execution. In this manner the basic or fundamental word times are determined and from these a set of rules is derived through which optimum programming may be effected.

Angular drum locations are all that must be considered when optimum coding. The 650 is completely synchronous so that the words located in equivalent angular positions are in phase and are equivalent from a timing viewpoint. Thus, the words in addresses n modulo 50 are equivalent, so that, for example, addresses 0003, 0053, 0103, ..., 1903, 1953 are in phase and therefore, equivalent so far as optimum programming is concerned. This means that there are 40 different locations along the length of the drum, any of which may be used without loss of time.

The lower half of the accumulator can be read into or out of only during an even word time and the upper half of the accumulator only during an odd word time. Thus, when the operation called for is one which uses the accumulator, it may be necessary to wait for an even word time or an odd word time in order to obtain the

data. This is why the cycles 'wait for even', or 'wait for odd' on the sequence chart, are required.

The D-I address system used in the 650 greatly facilitates optimum programming. Data and instructions may be in any locations on the drum and instructions may be taken in any desired sequence. As mentioned above, once the optimum location has been determined there are 40 different locations in which the datum or instruction could be placed since the 40 bands are in phase.

It is also significant that the address of many instructions will be optimum within a range of angular locations. This is due to the 650's ability to overlap arithmetic execution with search for the next instruction.

The characteristics of the 650 which facilitate optimum programming are,

1. It is synchronous, that is, all timing is controlled from timing pulses on the drum.

2. Interlocks make it impossible to cause errors by violation of timing conditions.

3. Equivalence of angular locations in the 40 bands.

4. Parallel operation, that is, simultaneous execution of operation and search for next instruction.

5. The D-I address system which makes possible flexible location of data and instructions.

Sequence Chart

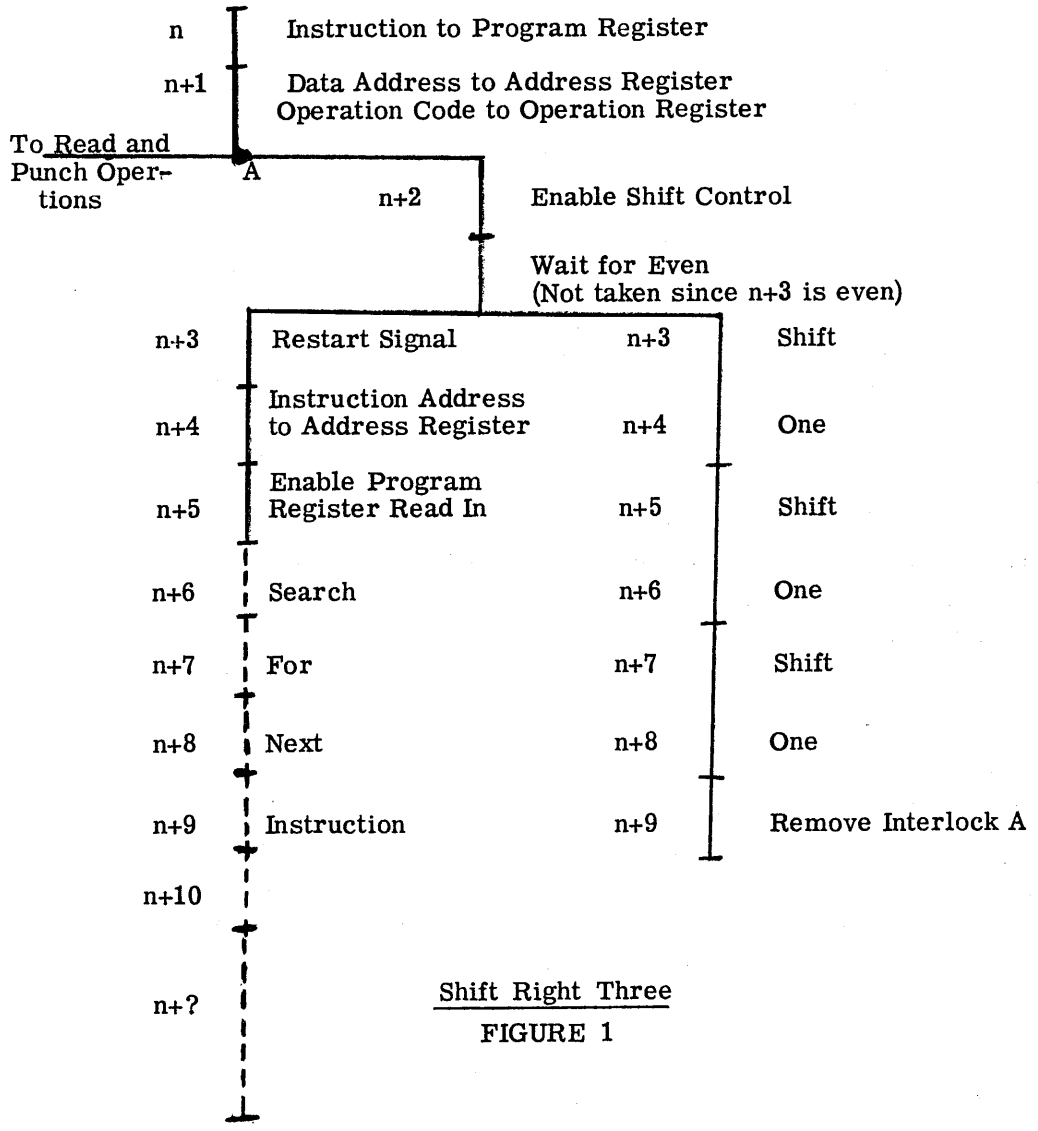
The sequence chart shows the steps taken in the interpretation and execution of each operation. It is drawn in segments which are usually equivalent to one word time. Where events are shown in parallel, they are performed simultaneously. Most operations branch into parallel execution shortly after the data is made available to the arithmetic unit. One branch indicates the arithmetic process and the other the obtaining of the next instruction.

Analyzing the steps carried out by the machine in the performance of any operation, one sees that in every case certain fundamental word times are required. Begin consideration of each instruction at the time when the instruction has been located but has not entered the program register. Starting at this point, the first word time of every operation is used to transfer the instruction from its memory location to the program register. The next word time is used to initiate the interpretation of this instruction. This is done by transferring the data address to the address register and the operation code to the operation register. The steps performed during the word times beyond this point will depend upon the particular instruction being executed.

Consider the shift right 3 operation. The portion of the sequence chart for this particular operation is shown in Figure 1. Assume that the shift instruction is in location n and that n is odd. During word time n, the instruction is read

into the program register. During word time n+1 the operation code (30) is transferred to the operation register and the data address (0003) is transferred to the address register. Word time n+2 "Enable Shift Control", is used to set up the necessary control circuits to perform the shifting. Since n+3 is even, the next cycle, "Wait for Even," is skipped and shifting begins immediately. At this point parallel operation begins and the actual shifting process on the right branch occurs simultaneously with the process of searching for and obtaining the next instruction indicated on the left branch.

The restart signal which occurs during word time n+3 indicates that the data address in the address register is no longer needed. During word time n+4 the instruction address is transferred to the address register. Word time n+5 is the "Enable Program Register Read In" cycle which indicates that a new instruction is to be read into the program register. Thus, beginning with word time n+6 the 650 starts searching for the next instruction. This search may require from 0 word times to as many as 49 word times depending on the location of the next instruction.



Shift Right Three

FIGURE 1

Examining the arithmetic branch it is seen that during word times n+3 through n+8 the actual shifting takes place. The interlock at point A is removed during word time n+9. The function of this interlock will be explained later.

Although the arithmetic portion of the operation is not completed until word time n+9, the program register is able to accept the next instruction at word time n+6. If the next instruction were placed in a location corresponding to word times n+6, n+7, or n+8, it would be read into the program register before completion of the arithmetic portion of the shift operation. In addition to this, interpretation of this instruction would begin in that the operation code would be transferred to the operation register and the data address transferred to the address register. Thus, the first two cycles of the interpretation of the next instruction occur in parallel with the execution of the shift instruction. At this point, interpretation must cease since further steps may make use of control circuits or portions of the arithmetic unit which are already in use (unless it is a read or punch instruction). The interlock at A is provided to insure that interpretation of the next instruction does not proceed past this point.

If the next instruction were placed in a location corresponding to word time n+9, the instruction would be read into the program register during completion of the arithmetic portion of the shift operation. In this case only the first cycle of the interpretation of the next instruction occurs in parallel with the execution of the shift instruction.

If the next instruction were placed in a location corresponding to word time n+10 it would be read into the program register immediately upon completion of the shift operation.

Each of the locations corresponding to word times n+6 through n+10 effectively reduces the search time to 0 since there is no wait time between completion of the shifting operation and the obtaining of the next instruction. However, if the next instruction were placed in a location corresponding to word times n+6 through n+9, one cycle of the interpretation of the next instruction would take place. If it were restricted to a location corresponding to word times n+6 through n+8, two cycles of interpretation of the next instruction would take place.

Thus, not only can the search for the next instruction take place during completion of the arithmetic portion of an operation, but also interpretation of this next instruction can begin. If the next instruction calls for a read or punch operation, execution of either of these operations can proceed since they do not require use of the arithmetic unit. In this case the interlock A does not stop interpretation of the next instruction after the first 2 cycles but it will be noted that read and punch instructions bypass this interlock.

As a numerical example assume n=643, that is, the instruction is in location 0643. The instruction address should be 0649, 0650, or 0651 to reduce the access time to zero. Of course, locations 0699, 0700, 0701, etc., are equally good.

There are three classes of operations. They are read, punch and arithmetic operations. Correspondingly, there are three interlocks. The read interlock "R" stops execution of the following read instructions, until completion of the previous read instruction. If the following instructions are punch or arithmetic operations however, they may proceed without delay. The punch interlock "P" stops execution of the following punch instructions, until completion of the previous punch instruction. If the following instructions are read or arithmetic operations however, they may proceed without delay. The arithmetic interlock "A" stops execution of the following arithmetic instructions until completion of the previous arithmetic operation. If the following instructions are read or punch instructions however, they may proceed without delay. Thus simultaneous reading, punching, and computing is possible. However, simultaneous execution of two arithmetic operations, two punching operations or two reading operations is not possible.

Referring to the shift right three example again, it is seen that since read or punch instructions are not held up by the interlock at A, they should be placed in a location corresponding to word time n+6 (649 in the numerical example). If the next instruction is another arithmetic operation, there is no advantage to n+6 or n+7 over n+8, since the interlock at A causes the machine to wait for completion of the shifting process.

The rules developed for optimum coding are based on an effective step being executed during every word time. This assumption is made to keep the rules simple. For this reason, it is best that the next instruction be placed in a location corresponding to word time n+8 (651 in the numerical example) if it is another arithmetic operation. This allows maximum overlap and assures an effective step for every word time (no waiting at interlock points).

Because of the above considerations the rules for shift instructions are stated in terms of 2 limits. The lower limit is best if the next instruction calls for a read or punch operation. The upper limit is best if the next instruction is an arithmetic operation. If the upper limit is not convenient, any location within the two limits will result in zero access time. However, when considering the succeeding instructions and their data or instruction addresses, one should apply the rules as if the upper limit were used. With this in mind, the following rules apply for the shift right 3 instruction.

Shift Right Three	Location of Next Instruction	
	Lower Limit	Upper Limit
n even	i=n+7	i=n+9
n odd	i=n+6	i=n+8

This example demonstrates how the sequence chart is used to determine optimum

locations. It would be too lengthy a task to show how every rule is obtained from the sequence chart, so we will state the rules and leave the proofs for those readers sufficiently curious.

Rules for Optimum Programming

n : location of instruction
d : data address of instruction
i : instruction address of instruction
i_1: lower limit of i
i_u: upper limit of i
a : location of argument found by table look up operation
Σm : sum of multiplier digits
Σq : sum of quotient digits

Operation	n even d=n+	n odd d=n+	d even i=d+	d odd i=d+
Add, subtract, etc., 10, 11, 15, 16, 17, 18, 60, 61, 65, 66, 67, 68	3	3	5	4
Multiply, 19	3	3	21+2Σm	20+2Σm
Divide, 14, 64	3	3	61+2Σq	60+2Σq
Store Lower Accumulator, 20	5	4	3	3
Store Upper Accumulator, 21	4	5	3	3
Load Dist., Store Dist., 69, 24	3	3	3	3
Store D Address, Store I Address, 22, 23	3	4	3	3

Operation	n even d=n+	n odd d=n+	a even i=a+	a odd i=a+
Table Look Up, 84	3	3	5	6

Operation	n even d=n+	n odd d=n+	n even i=n+	n odd i=n+
Branch Non-Zero Upper, 44	3	4	4	5
Branch Non-Zero, 45	4	3	5	4
Branch Minus, 46	3	3	4	4
Branch Overflow, Branch Dist. 8, 47 91-98	3	3	5	5
Branch Dist. 8, 90, 99	4	4	5	5
No Operation, Stop, 00, 01	-	-	4	4

Shift Operations	No. of Positions	n even		n odd	
		i_1=n+	i_u=n+	i_1=n+	i_u=n+
Shift Right, 30	(0)	6	6	5	5
Shift Left, 35	(1)	7	7	6	6
Shift and Count, 36	(2)	7	7	6	6
	(3)	7	9	6	8
	(4)	7	11	6	10
	(5)	7	13	6	12
	(6)	7	15	6	14
	(7)	7	17	6	16
	(8)	7	19	6	18
	(9)	7	21	6	20
Shift and Count Only, 36	(10)	7	23	6	22
Shift and Round, 31	(1)	7	7	6	6
	(2)	7	9	6	8
	(3)	7	11	6	10
	(4)	7	13	6	12
	(5)	7	15	6	14
	(6)	7	17	6	16
	(7)	7	19	6	18
	(8)	7	21	6	20
	(9)	7	23	6	22
	(10)	7	25	6	24

800X Addresses

Since the storage entry switches (8000), distributor (8001), lower accumulator (8002) and upper accumulator (8003) are addressable, special consideration must be given to the cases where one of these locations is addressed. The storage entry switches and distributor are always immediately accessible. However, the lower accumulator can be read into or out of only during an even word time. The upper accumulator can be read into or out of only during an odd word time. Thus, for purposes of optimum coding the storage entry switches and the distributor may be treated as being equivalent to any address, the lower accumulator as equivalent to an even address and the upper accumulator as equivalent to an odd address.

If, for example, a reset add upper operation (or any add type operation) has a data address of 8000 or 8001, these addresses can be treated as being equivalent to n+3 and the instruction address of that instruction can be determined accordingly. If a data address of 8002 were used, and the instruction were in an even location, the optimum location for the datum for that instruction would be n+3, which is odd in this case. Therefore, an extra cycle must be taken to wait for an even location so that the lower accumulator may be read out. Since the effective data address is then even, the rule i=d+5 must be used to determine the location of the next instruction. Similar analyses may be made for each of the other cases.

The following table gives the rules for determining the instruction address of an instruction in cases in which a data address of 800X is used.

Operation	Data Address	n even i=n+	n odd i=n+
Add, Subtract, etc., (10, 11, 15, 16, 17, 18, 60, 61, 65, 66, 67, 68)	8000	7	8
	8001	7	8
	8002	9	8
	8003	7	8
Load Distributor, 69	8000	6	6
	8001	6	6
	8002	7	6
	8003	6	7

Since the accumulator may be in use when an instruction address of 8002 or 8003 is given, it would seem to be possible to take the next instruction from the accumulator before the arithmetic operation was completed. For example, if an 8002 instruction address were used on a multiply operation, one of the partial products might be taken as the next instruction rather than the final product. For this reason, an additional interlock is provided to prevent the next instruction from being taken from an 800X address until completion of the arithmetic portion of the operation. This prevents the above from occurring.

The following table gives the rules for determing the effective instruction address in those cases in which an instruction address of 800X is used.

Operation	Inst. Address	d even i~d+	d odd i~d+
Add, Subtract, etc. (10,11,15,16,17,18, 60,61,65,66,67,68) No Complement Cycle Required	8000 8001 8002 8003	5 5 6 5	4 4 5 4
Add, Subtract, etc., (10,11,15,16,17,18, 60,61,65,66,67,68) Complement Cycle Required	8000 8001 8002 8003	7 7 8 7	6 6 7 6
Load Distributor, 69	8000 8001 8002 8003	3 3 4 3	3 3 3 4

~ means "equivalent to"

By application of the principles used in determining the above rules it is possible to determine equivalent addresses for any other desired case. It is only necessary to remember that an 8002 address is equivalent to an even drum address and 8003 is equivalent to an odd drum address.

Consider the instruction 1006438000. Since 0643 is odd, the rule in d+4 must be used. Therefore, the instruction at 8000 can be programmed as though it were located in location 0647.

Techniques for Using Optimum Programming

The gains obtained by optimum programming will depend upon the particular problem considered and the skill of the programmer. Every instruction cannot be optimum as conflict will occasionally exist between instructions. For example, data placed optimumly for a store operation may not be optimumly located for a later add operation. Furthermore, an instruction which is preceded by several branch instructions cannot usually be optimumly located for each of these branch instructions.

Techniques for efficient use of optimum programming will be developed through experience. It is too lengthy a task to thoroughly explore techniques for its use in this paper. However, certain general principles can be applied to any program one may wish to consider. Some of these principles will be indicated in the following paragraphs.

The first consideration that must be made is when should optimum programming be used.

Any problem in which the speed of input or output is appreciably reduced due to lengthy calculations, can justifiably be programmed optimumly.

Once it has been determined that optimum programming is necessary, there are two ways in which it can be done. One could simply program the problem optimumly by straightforward application of the rules derived above without consideration of the

program as a whole. Such a procedure may result in data and instructions being poorly located for the latter part of the problem. This type of programming can be done quickly and with very little more difficulty than if the instructions were programmed in sequence. Even though it is done roughly, it will usually result in a significant increase in the overall speed, and will be well worth the small amount of additional effort required.

The second type of optimum programming is where the programming is done elegantly. This type of programming requires a greater amount of work and thought on the part of the programmer. Instead of simply programming each step optimumly as it occurs, the programmer must think ahead to see how this might possibly affect later steps which will use the same data or perhaps branch to the same instruction. Many possibilities exist when programming in this manner. One can see after he has completed the problem how a simple re-arrangement of the beginning may improve the latter part of the problem. It may be necessary to re-program the same problem several times in order to obtain the most efficient program. Obviously such a procedure would require much more time than "sequential" programming or the "rough" optimum programming described above.

Elegant optimum programming should be used only when it is desirable to use the 650 in the most efficient manner possible. The types of problems for which this is necessary are those which must be done over and over on a mass basis and which realize reduced input-output speeds. It will not always be necessary to program the entire problem this way but only those segments or sub-routines which are most frequently used. For example, sub-routines, such as floating decimal operations, square root, sine, and cosine evaluations, which occur frequently in technical applications, should be programmed to function in the most efficient manner possible. Similar applications which will occur in commercial problems are extension from gross to net in payroll calculation, insurance dividend calculation, and bill computation in public utility customer accounting.

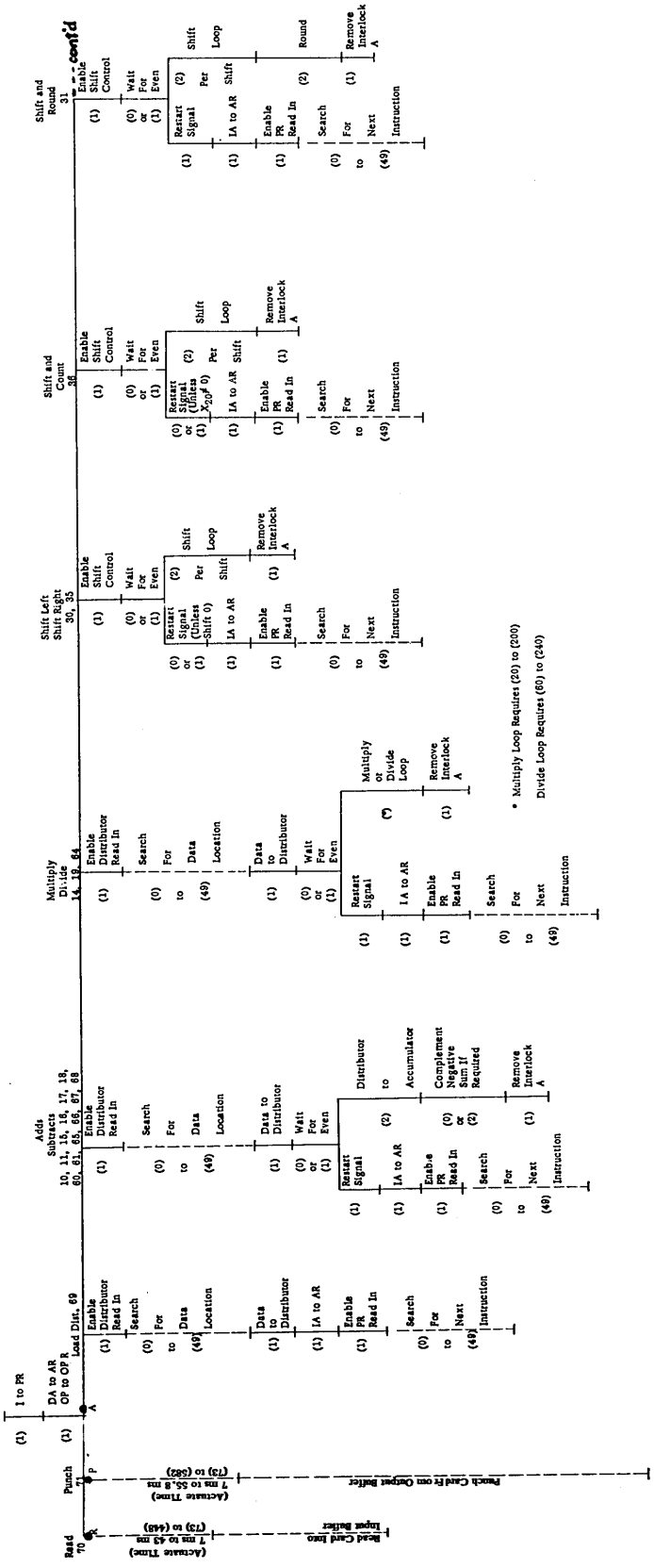
Example

The following example of programming for square root is included to help clarify how optimum programming is used. When programmed sequentially, this routine requires approximately .321 seconds (assuming 5 iterations). Secondly is shown the same routine coded optimumly. The time required is reduced to .152 seconds. Thus, the increase in speed is a factor of 2.1 to 1. Note, however, that the optimumly programmed routine requires 2 additional storage locations so that the constant 1/2 will always be located optimumly. Experience to date has shown that the gain realized may be a factor of less than 2 to 1 to a factor as large as 6 or 7 to 1.

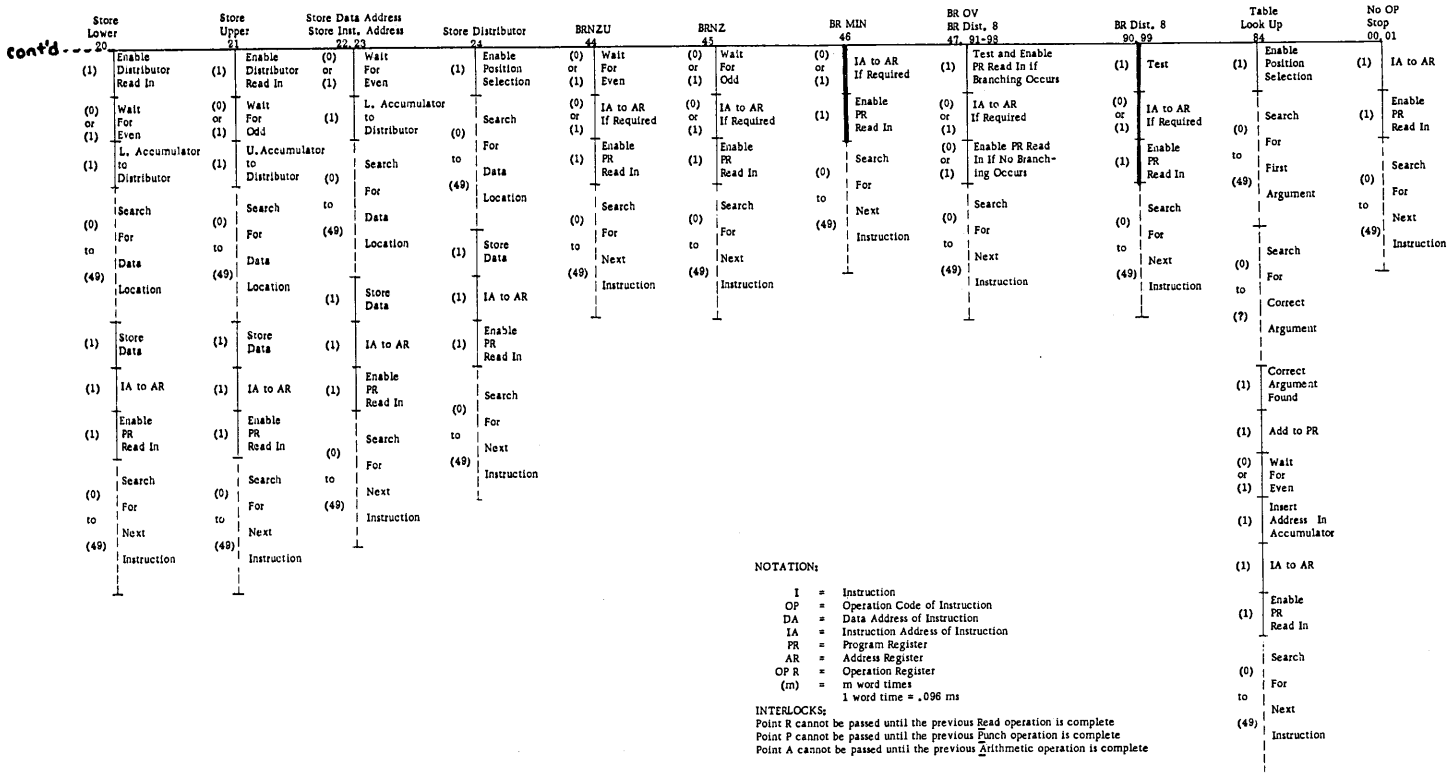
Computing During Input and Output Operations

At a card punching rate of 100 cards per minute up to 544 milliseconds are available for computing. This is approximately 5600 word times or 110 drum revolutions. At a card reading speed of 200 cards per minute up to 257 milliseconds are available for computing. (This is approximately 2700 word times or 54 drum revolutions.)

IBM TYPE 650 SEQUENCE CHART



cont'd



NOTATION:

I	=	Instruction
OP	=	Operation Code of Instruction
DA	=	Data Address of Instruction
IA	=	Instruction Address of Instruction
PR	=	Program Register
AR	=	Address Register
OP R	=	Operation Register
(m)	=	m word times
		1 word time = .096 ms

INTERLOCKS:
Point R cannot be passed until the previous Read operation is complete
Point P cannot be passed until the previous Punch operation is complete
Point A cannot be passed until the previous Arithmetic operation is complete

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: Square Root WRITTEN BY:

$a = \sqrt{A}$, $0 < A < 1$

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
		SEQUENTIAL PROGRAMMING REQUIRES .321 SEC.			
0102	RAL	65	0200	0103	
0103	AU	10	8001	0104	$a_0 = (A + 1)/2$
0104	MULT	19	0201	0105	
0105	ST U	21	0202	0106	
0106	RAU	60	0200	0107	
0107	MULT	19	0201	0108	
0108	DIV RU	64	0202	0109	$a_{n+1} = (a_n + A/a_n)/2$
0109	AU	10	0200	0110	
0110	MULT	19	0202	0111	
0111	ST U	21	0203	0112	
0112	SU	11	0202	0113	Test $a_{n+1} - a_n$. If $a_{n+1} - a_n \geq 0$,
0113	BRNZU	44	0114	0101	transfer back to main routine (0101).
0114	BR MIN	46	0115	0101	
0115	LD	69	0203	0116	$a_{n+1} - a_n < 0$, thus, prepare to repeat
0116	ST D	24	0202	0106	iteration
0200:	1/2 =	.5000000000			
0201:	A				
0202:	a_n				
0203:	Temporary Storage				
0101:	Contains next instruction in main routine.				
		OPTIMUM PROGRAMMING REQUIRES .152 SEC.			
0139	RAL	65	0142	0148	
0148	AU	10	8001	0105	$a_0 = (A + 1)/2$
0105	MULT	19	0108	0189	
0189	ST U	21	0144	0147	
0147	RAU	60	0100	0155	
0155	MULT	19	0108	0140	
0140	DIV RU	64	0144	0134	$a_{n+1} = (a_n + A/a_n)/2$
0134	AU	10	0137	0141	
0141	MULT	19	0144	0125	
0125	ST U	21	0130	0133	
0133	SU	11	0144	0120	Test $a_{n+1} - a_n$. If $a_{n+1} - a_n \geq 0$,
0120	BRNZU	44	0124	0128	transfer back to main routine (0128).
0124	BRMIN	46	0127	0128	
0127	LD	69	0130	0135	$a_{n+1} - a_n < 0$, thus, prepare to
0135	ST D	24	0144	0147	repeat iteration.
0142:	1/2 =	.5000000000			
0108:	A				
0144:	a_n				
0100:	1/2 =	.5000000000			
0137:	1/2 =	.5000000000			
0130:	Temporary Storage				
0128:	Contains next instruction in main routine.				

PAGE ____ OF ____

AN INTERPRETATIVE FLOATING DECIMAL SYSTEM FOR THE IBM TYPE 650

G. R. Trimble, Jr.

E. C. Kubie

Introduction

This floating decimal system will perform 18 basic operations using a floating decimal number system by means of interpretative programming. It was designed with coding convenience as a prime objective. This is illustrated by the fact that one of the basic operations is essentially a vector by vector multiplication.

Floating decimal instructions, that is, instructions which are to be interpreted, have a negative sign. A floating decimal instruction consists of a two digit operation code, a 4 digit address specifying the location of the first factor, a 4 digit address specifying the location of the second factor (if required) and another 4 digit address specifying the location of the third factor (if required). It is a variable address system in that only as many addresses as are needed for the particular operation are required. Thus, some operations use only one address, some require two addresses and others require 3 addresses.

Floating decimal instructions will be taken from consecutive memory locations. If an instruction requires one or two addresses, the instruction is stored in one memory location. If three addresses are required, the third address is stored in the memory location immediately following the one containing the instruction.

Should a positive instruction appear, it is interpreted as a normal 650 instruction and subsequent instructions are not interpreted. Thus, upon occurrence of a positive instruction the 650 operates in its usual D-I mode. This continues until an instruction address of 0026 is given which causes control to return to the interpretative routine. The program will return to the floating decimal mode of operation at the point of departure.

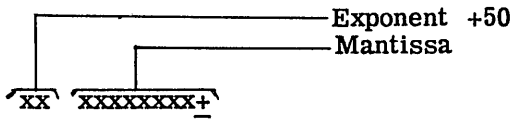
For example, consider the following sequence:

Location	Contents
n	floating point instruction (-)
n+1	floating point instruction (-)
n+2	floating point instruction (-)
n+3	normal 650 instruction (+)

(The contents of n+3 are interpreted as a normal 650 instruction including the instruction address for sequencing. Normal execution of instructions continues until 0026 is used as an instruction address, at which time the program returns to the floating decimal mode of operation beginning with the instruction in location n+4).

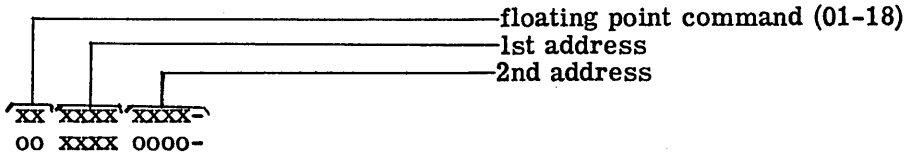
Number Form

The number form is as follows:



Instruction Form

The instruction form is as follows:



i+1 has this form only if the third address is needed, otherwise, the form of i+1 is the same as the form of i.

Floating Decimal Accumulator

A floating point accumulator referred to as K is used for accumulation and accumulative multiplication. It does not need to be addressed in operations which make use of it. For example, operation 01 (A+B ⟶ K) uses two addresses, A and B. The sum is automatically stored in K as the result of this operation.

Operations

Below is a list of the operations with their codes, addresses required and estimated average time of execution.

Code and Addresses	Operation	Estimated Average Time
01, A, B	A+B⟶K	62.8
02, A	A+K⟶K	54.7
03, A, B	A-B⟶K	63.1
04, A	K-A⟶K	59.5
05, A, B, C	A+B⟶C	81.8

Code and Addresses	Operation	Estimated Average Time
06, A, B, C	$A-B \longrightarrow C$	96.2
07, A, B, C	$AxB \longrightarrow C$	84.8
08, A, B	$AxB \longrightarrow K$	78.4
09, A, B, C	$A/B \longrightarrow C$	92.2
10, A, B	$A/B \longrightarrow K$	72.2
11, A, B	BR MIN A	28.4
12, A	BR	18.7
13, A, B	BRNZ A	30.8
14, A, B	$(AxB)+K \rightarrow K$	102.5
15, A, B	$K-(AxB) \rightarrow K$	102.5
16, A, B	$\sqrt{A} \longrightarrow B$	157.9
17, A, B, C	$\sqrt{A^2+B^2+C^2} \longrightarrow K$	416.3
18, A_1, B_1, n	$\sum_{i=1}^{n} A_i B_i \rightarrow K$	(32.4+92.3n)

Explanation of Programs

1. General Interpretation

The general interpretation routine takes instructions from consecutive memory locations and analyzes them to see if they are normal 650 instructions or if they must be interpreted. If the instruction is a normal 650 instruction, it is executed as such. If the instruction is to be interpreted, the routine obtains the factor at address A and stores it in location 0037. Control is then transferred to the proper sub-routine. The constant in location 0193 facilitates use of the translating routine TR1. The amount of translation is placed in the instruction address positions of this constant. This amount is then added to the operation code and control is transferred to the translated sub-routine.

2. Addition Sub-Routine

The addition sub-routine adds the factors in locations 0037, in which A is stored, and 0057, which is the floating decimal accumulator K. The result is normally stored in the floating decimal accumulator K.

3. Multiplication Sub-Routine

The multiplication sub-routine multiplies the factors in 0037, which is A, and 0089, in which B is placed. The product is normally stored in 0037.

4. Division Sub-Routine

The division sub-routine divides the factor in 0037 which is A, by the factor B, which had previously been placed in the lower accumulator. The result is normally stored in 0057 which is the floating decimal accumulator.

The following sub-routines interpret the second and third addresses if required, obtain the necessary factors and modify the addition, multiplication and division sub-routines so as to store the result in the desired location. In some cases, they also modify portions of other sub-interpretative routines.

5. Interpretation of 01

This sub-interpretative routine obtains the factor B and transfers control to the addition sub-routine.

6. Interpretation of 02

Since the factors are already in place all that is necessary for this sub-interpretative routine is to transfer control to the addition sub-routine.

7. Interpretation of 03

This sub-interpretative routine obtains the factor B, reverses its sign and transfers control to the addition sub-routine.

8. Interpretation of 04

The sign of the factor A is reversed and control is transferred to the addition sub-routine.

9. Interpretation of 05

Since the addition sub-routine makes use of the floating decimal accumulator K, the contents of K must be temporarily transferred to another location, and returned after the operation has been completed. This sub-interpretative routine obtains B, modifies the addition sub-routine to store the result in location C and transfers control to the addition sub-routine. After completion of the operation, control is returned to the sub-interpretative routine and the addition sub-routine is restored to normal.

10. Interpretation of 06

This sub-interpretative routine modifies the sub-interpretative routine, "Interpretation of 05" so that the sign of factor B is reversed.

11. Interpretation of 07

This sub-interpretative routine obtains the factor B, modifies the multiply sub-routine to store the product in location C and transfers control to the multiply sub-routine. After completion of the multiplication, control is returned to this sub-interpretative routine and the multiply sub-routine is restored to normal.

12 Interpretation of 08

The factor B is obtained, the last instruction of the multiply sub-routine is modified to store the product in K and control is transferred to the multiply sub-routine. After completion of the multiplication, control is returned to the sub-interpretative routine "Interpretation of 07" which restores the last instruction of the multiply sub-routine to normal.

13. Interpretation of 09

The factor B is obtained, the last instruction of the divide sub-routine is modified to store the quotient in C and control is transferred to the divide sub-routine. After completion of the division, control is returned to this sub-interpretative routine and the last instruction of the divide sub-routine is restored to normal.

14. Interpretation of 10

The factor B is obtained and control is transferred to the divide sub-routine.

15. Interpretation of 11

The sign of the factor in location A is examined. If the sign is minus, control is transferred to location B by modifying the general interpretation routine. Sequencing of instructions then continues in the normal fashion beginning with B. If the factor in location A is positive or its mantissa is zero, sequencing of instructions continues in the normal fashion.

16. Interpretation of 12

The general interpretation routine is modified so that the next instruction is taken from location A. Sequencing of instructions then continues in the normal fashion beginning with A.

17. Interpretation of 13

The factor in A is examined and if its mantissa is not zero, the general interpretation routine is modified to take the next instruction from location B. Sequencing of instructions then continues in the normal fashion beginning with B. If the factor in A is zero, sequencing of instructions continues in the normal manner.

18. Interpretation of 14

The factor B is obtained, the last instruction in the multiply sub-routine is modified to store the product in 0037, which is A, and control is transferred to the multiply sub-routine. After completion of the multiplication, control is returned to the sub-interpretative routine, the last instruction of the multiply sub-routine is restored to normal and control is transferred to the addition sub-routine which computes the desired sum.

19. Interpretation of 15

The sign of the factor B is reversed and control is transferred to the sub-interpretative routine "Interpretation of 14".

20. Interpretation of 16

This sub-interpretative routine computes the required square root and stores it in location B. The initial approximation is $X_0 = (1+4A)/(4+A)$, obtained from the RAND Corporation "Approximations in Numerical Analysis" form 15S, Notes.

21. Interpretation of 17

Factor B is obtained and temporarily stored in 0265. Factor A is squared and stored in 0057 by the multiplication sub-routine. Factor B is then squared and added to the square of A. Factor C is then obtained, squared and added to $A^2 + B^2$. Finally, control is transferred to the square root sub-routine and $\sqrt{A^2+B^2+C^2}$ is computed and stored in 0057 which is K.

22. Interpretation of 18

Operation 18 is a vector by vector multiplication. Address A_1 and address B_1 are the addresses of the first elements of each of the vectors. Succeeding elements of the vectors are then taken from consecutive memory locations starting with the initial locations A_1 and B_1. The third address, n, indicates the number of elements in each vector.

Use is made of the sub-interpretative routine "Interpretation of 14" to perform accumulative multiplication. As each pair of factors are multiplied together and added to the previous sum, n is reduced by 1 and zero tested. When n has been reduced to zero, the last two factors have been multiplied and the operation is complete.

Storage

This system requires 467 storage locations. It uses locations 0000 through 0465. Every location within this block is used, thus making it easy to incorporate this program with other programs. The translating routine, TR1 may be used to translate this program to any desired block of memory locations. It should be translated by an even amount however to preserve the even-odd conditions.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: General Interpretation WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0026	RAU	60	0029	0092	
0092	AU	10	0099	0118	Increase i to i+1
0118	ST U	21	0029	8001	
8001	RAL	65	i+1	0439	Pick up instruction i+1
0439	BRMIN	46	0025	8002	Is instruction i+1 to be interpreted?
0025	LD	69	0028	0031	
0031	ST DA	22	0035	8001	Put data at address A in location
8001	LD	69	A	0455	0037.
0455	SLT	35	0002	0334	
0334	ST D	24	0037	0040	
0040	AU	10	0193	8003	Transfer to proper sub-routine.
8003	NO OP	00	0000	00fc	
0029	RAL	65	i	0439	
0099		00	0001	0000	Constants
0028	LD	69	0000	0455	
0193		00	0000	0000	

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: (A) + (K) → K Subroutine WRITTEN BY: _____

(A) = $a_1 A_1$, (K)= $a_2 A_2$, a=xx, A= x. xxxxxxx+

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0054	RAL	65	0057	0061	0057: K
0061	SLT	35	0002	0067	
0067	STU	21	0073	0027	$a_1 \rightarrow$ 0073
0027	STL	20	0081	0034	$A_1 \rightarrow$ 0081
0034	RAL	65	0037	0041	0037: A
0041	SLT	35	0002	0048	
0048	STU	21	0052	0055	$a_2 \rightarrow$ 0052
0055	STL	20	0059	0062	$A_2 \rightarrow$ 0059
0062	RAABL	67	8003	0069	
0069	SABL	18	0073	0077	$\lvert a_2 \rvert - \lvert a_1 \rvert$. Modify SRD instruction
0077	SLT	35	0004	0087	for case where $0 < \lvert a_2 - a_1 \rvert < 10$
0087	LD	69	0090	0043	
0043	ST DA	22	0098	0051	
0051	BRNZ	45	0104	0056	Is $a_2 = a_1$?
0056	RAL	65	0059	0078	$a_2 = a_1$. Thus $A_3^1 = A_2 + A_1$
0078	AL	15	0081	0038	
0038	BRNZU	44	0091	0044	Is $\lvert A_3^1 \rvert \geq 10$?
0091	SRD	31	0003	0101	$\lvert A_3^1 \rvert \geq 10$. Thus $A_3 = 0.1\ A_3^1$
0101	SLT	35	0002	0058	
0058	BR MIN	46	0111	0112	Is $A_3 < 0$?
0111	SL	16	0065	0070	$A_3 < 0$. Increase exponent by 1
0070	SABL	18	0073	0127	
0127	SLT	35	0008	0045	Put exponent in normal positions
0045	AL	15	8003	0053	(2 high order digit positions)
0112	AL	15	0065	0120	$A_3 > 0$. Increase exponent by 1
0120	AABL	17	0073	0127	
0044	BRNZ	45	0049	0053	$\lvert A_3^1 \rvert < 10$. Is $A_3^1 = 0$?
0049	RAU	60	8002	0066	
0066	SCT	36	0000	0088	$0 < \lvert A_3^1 \rvert < 10$. Thus A_3 equals A_3^1
0088	SU	11	8002	0105	shifted and counted.
0105	RAL	65	8003	0116	
0116	BR MIN	46	0070	0120	Is $A_3 < 0$?
0104	LD	69	0057	0060	
0060	SRT	30	0005	0074	$a_2 \neq a_1$. Is $\lvert a_2 - a_1 \rvert \geq 10$?
0074	BRNZ	45	0080	0079	
0080	BR MIN	46	0033	0084	$\lvert a_2 - a_1 \rvert \geq 10$. Is $a_2 > a_1$?
0033	RAL	65	8001	0053	$a_1 \geq a_2 + 10$. Thus $a_3\ A_3 = a_1\ A_1$
0084	RAL	65	0037	0053	$a_2 \geq a_1 + 10$. Thus $a_3\ A_3 = a_2\ A_2$
0079	BR MIN	46	0032	0039	$0 < \lvert a_2 - a_1 \rvert < 10$. Is $a_2 > a_1$?
0032	RAL	65	0059	0064	
0064	SRT	30	0002	0072	$a_1 > a_2$. Thus $A_3^1 = A_1$ + shifted A_2
0072	LD	69	0081	0098	
0098	SRD	31	(0000)	0071	
0071	SLT	35	0002	0030	Compute A_3^1
0030	AL	15	8001	0038	
0039	LD	69	0052	0068	
0068	ST D	24	0073	0076	
0076	RAL	65	0081	0036	$a_2 > a_1$. Thus $A_3^1 = A_2$ + shifted A_1
0036	SRT	30	0002	0046	Also, use a_2 to compute a_3
0046	LD	69	0059	0098	

PROBLEM: (A) + (K) ⟶ K Subroutine WRITTEN BY: _____

(A) = a_1 A_1, (K) = a_2 A_2 a = xx, A = x.xxxxxxx±

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0053	ST L	20	0057	0026	Store sum in K
0090	: 31 0000 0071				Constants
0065	: 00 0000 0001				

PROBLEM: (A) x (B) ⟶ A Subroutine WRITTEN BY: _____

(A) = m_1 M_1 , (B) = m_2 M_2 , m=xx, M=x.xxxxxxx ±

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0134	RAL	65	0037	0141	0037: A. Separate exponent and
0141	SLT	35	0002	0097	mantissa
0097	STL	20	0151	0106	M_1 ⟶ 0151 M_1 =x.xxxxxxxxoo
0106	RAU	60	8003	0063	
0063	SRT	30	0002	0119	m_1 - 50
0119	RAABL	67	8002	0128	
0128	SL	16	0131	0085	
0085	AU	10	0089	0093	0089: B. Test to see if M_2<0
0093	BR MIN	46	0096	0147	
0096	SU	11	8001	0103	
0103	SL	16	8001	0161	$M_2 < 0$
0161	SLT	35	0002	0117	Put M_2 in upper accumulator and
0117	AU	10	8002	0075	$M_3' = m_1 + m_2$-50 in lower accumulator.
0075	RSU	61	8003	0138	Acc: ooMMMMMMMM/mmoooooooo
0138	SRT	30	0002	0095	
0147	SU	11	8001	0155	
0155	AL	15	8001	0113	$M_2 \geq 0$.
0113	SLT	35	0002	0121	Put M_2 in upper accumulator and
0121	AU	10	8002	0115	$m_3^1 = m_1 + m_2$ -50 in lower accumulator
0115	RAU	60	8003	0138	
0095	MPY	19	0151	0082	$M_3' = M_1 \times M_2$
0082	BR MIN	46	0135	0086	
0135	SL	16	0139	0094	
0086	AL	15	0139	0094	Round M_3^1 in 7th decimal place
0094	LD	69	8003	0100	
0100	SLT	35	0002	0107	
0107	SRT	30	0001	0163	Test to see if M_3^1 is of the form
0163	SCT	36	0001	0170	ox.xxxxxxxx or xx.xxxxxxx
0170	BROV	47	0123	0126	
0123	SRT	30	0009	0181	
0181	SRD	31	0002	0102	
0102	ST DA	22	0205	0108	
0108	ST IA	23	0211	0114	M_3^1 is of the form ox.xxxxxxxx
0114	RAL	65	8001	0171	Round M_3' in 8th decimal place
0171	BR MIN	46	0124	0125	to get M_3.
0124	AL	15	0178	0083	
0125	SL	16	0178	0083	
0083	ST L	20	0037	0026	
0126	RAL	65	8001	0042	M_3^1 is of the form xx.xxxxxx. Thus,
0042	BR MIN	46	0000	0050	$m_3 = m_3 + 1$
0000	SL	16	0109	0083	
0050	AL	15	0109	0083	
0131	: 5000000000				Constants
0139	: 5000000000				
0178	: 0000000005				
0109	: 0100000000				

* In floating decimal mult. 0181 will result in overflow in case of all 9's. This will invalidate the 2 subsequent store Address instructions.

IBM

Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: (A)/(B) → K Subroutine WRITTEN BY: _____

(A) = $d_1 D_1$, (B) = $d_2 D_2$ d=xx, D= x. xxxxxxx

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0213	SLT	35	0002	0169	
0169	STU	21	0174	0177	d_2 → 0174
0177	STL	20	0081	0184	D_2 → 0081
0184	RAL	65	0037	0191	
0191	SLT	35	0002	0148	
0148	STU	21	0059	0263	d_1 → 0059
0263	RAU	60	8002	0221	
0221	SRT	30	0001	0227	$D_3^1 = D_1/D_2$
0227	DIV RU	64	0081	0464	
0464	BR MIN	46	0159	0209	
0159	SL	16	0162	0167	Round D_3^1 in 8th decimal place.
0209	AL	15	0162	0167	
0167	SLT	35	0001	0223	Test to see if rounded D_3^1 is of the
0223	BRNZU	44	0130	0228	form x. xxxxxxxxx or . xxxxxxxxx
0228	BR MIN	46	0132	0182	
0132	AL	15	0185	0189	D_3^1 is of the form o. xxxxxxxxx, thus
0182	SL	16	0185	0189	round D_3^1 in the 9th decimal place
0189	SRT	30	0002	0145	to get D_3.
0145	SLT	35	0002	0152	
0152	BR MIN	46	0255	0156	
0255	SABL	18	0059	0164	
0164	SL	16	0217	0321	
0321	AABL	17	0174	0179	$d_3 = d_1 - d_2 + 50-1$
0156	AABL	17	0059	0214	
0214	AL	15	0217	0371	
0371	SABL	18	0174	0179	
0130	SRT	30	0003	0140	D_3^1 is of the form x. xxxxxxxxx, thus
0140	SLT	35	0002	0149	$D_3 = D_3^1$
0149	BR MIN	46	0153	0154	
0153	SABL	18	0059	0313	
0313	SL	16	0166	0321	$d_3 = d_1 - d_2 + 50$
0154	AABL	17	0059	0363	
0363	AL	15	0166	0371	
0179	RAU	60	8002	0137	
0137	SRT	30	0002	0143	Put d_3 in normal position.
0143	AU	10	8002	0202	
0202	STU	21	0057	0026	
0162	:0000000050				
0185	:0000000450				Constants
0217	:0000000049				
0166	:0000000050				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 01: (A) + (B) → K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0001	SLT	35	0002	0157	
0157	LD	69	0110	0165	
0165	ST DA	22	0073	8001	Put B in 0057
8001	RAL	65	B	0453	
0453	STL	20	0057	0054	
0110	: 6500000453				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 02: (A) + (K) → K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0002	NO OP	00	0000	0054	

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 03: (A) - (B) → K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0003	SLT	35	0002	0160	
0160	LD	69	0215	0168	
0168	ST DA	22	0073	8001	Put -B in 0057
8001	RSL	66	B	0453	
0453	STL	20	0057	0054	
0215	: 6600000453				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 04: (K) - (A) → K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0004	RSL	66	0037	0133	Put -A in 0037
0133	STL	20	0037	0054	

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 05. (A) + (B) → C

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0005	SLT	35	0002	0212	
0212	LD	69	0216	0220	Modify instruction to get B
0220	ST DA	22	0273	0176	
0176	RAU	60	0029	0183	
0183	AU	10	0186	0192	Increase i+1 to i+2
0192	AL	15	8003	0199	
0199	LD	69	0252	0206	
0206	ST DA	22	0059	8001	Get third address (C) from i+2
8001	AABL	17	i+2	0463	
0463	LD	69	0057	0262	Put K in 0265 temporarily
0262	ST D	24	0265	0218	
0218	SL	16	8003	0175	Store i+2 in 0029
0175	ST D	24	0029	0187	
0187	LD	69	0142	0196	Modify add routine to store sum
0196	ST DA	22	0053	0273	in C.
0273	RAL	65	B	0453	Get (B) and transfer to add
0453	ST L	20	0057	0054	routine
0188	LD	69	0265	0195	Return K to 0057
0195	ST D	24	0057	0194	
0194	LD	69	0197	0150	Restore last instruction of Add
0150	ST D	24	0053	0026	routine to normal.
0216	: 6500000453				
0186	! 0000010000				
0252	: 1700000463				Constants
0142	: 2000000188				
0197	: 2000570026				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 06: (A) - (B) → C

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0006	SLT	35	0002	0264	Modify interpretation of 05 to
0264	LD	69	0267	0220	obtain -B
0267	: 6600000453				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 07: (A) x (B) → C

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0007	SLT	35	0002	0266	
0266	RSL	66	8002	0225	
0225	AU	10	0029	0233	Modify instruction to get B
0233	LD	69	0236	0190	
0190	ST DA	22	0243	0246	
0246	RAL	65	8003	0203	
0203	AL	15	0207	0261	
0261	LD	69	0314	0268	Increase i+1 to i+2. Get C and
0268	ST DA	22	0073	8001	
8001	SU	11	i+2	0456	modify last instruction of multiply
0456	ST L	20	0029	0239	
0239	AU	10	0242	0198	routine to store product in C
0198	ST U	21	0083	0243	
0243	RAL	65	B	0456	Get B and store it in 0089. Go
0456	ST L	20	0089	0134	to multiply sub-routine.
0454	LD	69	0226	0230	Restore last instruction of multiply
0230	ST D	24	0083	0026	routine to normal
0236:	6500000465				
0207:	0000010000				
0314:	1100000456				Constants
0242:	2000000454				
0226:	2000370026				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 08: (A) x (B) → K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0008	SLT	35	0002	0315	
0315	LD	69	0219	0172	
0172	ST DA	22	0277	8001	Get B and store it in 0089
8001	LD	69	B	0462	
0462	ST D	24	0089	0244	
0244	LD	69	0158	0129	Modify last inst. of mult. routine
0129	ST D	24	0083	0134	to store product in K, then go to
					0454 to restore last inst. of mult.
					routine to normal.
0219:	6900000462				
0158:	2000570454				Constants

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 09: (A)/(B) → C

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0009	SLT	35	0002	0317	
0317	RSL	66	8002	0276	
0276	AU	10	0029	0234	Modify instruction to get B
0234	LD	69	0237	0240	
0240	ST DA	22	0243	0248	
0248	AU	10	0201	0208	
0208	RAL	65	8003	0316	Increase i+1 to i+2, get C and
0316	AU	10	0269	0224	modify last instruction of divide
0224	ST L	20	0029	0232	routine to store quotient in C.
0232	LD	69	0285	0238	
0238	ST DA	22	0241	8001	
8001	SU	11	i+2	0461	
0461	ST U	21	0202	0243	
0243	RAL	65	B	0213	Get B and go to divide routine.
0293	LD	69	0296	0249	Restore last instruction of divide
0249	ST D	24	0202	0026	routine to normal.
0237:	6500000213				
0201:	0000010000				
0269:	2100000293				Constants
0285:	1100000461				
0296:	2100570026				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 10: (A)/(B) → K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0010	SLT	35	0002	0222	
0222	LD	69	0237	0290	Put B in lower accumulator and go
0290	ST DA	22	0243	8001	to divide routine
8001	RAL	65	B	0213	
0237	6500000213				Constant

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 11: Branch Minus

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0011	SRT	30	0002	0417	
0417	AU	10	0037	0378	Test for minus. If plus, do not branch
0378	BR MIN	46	0382	0026	
0382	SLT	35	0002	0339	
0339	SRT	30	0002	0397	Test for zero. If zero, do not branch
0397	BRNZU	44	0352	0026	
0352	SLT	35	0004	0420	
0420	LD	69	0373	0426	Modify 0029 and get next instruction
0426	ST DA	22	0029	8001	
8001	RAL	65	B	0439	
0373:	6500000439				Constant

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 12: Branch

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0012	SRT	30	0002	0420	
0420	LD	69	0373	0426	Put A in 0029
0426	ST DA	22	0029	8001	
0373:	6500000439				Constant

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 13: Branch Non-Zero

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0013	SRT	30	0002	0423	
0423	AU	10	0037	0431	
0431	BR MIN	46	0382	0336	Put -|A| in upper accumulator
0336	SU	11	8001	0443	
0443	SU	11	8001	0382	
0382	SLT	35	0002	0339	
0339	SRT	30	0002	0397	Test for zero. If zero, do not branch
0397	BRNZU	44	0352	0026	
0352	SLT	35	0004	0420	
0420	LD	69	0373	0426	Modify 0029 and get next instruction
0426	ST DA	22	0029	8001	
8001	RAL	65	B	0439	
0373:	6500000439				Constant

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 14: (A) x (B) +(K) ⟶ K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0014	SLT	35	0002	0422	
0422	LD	69	0375	0278	
0278	ST DA	22	0081	8001	Put B in 0089
8001	RAL	65	B	0358	
0358	ST L	20	0089	0392	
0392	LD	69	0398	0357	Modify last instruction of multiply
0357	ST D	24	0083	0134	routine to store product in 0037
0399	LD	69	0402	0355	Restore last instruction of multiply
0355	ST D	24	0083	0054	routine to normal
0375:	6500000358				
0398:	2000370399				Constants
0402:	2000370026				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 15: (K) - (A) x (B) ⟶ K

WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0015	SLT	35	0002	0421	Modify "Interpretation of 14" to
0421	LD	69	0424	0278	obtain -B
0424:	6600000358				Constant

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 16: $\sqrt{(A)} \longrightarrow B$ WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0016	LD	69	0326	0329	
0329	SLT	35	0002	0335	Modify instruction to store square root in B
0335	ST DA	22	0241	0306	
0306	RAL	65	0037	0323	
0323	SLT	35	0002	0279	Separate a_1 A_1. Store A_1 in 0283
0279	ST L	20	0283	0286	
0286	RAL	65	8003	0343	
0343	AU	10	8001	0251	
0251	AU	10	8001	0210	$1/2\ a_1$
0210	AU	10	8003	0367	
0367	AU	10	8002	0325	
0325	SRT	30	0001	0281	
0281	SU	11	8003	0289	Is a_1 even or odd?
0289	SLT	35	0001	0245	
0245	BRNZU	44	0250	0200	
0250	RAU	60	8001	0307	
0307	AU	10	0260	0415	a_1 even. $a_2 = \lfloor 1/2\ a_1 \rfloor + 25$
0415	SRT	30	0002	0322	
0322	ST L	20	0277	0330	$a_2 \rightarrow 0277$
0330	RAU	60	0283	0287	
0287	SRT	30	0002	0294	
0200	RAU	60	8001	0257	
0257	AU	10	0260	0365	a_1 odd. $a_2 = \lfloor 1/2\ a_1 \rfloor + 25$
0365	SRT	30	0002	0272	
0272	ST L	20	0277	0280	$a_2 \rightarrow 0277$
0280	RAU	60	0283	0337	
0337	SRT	30	0001	0294	
0294	AL	15	8003	0301	
0301	AL	15	0204	0259	Store A in 0265.
0259	ST U	21	0265	0318	Store 4+A in 0273.
0318	ST L	20	0273	0376	
0376	RAU	60	8003	0284	
0284	AU	10	8001	0291	Compute 1+4A
0291	AU	10	8003	0299	
0299	AU	10	0253	0258	
0258	SRT	30	0001	0270	$X_0 = (1+4A)/(4+A)$
0270	DIV RU	64	0273	0440	
0440	AU	10	0265	0369	
0369	SL	16	8002	0327	Test to see if A is zero.
0327	BRNZ	45	0380	0274	
0380	DIV RU	64	8001	0460	
0460	AL	15	8001	0413	
0413	SRT	30	0001	0320	
0320	AU	10	8002	0379	$X_1 = 1/2\ (X_0 + A/X_0)$
0379	AL	15	8001	0387	
0387	AL	15	8002	0295	
0295	AL	15	8003	0303	
0303	RAL	65	8002	0311	
0311	AU	10	0265	0370	
0370	SL	16	8002	0381	

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 16: $\sqrt{(A)} \rightarrow B$ WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0381	DIV RU	64	8001	0451	
0451	AL	15	8001	0304	
0304	SRT	30	0001	0312	
0312	AU	10	8002	0374	$X_2 = 1/2\ (X_1+A/X_1)$
0374	AL	15	8001	0282	
0282	AL	15	8002	0292	
0292	AL	15	8003	0300	
0300	RAL	65	8002	0310	
0310	AU	10	0265	0419	
0419	SL	16	8002	0377	
0377	DIV RU	64	8001	0452	
0452	AL	15	8001	0256	
0256	SRT	30	0001	0364	$X_3=1/2\ (X_2+A/X_2)$
0364	AU	10	8002	0324	
0324	AL	15	8001	0331	
0331	AL	15	8002	0340	
0340	AL	15	8003	0450	
0450	RAL	65	8002	0409	
0409	AU	10	0265	0180	
0180	SL	16	8002	0144	
0144	DIV RU	64	8001	0458	
0458	AL	15	8001	0459	$X_4 = 1/2\ (X_3+A/X_3)$
0459	SRT	30	0001	0235	
0235	AU	10	8002	0146	
0146	AL	15	8001	0405	
0405	AL	15	8002	0231	
0231	AL	15	8003	0247	
0247	RAU	60	8002	0305	
0305	SCT	36	0000	0361	Normalize the square root and
0361	BROV	47	0366	0366	reset overflow circuit.
0366	SRT	30	0002	0274	
0274	AU	10	0277	0241	Put exponent in normal position
0241	ST U	21	B	0026	and store result in B
0236:	2100000026				
0260:	-0000000025				Constants
0204:	0400000000				
0253:	0100000000				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 17: $\sqrt{(A)^2 + (B)^2 + (C)^2} \longrightarrow K$ WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0017	SLT	35	0002	0275	
0275	LD	69	0037	0136	
0136	ST D	24	0089	0342	
0342	LD	69	0345	0298	
0298	ST DA	22	0351	8001	Prepare to compute $(A)^2$ and
8001	RAL	65	B	0408	
0408	LD	69	0427	0430	put $(A)^2$ in 0057
0430	ST D	24	0083	0346	
0346	ST L	20	0265	0134	temporarily put (B) in 0265.
0360	RAL	65	0265	0333	
0333	ST L	20	0089	0297	Prepare to compute $(B)^2$ and
0297	LD	69	0254	0308	
0308	ST D	24	0083	0141	put $(B)^2$ in 0037
0341	LD	69	0395	0349	Prepare to compute $(A)^2 + (B)^2$ and
0349	ST D	24	0053	0054	store it in 0057.
0362	RAL	65	0029	0383	
0383	AL	15	0186	0391	
0391	LD	69	0394	0347	
0347	ST DA	22	0351	0354	Increase i+1 to i+2. Get C
0354	AU	10	8001	0411	
0411	ST L	20	0029	8003	and prepare to compute $(C)^2$
8003	RAL	65	i+2	0466	
0466	SL	16	0338	8002	and put it in 0037
8002	RAL	65	C	0437	
0437	ST L	20	0089	0396	
0396	LD	69	0302	0368	
0368	ST D	24	0083	0141	
0390	LD	69	0393	0348	
0348	ST D	24	0053	0309	Prepare to compute $(A)^2+(B)^2+(C)^2$
0309	LD	69	0226	0429	and store it in 0037
0429	ST D	24	0083	0054	
0328	LD	69	0332	0288	Restore last instruction of add
0288	ST D	24	0241	0344	routine to normal
0344	LD	69	0197	0350	Prepare to compute $\sqrt{(A)^2+(B)^2+(C)^2}$
0350	ST D	24	0053	0306	
0345:	6500000408				
0427:	2000570360				
0254:	2000370341				
0395:	2000570362				
0394:	6500000466				
0338:	6500000437				Constants
0302:	2000370390				
0393:	2000370328				
0332:	2100570026				

IBM

Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION OF 18: $\sum_{i=1}^{n} (A_i) \times (B_i) \rightarrow K$ WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0018	SRT	30	0002	0425	
0425	SU	11	0029	0384	Store (18 $A_1 B_1$) in 0389
0384	ST L	20	0389	0442	
0442	RSL	66	8003	0401	
0401	ST U	21	0057	0412	Put zeros in K
0412	AL	15	0416	0432	
0432	LD	69	0385	0388	Increase i+1 to i+2 and get n
0388	ST DA	22	0241	8001	
8001	SU	11	i+2	0457	
0457	ST L	20	0029	0433	
0433	SU	11	0386	0441	Store n-1 in 0386
0441	LD	69	0444	0447	Modify last instruction of add
0447	ST D	24	0053	0356	routine to return control to 410
0356	ST U	21	0414	0434	
0434	RAL	65	0389	0445	Prepare to get B_1
0445	SLT	35	0004	0422	
0410	RAU	60	0414	0418	Test to see if n has been reduced
0418	BRNZ	45	0122	0438	to zero.
0122	SU	11	0428	0436	
0436	AABL	17	0389	0446	Decrease n by 1. Increase A_i and
0446	AL	15	0449	0359	B_i by 1.
0359	ST U	21	0414	0435	
0435	ST L	20	0389	0448	
0448	LD	69	0403	0406	
0406	ST DA	22	0059	8001	Get A_{i+1} and prepare to get B_{i+1}
8001	LD	69	A_{i+1}	0407	
0407	ST D	24	0037	0404	
0404	SLT	35	0004	0422	
0438	LD	69	0197	0400	Restore last instruction of add
0400	ST D	24	0053	0026	routine to normal.
0416:	0000010000				
0385:	1100000457				
0386:	0000010000				
0444:	2000570410				Constants
0428:	0000010000				
0449:	0000010001				
0403:	6900000407				

FLOATING DECIMAL SUB-ROUTINES FOR THE IBM TYPE 650

G. R. Trimble, Jr.

Introduction

The sub-routines given in this article will perform the basic operations of addition, multiplication, division and square root using a floating decimal point number system. These sub-routines are designed to obtain maximum speed at the expense of doubling the memory needed for storage of data. The coder places the factors in specified registers, places the next instruction in the distributor and transfers control to the sub-routine. The floating decimal operation called for is performed, the result is left in the lower accumulator and in the distributor and control is returned to the main routine. Usually, the next operation performed will be to store the result in the desired locations.

The entire set of sub-routines uses locations 0000 through 0193. It is programmed to use every memory location within this block of memory and does not use any memory locations outside of it.

Each number is composed of 2 parts. The mantissa is 10 digits in length and may be either positive or negative. The exponent is 6 digits in length and may be either positive or negative. There is no special relation between storage locations of the mantissae and exponents.

Discussion of Sub-Routines

The system consists of the following sub-routines.

Operation	Estimated Average Time
Addition	18.4 ms
Multiplication	17.5 ms
Division	17.5 ms
Square Root	118.6 ms

Subtraction can be performed by simply reversing the sign of one of the factors before performing an addition.

The times given above do not include the time required to obtain factors, place them in the calling registers, and store the result. This requires approximately an additional 30 ms for the add, multiply and divide sub-routines and 15 ms for the square root sub-routine. If results are stored judiciously, it will not always be necessary to

obtain both factors. For example, if a product was to be used in the addition sub-routine, it could be stored in the calling registers for the addition sub-routine and thus already be in place at the time it was needed. Use of techniques such as this can reduce the time required for calling sequence by a factor of about 1/2.

It is assumed that factors are in "normal" form when a sub-routine is entered. That is, the high order digit is non-zero unless the entire number is zero. If this is the case, the result will always be in normal form.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: **FLOATING DECIMAL ADDITION** WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0000	ST D	24	0003	0006	Store next instruction in 0003
0006	RAL	65	0009	0014	0009: a_2 = XXXXXX0000
0014	SL	16	0017	0021	0017: a_1 = XXXXXX0000
0021	BR MIN	46	0024	0025	Is $a_2 \geq a_1$?
0024	LD	69	0027	0030	Prepare to shift A_2 ($a_1 > a_2$)
0030	ST DA	22	0033	0036	
0036	SRT	30	0005	0049	Is $a_1 \geq a_2 + 10$?
0049	BRNZ	45	0002	0004	
0004	RAL	65	0008	0033	0008: A_2. Shift A_2 and add
0033	SRT	30	000N	0053	A_1 to get A_3
0053	AL	15	0058	0013	0058: A_1 ($a_2 < a_1 < a_2 + 10$)
0013	BRNZU	44	0019	0018	Is $A'_3 \geq 10$?
0018	RAU	60	8002	0077	
0077	SCT	36	0000	0040	$A'_3 < 10$. Shift and Count A'_3 to get A_3
0040	ST U	21	0044	0005	and store it in 0044
0005	RSABL	68	8002	0063	
0063	LD	69	0017	0020	$a_3 = a_1$ - Count Number. Put A_3 in
0020	SLT	35	0004	0032	distributor.
0032	AL	15	8001	0041	
0041	LD	69	0044	0048	
0048	BROV	47	0003	0003	Routine completed. Go to next inst.
0019	SRT	30	0001	0026	$A'_3 \geq 10$. Shift right 1 to get A_3.
0026	ST L	20	0044	0010	
0010	RAL	65	0017	0022	$a_3 = a_1 + 1$
0022	AL	15	0078	0041	
0002	LD	69	0058	0011	$a_1 \geq a_2 + 10$. Thus $A_3 = A_1$ and
0011	ST D	24	0044	0012	$a_3 = a_1$.
0012	RAL	65	0017	0041	
0025	LD	69	0028	0031	Prepare to shift A_1. ($a_2 \geq a_1$)
0031	ST DA	22	0035	0038	
0038	SRT	30	0005	0001	Is $a_2 \geq a_1 + 10$?
0001	BRNZ	45	0054	0055	
0055	RAL	65	0058	0035	Shift A_1 and add A_2 to get A'_3.
0035	SRT	30	000N	0052	($a_1 \leq a_2 \leq a_1 + 10$)
0052	AL	15	0008	0064	
0064	BRNZU	44	0067	0068	Is $A'_3 \geq 10$?
0068	RAU	60	8002	0127	$A'_3 < 10$. Shift and count A'_3 to get A_3
0127	SCT	36	0000	0039	
0039	ST U	21	0044	0047	
0047	RSABL	68	8002	0056	$a_3 = a_2$ -count number
0056	LD	69	0009	0020	
0067	SRT	30	0001	0029	$A'_3 \geq 10$. Shift right 1 to get A_3
0029	ST L	20	0044	0050	
0050	RAL	65	0009	0022	$a_3 = a_{2+1}$
0054	LD	69	0008	0015	
0015	ST D	24	0044	0051	$a_2 \geq a_1 + 10$. Thus, $a_3 = a_2$ and
0051	RAL	65	0009	0041	$A_3 = A_2$.
0078:	00 0001	0000			
0027:	30 0000	0053			Constants
0028:	30 0000	0052			

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: FLOATING DECIMAL MULTIPLICATION TEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0130	ST D	24	0083	0086	Store next instruction in 0003
0086	RAU	60	0089	0043	0089: A_1 } $A_3^1 = A_1 \times A_2$
0043	MULT	19	0046	0102	0046: A_2 }
0102	BRMIN	46	0105	0106	Is $A_3^1 \geqslant 2$ 0?
0105	SU	11	0059	0114	$A_3 < 0$. Test to see if $\mid A_3^1 \mid \geqslant 10$
0114	BR OV	47	0117	0069	
0117	SU	11	0071	0125	$\mid A_3^1 \mid \geqslant 10$. Store A_3 in 0080
0125	ST U	21	0080	0042	
0069	SLT	35	0001	0126	$\mid A_3^1 \mid < 10$. Store A_3 in 0080
0126	ST U	21	0080	0045	
0106	AU	10	0059	0113	$A_3^1 \geqslant 0$. Test to see if $A_3^1 \geqslant 10$.
0113	BROV	47	0016	0118	
0016	AU	10	0071	0076	$A_3^1 \geqslant 10$. Store A_3 in 0080
0076	ST U	21	0080	0042	
0118	SLT	35	0001	0075	$A_3^1 < 10$. Store A_3 in 0080
0075	ST U	21	0080	0045	
0042	RAL	65	0100	0007	$\mid A_3^1 \mid \geqslant 10$.
0045	RAL	65	0101	0007	$\mid A_3^1 \mid < 10$. } $a_3 = a_1 + a_2 + (0 \text{ or } 1)$
0007	AL	15	0061	0066	0061: a_1
0066	AL	15	0119	0023	0119: a_2
0023	LD	69	0080	0083	Put A_3 in dist. and go to next inst.
0059:	9000000	000			} Constants
0071:	1000000	000			
0100:	0000010	000			
0101:	0000000	000			

PAGE _____ OF _____

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: FLOATING DECIMAL DIVISION WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0081	ST D	24	0034	0037	Store next instruction in 0034
0037	RAABL	67	0091	0096	0091: A_1
0096	SABL	18	0099	0103	0099: A_2 } Is $\|A_1\| \geq \|A_2\|$?
0103	BRMIN	46	0156	0057	
0156	RSU	61	0109	0163	
0163	AU	10	0116	0121	0116: a_1 $\|A_2\| > \|A_1\|$. Thus
0121	SU	11	0074	0079	0074: a_2 $a_3 = a_1 = a_2 - 1$. $a_3 = a_2 - a_1 + 1$
0079	ST U	21	0084	0087	
0087	RAU	60	0091	0145	$A_3 = A_1 / A_2$
0145	DIV RU	64	0099	0073	
0057	RAU	60	0116	0171	$\|A_1\| \geq \|A_2\|$. Thus
0171	SU	11	0074	0129	$a_3 = a_1 - a_2$
0129	ST U	21	0084	0088	
0088	RAU	60	0091	0095	
0095	LD	69	0099	0153	$A_3 = A_1 / (10\ A_2)$
0153	SRT	30	0001	0060	
0060	DIV RU	64	8001	0073	
0073	ST L	20	0128	0131	Store A_3 in 0128
0131	RAL	65	0084	0090	Put a_3 in lower accumulator
0090	LD	69	0128	0034	A_3 in distributor and go to next instruction
0109:	0000010000				Constant

PAGE _____ OF _____

IBM

Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: FLOATING DECIMAL SQUARE ROOT WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0094	STD	24	0097	0150	Store next instruction in 0097
0150	RAU	60	0104	0110	
0110	MULT	19	0065	0070	0065: a_1 , $a_2 = \left[1/2\ a_1\right]$. Test to
0070	AL	15	8003	0154	
0154	SU	11	8003	0111	see if a_1 was even or odd.
0111	SLT	35	0004	0072	
0072	STL	20	0177	0180	
0180	BRNZU	44	0133	0134	
0133	RAU	60	0137	0141	0137: A_1. If a_1 was odd, then
0141	SRT	30	0002	0098	$A_3 = \sqrt{.1\ A_1} = \sqrt{A}$
0134	RAU	60	0137	0191	If a_1 was even, $A_3 = \sqrt{A_1^1} = \sqrt{A}$
0191	SRT	30	0001	0098	
0098	RAU	60	8003	0155	
0155	AL	15	8003	0164	
0164	AL	15	0167	0122	
0122	STU	21	0176	0179	First approximation
0179	STL	20	0183	0136	
0136	AU	10	8003	0093	$X_0 = (1+4A)\ /(4+A)$
0093	AU	10	8003	0151	
0151	AU	10	0107	0062	
0062	RAU	60	8003	0120	
0120	SRT	30	0001	0178	
0178	DIV RU	64	0183	0161	
0161	AU	10	0176	0181	Is $A_1 = 0$?
0181	SL	16	8002	0139	
0139	BRNZU	44	0193	0115	
0193	DIV RU	64	8001	0185	
0185	AL	15	8001	0174	
0174	SRT	30	0001	0182	$X_1 = 1/2\ (X_0 + A/X_0)$
0182	AU	10	8002	0192	
0192	AL	15	8001	0149	
0149	AL	15	8002	0158	
0158	AL	15	8003	0165	
0165	RAL	65	8002	0123	
0123	AU	10	0176	0082	
0082	SL	16	8002	0092	
0092	DIV RU	64	8001	0186	
0186	AL	15	8001	0190	
0190	SRT	30	0001	0157	$X_2 = 1/2\ (X_1 + A/X_1)$
0157	AU	10	8002	0166	
0166	AL	15	8001	0138	
0138	AL	15	8002	0147	
0147	AL	15	8003	0112	
0112	RAL	65	8002	0172	
0172	AU	10	0176	0132	
0132	SL	16	8002	0142	
0142	DIV RU	64	8001	0188	
0188	AL	15	8001	0140	
0140	SRT	30	0001	0159	$X_3 = 1/2\ (X_2 + A/X_2)$
0159	AU	10	8002	0169	

PAGE _____ OF _____

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: FLOATING DECIMAL SQUARE ROOT WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0169	AL	15	8001	0187	
0187	AL	15	8002	0146	
0146	AL	15	8003	0162	$X_3 = 1/2\ (X_2 + A/X_2)$
0162	RAL	65	8002	0173	
0173	AU	10	0176	0135	
0135	SL	16	8002	0144	
0144	DIV RU	64	8001	0189	
0189	AL	15	8001	0148	
0148	SRT	30	0001	0108	$X_4 = 1/2\ (X_3 + A/X_3)$
0108	AU	10	8002	0168	
0168	AL	15	8001	0175	
0175	AL	15	8002	0085	
0085	AL	15	8003	0143	
0143	RAU	60	8002	0152	
0152	SCT	36	0000	0160	$A_2 = \sqrt{A_1} = X_4$ shifted 0 or 1
0160	BR OV	47	0115	0115	
0115	STU	21	0170	0124	Put a_2 in lower accumulator, A_2 in
0124	RAL	65	0177	0184	distributor and go to next inst.
0184	LD	69	0170	0097	
0107:	0100000	000			
0167:	0400000	000			Constants
0104:	0000500	000			

PAGE _____ OF _____

IBM TYPE 650 LOADING ROUTINES

G. R. Trimble, Jr.
E. C. Kubie

One of the first problems that arises in using a computer of any sort is how to enter information into the machine. The flexibility of the input system of the IBM Type 650 gives the programmer a wide range of possibilities. Most problems will require their own individual loading routines which are adapted to the particular type of data which they will be processing. The following routines, however, are of general interest since all problems require that programs and possibly tables, be entered. These routines are by no means exhaustive but merely indicate the many possibilities which exist and the ease with which this problem may be solved for the 650.

These routines are programmed so that the card reader will operate at 200 cards per minute. In some cases, it was necessary to use optimum programming to obtain this speed. In addition, they are programmed so as to use a block of memory location. The translating routine may be used on each of these programs, as well as the translating routine itself, to store them in a different set of memory locations from those for which they are programmed.

Since unpunched columns enter the machine as invalid information, the word entered will not pass the validity check if the program tries to use it. By punching zeros with a plus sign in all unused fields of the different types of loading cards, the information which enters the machine will be valid though perhaps meaningless. Thus, if it is necessary to punch out the contents of the entire memory in order that a new program can be entered, it is not necessary to make provision for avoiding locations in which invalid information may possibly have entered. This procedure greatly simplifies the routine which would be required for unloading memory.

DESCRIPTION AND FUNCTION

LL1 Loading Routine

This routine enters four words from a card, each card containing the addresses of the locations in which these four words are to be stored. It consists of nine instructions, five of which are kept on the drum in locations 1995 through 1999, the other four being entered on each card into locations 1951, 1953, 1955, and 1957. The four words to be stored are entered into locations 1952, 1954, 1956, and 1958. The routine consists essentially of a read instruction followed by four sets of load distributor-store distributor instructions. The data addresses of the store distributor instructions,

which are punched on the load card, specify the locations into which the corresponding words are to be entered. Load cards are identified by a 12 punch in column 1. The only control panel wiring necessary is from column 1 of first reading to the load hub.

LL1 is particularly useful for entering instructions. The card form is designed so that the card can be punched directly from the program sheet. The address of the location to be entered is punched followed by the word itself.

This routine is also useful for entering programs which are coded optimumly where instructions are scattered throughout the drum. It can also be used to load other loading routines which are not self-loading.

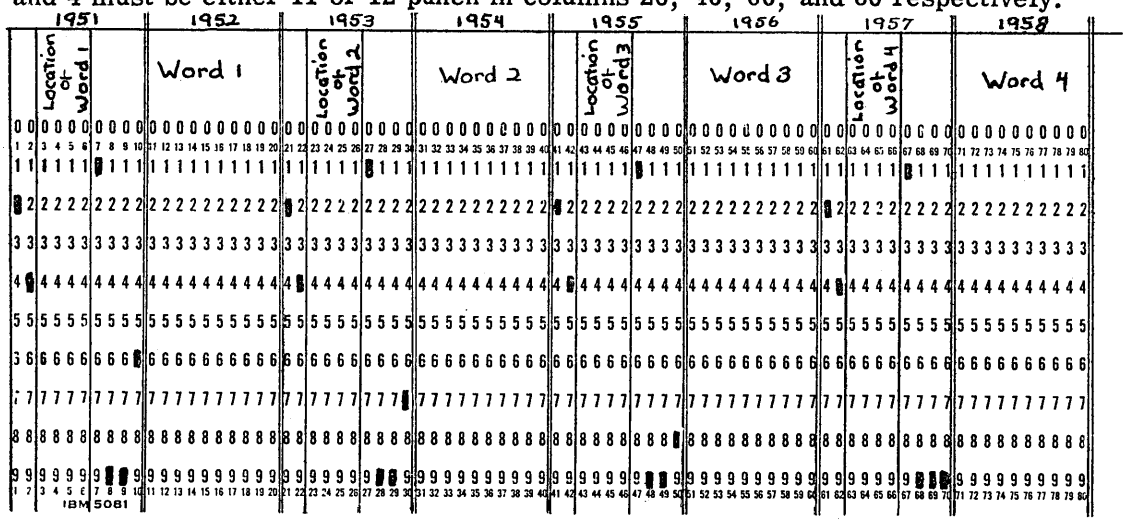
LL1 can be loaded with only two cards. To use LL1 simply place these two cards in front of the program to be loaded, set 70 1951 3000 on the storage entry switches and take the first instruction from the storage entry switches. This causes cards 1 and 2 to read into the 650 and the program punched on these cards enter LL1 in the desired locations. Control is then transferred to LL1 and the cards following are entered by it.

Once all of the desired cards have been read, it is necessary to transfer control to the main program. This can be accomplished in one of two ways. If the card read is not a load card, the next instruction will not be taken from 1995 as is usually the case, but will be taken from the location specified by the I address of the read instruction. Thus, the I address may be changed to cause control to transfer to the first instruction of the routine when the program has been entered.

The second and probably most useful method is simply to change the I address of one of the store distributor instructions punched on the load card. Thus, if only two instructions are punched on a particular load card, the I address of the second store distributor instruction, which is in columns 27 through 30 should be changed to cause control to transfer to the desired location. Obviously load cards containing one, three, or four words can be handled in the same manner.

In case only one, two, or three words are entered from the card, the remaining fields should have zeros punched. This makes it easy to punch out the contents of the entire memory at a later time if desired.

Load Card Form - 12 punches in columns 1, 10, 30, 50, 70. Signs of words 1, 2, 3, and 4 must be either 11 or 12 punch in columns 20, 40, 60, and 80 respectively.



IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: LL1 LOADING ROUTINE WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
1995	LD	69	1952	1951	
1996	LD	69	1954	1953	
1997	LD	69	1956	1955	These instructions are kept on the drum.
1998	LD	69	1958	1957	
1999	READ	70	1995	0000	
1951	STD	24	XXXX	1996	These instructions are read into the 650 on each load card. The data addresses specify the location where each of the four pieces of information punched on the card are to be stored.
1953	STD	24	XXXX	1997	
1955	STD	24	XXXX	1998	
1957	STD	24	XXXX	1999	
PROBLEM: LOADING ROUTINE TO LOAD LL1					
8000	READ	70	1951	3000	Read Card 1 into 1951 thru 1958
1951	READ	70	1901	3000	Read Card 2 into 1901 thru 1908
1901	LD	69	1952	1902	
1902	STD	24	1995	1903	
1903	LD	69	1953	1904	
1904	STD	24	1996	1905	
1905	LD	69	1954	1906	Store LL1 in 1995 thru 1999
1906	STD	24	1997	1907	
1907	LD	69	1955	1908	
1908	STD	24	1998	1957	
1957	LD	69	1956	1958	
1958	STD	24	1999	1999	
1952:	69 1952	1951			
1953:	69 1954	1953			
1954:	69 1956	1955			
1955:	69 1958	1957			
1956:	70 1995	0000			

PAGE _____ OF _____

Cards 1 and 2 for LL1

Card 1 is punched as follows: 12 punches in columns 1, 10, 20, 30, 40, 50, 60, 70, 80

```
7019013000691952195169195419536919561955691958195770199500006919561958 2419991999
      1951       1952       1953       1954       1955       1956       1957       1958
```

Card 2 is punched as follows: 12 punches in columns 1, 10, 20, 30, 40, 50, 60, 70, 80

```
6919521902241995190369195319042419961905691954190624199719076919551908 2419981957
      1901       1902       1903       1904       1905       1906       1907       1908
```

L1 Loading Routine

L1 is used to enter load cards as identified by a 12 punch in column 1. As many as seven ten-digit words may be entered on a card by means of L1. A control word punched on the card indicates where the first word is to be entered as well as the number of words to be entered. Successive words on the card are entered into successive memory locations.

Card Form for L1 Load Card:

1 10	11 20	21 30	31	70	71 80
Control Word	Word 1	Word 2	Wo	rd 6	Word 7

The control word is constructed as follows:

Column 1, double punch 12 and 0;
Column 1, punch 0;
Columns 3-6, punch address where Word 1 is to be entered.
This address is f=f_1 f_2 f_3 f_4;
Columns 7-9, punch 0;
Column 10, double punch 12 and n, where n is number of words to be entered. (n=1, 2, 3, 4, 5, 6 or 7)

Load cards must have all ten columns of the fields to be entered punched and must have the sign punched over the units position of the field (columns 10, 20, 30, etc.) 12 for plus, and 11 for minus. Fields not entered should have zeros punched in them.

Control Panel Wiring:

Column 1 of first reading is wired to the load hub. The 12 in column 1 identifies load cards.

Explanation of Program:

Instruction 1988 calls for a Read. If the card read is not a load card, the data is entered into locations 1951-1960 and the next instruction is taken from m, where m is the location in the main routine to which control is transferred. If the card read is a load card the data read is entered into locations 1951-1958 and the next instruction is taken from 1994.

Words 1 through n will be transferred to locations f through f + n - 1 by LD and ST D instructions. These two instructions will be kept in the accumulator where they will be modified and executed.

Instruction 1994 puts the control word in the accumulator.

Instructions 1979 and 1986 modify the ST D instruction so that Word 1 is stored in location f. The ST D instruction is then 24 (f) 1977.

Instructions 1989, 1981, and 1990 construct a dummy instruction which is used to indicate that the last word has been transferred. This dummy instruction is 24 (f + n) 1977.

Instructions 1984 and 1992 load the accumulator with the LD and ST D instructions. These instructions are then executed from the accumulator.

Instructions 1977 and 1985 add 1 to the data addresses of the LD and ST D instructions in the accumulator so that the next time they are executed, the next word will be transferred to its proper location.

Instructions 1993 and 1983 compare the ST D instruction in the accumulator with the dummy ST D instruction. If they are equal, as indicated by a zero in the upper half of the accumulator, the last word has been transferred and control is returned to 1988 and a new card is read. If they are not equal, the ST D instruction is regenerated in the upper half of the accumulator by adding the dummy instruction back in on instruction 1987. Control is returned to 8002 and the next word is transferred.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: L1 LOADING ROUTINE WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
1988	READ	70	1994	m	(Transfer control to location m if no load card)
1994	RAL	65	1951	1979	
1979	LD	69	1982	1986	
1986	ST DA	22	1982	1989	
1989	SLT	35	0004	1981	Prepare to store starting with location f.
1981	AL	15	8001	1990	
1990	ST DA	22	1978	1984	
1984	RAL	65	1991	1992	
1992	AU	10	1982	8002	
8002	LD	69	(1952)	8003	Store word j in location f + j - 1
8003	ST D	24	(f)	1977	
1977	AU	10	1980	1985	Modify instructions in preparation to store
1985	AL	15	8001	1993	word j + 1.
1993	SU	11	1978	1983	Test to see if word n - 1 has been stored. If
1983	BRNZU	44	1987	1988	so, read the next card.
1987	AU	10	8001	8002	Prepare to store word j + 1.
1982	ST D	24	0000	1977	
1991	LD	69	1952	8003	Constants
1980		00	0001	0000	
1978	ST D	24	f + n	1977	Temporary Storage
1951		00	f	000n	Control word punched on load card. f is address where first word on card is to be entered. n is the number of words to be entered.

L2, L2A, L2B, L2C. Loading Routine

L2 is used to load eight full words from each card. The initial address f_o is given, and words are located sequentially beginning at f_o. Cards containing a 12 in column 1 will be loaded in consecutive addresses. This will continue until occurrence of a card with no 12 in the load control column at which time automatic branching is made from L2.

An optional way to get out of the routine is to set the address switches at the first location beyond the last one for which loading is desired. Then branching can be manually effected to other routines. In this case the word following the last word to be loaded must have ten zeros punched.

The calling for L2 and f_o can be effected manually, from the first card read, or from the preceding program by L2A, L2B and L2C respectively.

Wiring:

Column 1 of first reading is wired to the load hub. The 12 punch in column 1 identifies load cards.

Program Explanation:

The store distributor operations are executed from the lower accumulator and stepped up by the contents of the upper accumulator. The load distributor operations are stored in memory. Loading is then realized by repeated load distributor, store distributor (per instruction in lower accumulator), and add to lower accumulator from upper operations.

L2A: L2A is designed to initiate L2 and specify f_o manually. The storage entry switches are set to 00 f_o 1966 and control is sent to these switches by depressing in turn the program reset button and the program start button. L2A then inserts f_o into the L2 routine and reading is begun.

L2B: L2B is designed to have the first card of a group to be loaded by L2 designate f_o. The first word on this card is punched 24 f_o 1981. L2B transfers f_o into L2 and initiates L2. To call for L2B all that is required is to give an instruction address of 1970.

L2C: L2C is designed to have f_o designated by a word in memory. This word is in the

following form; 24 f_o 1979. L2C inserts f_o into L2 and initiates L2. To call for L2C all that is required is to give an instruction of the form 65 m′ 1969.

Fields on the last card which are not entered should have zeros punched in them.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-3
PRINTED IN U.S.A.

PROBLEM: L2A, B, and C LOADING ROUTINE WRITTEN BY:

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
L2 A (f_o taken from Storage Entry Switches)					
8000	No Op	00	f_o	1966	
1966	RAL	65	8000	1974	
1974	SRT	30	0004	1967	
1967	SLT	35	0004	1968	
1968	AL	15	1971	1969	L. Acc: 24 f_o 1979
1969	AU	10	1972	1977	U. Acc: 00 0001 0002
1971	ST D	24	0000	1979	Constants
1972		00	0001	0002	
L2B (f_o taken from first word of first card)					
1970	READ	70	1975	0000	
1975	RAL	65	1951	1976	L. Acc: 24 f_o 1981
1976	AU	10	1972	1980	U. Acc: 00 0001 0002
1951	ST D	24	f_o	1981	First word punched on first card
1972		00	0001	0002	Constant
L2C (f_o in memory location m′)					
m″	RAL	65	m′	1969	L. Acc: 24 f_o 1979
1969	AU	10	1972	1977	U. Acc: 00 0001 0002
m′	ST D		24	f_o	1979

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

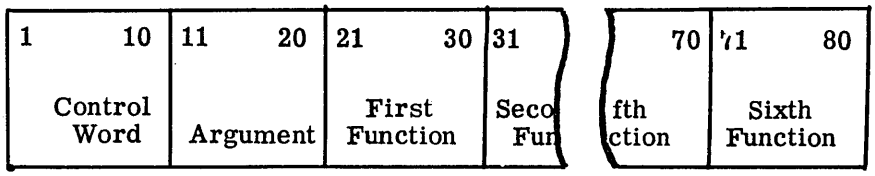
PROBLEM: L2 LOADING ROUTINE WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
1977	READ	70	1978	m	(Transfer control to location m if no load card)
1978	LD	69	1951	8002	
8002	ST D	24	(f)	1979	
1979	AL	15	8003	1980	
1980	LD	69	1952	8002	
8002	ST D	24	(f + 1)	1981	
1981	AL	15	8003	1982	
1982	LD	69	1953	8002	
8002	ST D	24	(f + 2)	1983	
1983	AL	15	8003	1984	
1984	LD	69	1954	8002	
8002	ST D	24	(f + 3)	1985	
1985	AL	15	8003	1986	Store 8 words in locations f thru f + 7.
1986	LD	69	1955	8002	
8002	ST D	24	(f + 4)	1987	
1987	AL	15	8003	1988	
1988	LD	69	1956	8002	
8002	ST D	24	(f + 5)	1989	
1989	AL	15	8003	1990	
1990	LD	69	1957	8002	
8002	ST D	24	(f + 6)	1991	
1991	AL	15	8003	1992	
1992	LD	69	1958	8002	
8002	ST D	24	(f + 7)	1993	
1993	AL	15	8003	1994	Modify instructions to repeat loop for next card.
1994	SL	16	1973	1977	
1973		00	0000	0016	Constant

LT 1 Loading Routine For Tables

LT 1 is used to enter table cards as identified by a 12 punch in column 1. One argument and as many as six functions associated with that argument may be entered on a card by means of LT 1. A control word punched on the card indicates where the argument is to be entered, the number of arguments and functions to be entered, and where the functions will be placed relative to the argument.

Card Form for LT 1 Table Loading Card:



The control word is constructed as follows:

Column 1, double punch 12 and 0.
Column 2, punch n, where n is the number of arguments and functions to be entered. For one argument n=1, for one argument and one function n=2, etc. n=7 is maximum.
Columns 3-6, punch address where argument is to be entered. This address is $a=a_1\ a_2\ a_3\ a_4$.
Columns 7-10, punch increment between argument and first function. This increment is i, $i=i_1\ i_2\ i_3\ i_4$. Thus the argument will be stored in a, the first function stored in a + i, the second function stored in a + 2i, etc. Column 10 must also have a 12 punch.

Table load cards must have all ten columns of the fields to be entered punched and must have the sign punched over the units position of the field (columns 10, 20, 30, etc.) 12 for plus and 11 for minus. Fields not entered should have zeros punched in them.

Control Panel Wiring:

Column 1 of first reading is wired to the load hub. The 12 in column 1 identifies table load cards.

Explanation of Program:

Instruction 1978 calls for a Read. If the card read is not a table load card, the data is entered into locations 1951-1960 and the next instruction is taken from m, where m is the location in the main routine to which control is transferred. If the card read is a table load card the data read is entered into locations 1951-1958 and the next instruction is taken from 1994.

Instruction 1994 places the control word in the lower accumulator.

Instructions 1985 and 1988 modify the ST D instruction so that the argument is stored in location a.

Instructions 1989, 1981, and 1991 store the increment i in 1975.

Instructions 1984, 1992, 1976 and 1986 construct a dummy instruction which is used to indicate that the last word has been transferred. This dummy instruction is 69 (1953+n) 8002. When the LD instruction has been modified so that it is equal to the dummy instruction the last function has been transferred.

Instructions 1993 and 1980 load the accumulator with the LD and ST D instructions. These instructions are then executed from the accumulator.

Instructions 1972 and 1979 modify the LD and ST D instructions in the accumulator so that the next time they are executed the next piece of data will be transferred to its proper location.

Instructions 1987 and 1973 compare the LD instruction in the accumulator with the dummy LD instruction. If they are equal, as indicated by a zero in the upper half of the accumulator, the last word has been transferred and control is returned to 1978 and a new card is read. If they are not equal, the LD instruction is regenerated in the upper half of the accumulator by adding the dummy instruction back in on instruction 1977. Control is returned to 8003 and the next word is transferred.

Note that if i=1, LT1 is functionally identical with L1. This increased flexibility is gained by an increase in the number of storage locations required however. LT1 uses 5 more storage locations than L1.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: LT 1 LOADING ROUTINE FOR TABLES WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
1978	READ	70	1994	m	(Transfer control to location m if no load card)
1994	RAL	65	1951	1985	
1985	LD	69	1974	1988	
1988	ST DA	22	1974	1989	
1989	LD	69	8003	1981	
1981	SLT	35	0004	1991	
1991	ST DA	22	1975	1984	Prepare to store starting with location a.
1984	RAL	65	8003	1992	
1992	SLT	35	0002	1976	
1976	AL	15	1983	1986	
1986	ST DA	22	1990	1993	
1993	RAL	65	1974	1980	
1980	AU	10	1983	8003	
8003	LD	69	(1952)	8002	Store word j in location a + (j-1)i.
8002	ST D	24	(a)	1972	
1972	AL	15	1975	1979	Modify instructions in preparation to store
1979	AU	10	1982	1987	word j + 1.
1987	SU	11	1990	1973	Test to see if word n-1 has been stored. If
1973	BRNZU	44	1977	1978	so, read the next card.
1977	AU	10	8001	8003	Prepare to store word j + 1.
1974	ST D	24	0000	1972	
1983	LD	69	1952	8002	Constants
1982		00	0001	0000	
1975		00	i	0000	Increment i gotten from control word.
1990	LD	69	(1952+n)	8002	Temporary Storage
1951		n	a	i	Control word punched on table load card. a is address where the first word is to be stored. i is increment to be added to addresses of successive words. n is the number of words to be stored.

TR 1 Translating Routine

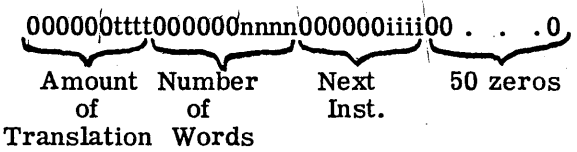
This routine is used to translate a sub-routine or sub-program to a different set of memory locations from those for which it was programmed. It will simultaneously modify each instruction as required, so that they will be properly executed from their new locations. Each instruction or constant in the sub-routine will have associated with it a six digit number. Four digits of this number specify its location within the sub-routine. The remaining two digits indicate whether or not either the D or I address parts of the number should be modified when it is translated. An 8 indicates that the corresponding address is to be modified, and a 9 indicates that the address is not to be modified. Thus, for example, the modification control digits for a constant would be 99 since nothing in the constant is to be modified. Most instructions, however, will have 88 as the modification control digits since both the data and instruction addresses will be modified. A shift instruction would usually have 98 as its modification control digit, since the data address of a shift instruction should not be modified. Similarly, an instruction in which the D address should be modified while the I address should not be modified will have 89 as its modification control digits.

A master card placed in front of the deck containing the sub-routine indicates the amount by which the sub-routine is to be translated, the number of instruction in that sub-routine, and an indication of what should be done after the sub-routine has been completely translated.

The detail cards will each contain five ten-digit words and their corresponding control information. If less than five words are punched in the detail card, the remaining fields should have zeros punched in them.

Master Card Form:

12 punches in columns 1, 10, 20, 30, 40, 50, 60, 70, 80.



Detail Card Form:

The detail card contains five sixteen-digit fields each punched as follows:
Col. 1: D modification control column; 8 if data address is to be modified, 9 if data address is not to be modified.

Col. 2: I modification control column; 8 if instruction address is to be modified, 9 if instruction address is not to be modified.
Col. 3-6: Location of word within sub-routine before translation. Column 6 also has a 12 punch.
Col. 7-16: Punch the word to be translated. The sign of the word is in column 16 (units position).

Control Panel Wiring:

Wire from column 1 of First Reading to the Load Hub. A 12 punch in column 1 identifies master cards.

The six control digits for the first word are wired to enter the 6 low order positions of Storage Entry Word 1. Word 1 is also wired for Word Size 6. The first word is wired to enter Storage Entry Word 2. The Sign over Units switch and Read Plus switch must also be wired. Word 2 must also be wired for Word Size 10. The remaining 4 fields on the card are wired similarly to enter Storage Entry Words 3 through 10.

Explanation of Program:

The instruction in location 1933 calls for a read. If the card read is a master card, the next instruction will be taken from location 1991 and the nine instructions starting with 1991 and ending with 1990 will be executed. These instructions store the amount of translation, the number of instructions and the location of the next instruction, i, in their proper locations. It places the amount of translation, t, in the data address positions of one word, the instruction address positions of another word, and in both data and instruction address positions of a third word. Thus, when it has been determined how the word to be translated should be modified, all that must be done is to add one of these three numbers or zero to that word before storing it in its new location.

The location of the next instruction, i, is used to indicate what should be done after the sub-routine has been translated. For example, if another sub-routine follows which must also be translated, i will take the form 00 0000 1933. When the translation of the first sub-routine is complete, another read instruction is given to call in the master card for the following sub-routine. When the last sub-routine has been translated, i is used to transfer control to the first instruction of the program.

The four instructions, starting with the one in location 1939, simply set two

other instructions to their initial values.

The five instructions beginning with the one in location 1937, add t to the sub-routine location of the word being translated and modifies a "store" instruction so that this word will be stored in location L + t. Beginning with the instruction in location 1968, the modification control digits are analyzed to determine whether the data address and/or the instruction address positions of the word to be stored should be modified. This is accomplished by adding one of the four constants computed from the master card. The instruction in location 1965 then stores the modified word in its new location.

The number of instructions, n, is decreased by one each time a word is stored and is zero tested to see if the last word in the sub-routine has been stored. If so, control is transferred to i. If the last word has not been stored, a test is made then to determine whether the last (5th) word on the card has been stored. If so, a new card is read. If more words remain to be transferred, the instructions are modified in preparation for storing the next word.

Use of TR 1:

Once it has been determined which sub-routines are to be used and where they are to be placed, the programmer punches the amount of translation necessary to place the library sub-routine in the desired locations and the other information necessary, on a master card. This master card is placed in front of the library sub-routine and all of the sub-routines are assembled. These sub-routines, along with the rest of the program, are then read into the 650 and the problem is begun.

Routines which are programmed optimumly should be translated only by an even amount in order to preserve the even-odd conditions.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: TR1 TRANSLATING ROUTINE WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
1933	READ	70	1991	1939	
1991	RAL	65	1951	1938	
1938	ST D	24	1992	1967	Store 00 0000 (t)
1967	SLT	35	0004	1934	
1934	ST DA	22	1941	1929	Store 00 (t) (t) — Master Card Only
1929	ST L	20	1993	1940	Store 00 (t) 0000 — Master Card Only
1940	LD	69	1953	1969	
1969	ST D	24	1948	1930	Store i
1930	RAL	65	1952	1990	
1990	ST L	20	1989	1933	Store n
1939	LD	69	1942	1945	
1945	ST D	24	1949	1972	Set instructions 1937 and 1949 to initial values.
1972	LD	69	1932	1935	
1935	ST D	24	1937	8001	
1937	RAL	65	(1951)	1964	
1964	SLT	35	0004	1976	Prepare to store a word in location L + t
1976	AL	15	1993	1947	
1947	LD	69	1950	1961	
1961	ST DA	22	1965	1968	
1968	LD	69	8002	1975	
1975	BRD 9	99	1979	1982	Test to see how word should be modified before being stored.
1979	BRD 10	90	1984	1988	
1982	BRD 10	90	1987	1986	
1986	RAL	65	1994	1949	
1987	RAL	65	1993	1949	
1984	RAL	65	1941	1949	Modify word and store it in location L + t.
1988	RAL	65	1992	1949	
1949	AL	15	(1952)	1965	
1965	ST L	20	(L + t)	1974	
1974	RSL	66	1981	1936	Test to see if last word in routine has been stored. If so, transfer control to i
1936	AL	15	1989	1943	
1943	BRNZ	45	1946	1948	
1946	AU	10	1949	1963	
1963	SU	11	1966	1971	Test to see if last word on card has been stored.
1971	BRNZU	44	1978	1990	
1978	AU	10	1931	1985	
1985	ST L	20	1989	1944	Prepare to store next word
1944	ST U	21	1949	1962	
1962	RAL	65	8003	1970	
1970	SL	16	1973	1977	Prepare to read next card.
1977	LD	69	1980	1983	
1983	ST DA	22	1937	8001	
1941		00	t	t	
1992		00	0000	t	
1993		00	t	0000	Constants determined from Master Card
1994		00	0000	0000	
1989		00	0000	n	
1948		00	0000	i	
1942		15	1952	1965	
1932		65	1951	1964	
1950		20	0000	1974	
1981		00	0000	0001	Constants
1966		15	1960	1965	
1931		15	1962	1965	
1973		00	0001	0000	
1980		65	0000	1964	

A METHOD FOR PERFORMING DOUBLE PRECISION ARITHMETIC ON THE IBM TYPE 650

G. R. Trimble, Jr.

Introduction

It is sometimes necessary to perform computations using a larger number size than is basically provided on the 650. This system has been developed to perform the basic operations of addition, subtraction, multiplication and division using numbers which are 20 decimal digits in length. It was designed with coding convenience as a prime objective and uses an interpretative routine to perform double precision arithmetic.

Double precision instructions, that is, instructions which are to be interpreted have a negative sign. A double precision instruction consists of a 2 digit operation code, a 4 digit address specifying the location of the first factor and another 4 digit address specifying the location of the second factor. The result of the operation is always stored in a pair of memory locations referred to as C. A 5th operation, "Store", is used to transfer the result from C to the locations desired by the programmer. In order to store the result where desired, it will usually be necessary to follow one of the 4 basic arithmetic operations by a store instruction. The net effect of such a procedure is four 3-address operations.

The address given refers to the 10 high order digits of the factor to be operated upon. The interpretative routine automatically selects the 10 low order digits from the next memory location. Thus, the high order and low order digits of a number will always be stored in successive memory locations.

Should a positive instruction appear, it is interpreted as a normal 650 instruction and subsequent instructions are not interpreted. Thus, upon occurrence of a positive instruction, the 650 operates in its usual D-I mode. This continues until an instruction address of 0004 is given which causes the control to return to the interpretative routine. The program will return to the double precision mode of operation at the point of departure.

For example, consider the following sequence:

Location	Contents
n	double precision instruction (-)
n+1	double precison instruction (-)

n+2 double precision instruction (-)

n+3 normal 650 instruction (+)

(the contents of n+3 are interpreted as a normal 650 instruction including the instruction address for sequencing. Normal execution of instructions continues until 0004 is used as an instruction address, at which time the program returns to the double precision mode beginning with the instruction in location n+4).

Number Form

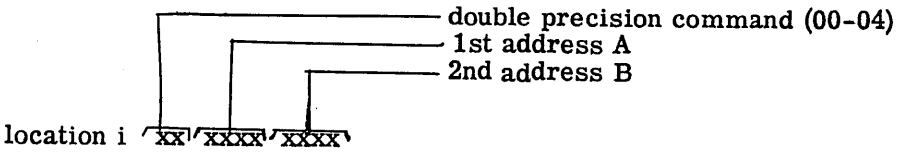
The number form is as follows:

.xxxxxxxxxxxxxxxxxxxx=$A_1 + A_2$.

The low order digits of the number are stored in the location immediately following the one containing the high order digits.

Instruction Form

The instruction form is as follows:



Operations

The following is a list of the operations with their codes and estimated average time.

Code	Operation	Estimated Average Time (ms)
00	B=O, C⟶A	23.88
01	A+B⟶C	49.46
02	A-B⟶C	49.46
03	AxB⟶C	113.40
04	A/B⟶C	182.52

Since the factors are assumed to be less than one, the numbers must be scaled so that the results will be less than one. If any answer should exceed one, the machine will stop. Also, the condition $|A| < |B|$ must be satisfied for the divide operation.

This system requires 115 storage locations. It uses locations 0000 through 0114. Every location within this block is used, thus making it easy to incorporate this program with other programs. The translating routine, TR1, may be used to translate this routine to any desired locations. It should be translated by an even amount, however,

to preserve the even-odd conditions.

Explanation of Programs.

The interpretative routine determines where the next instruction is located, obtains that instruction, and analyzes it to see if it must be interpreted or if it is a normal 650 instruction. If it is a normal 650 instruction it is executed as such. If it must be interpreted the interpretative routine continues to analyze it, obtains the required factors, transfers control to the proper sub-routine.

The constant in location 0091 facilitates use of TR1. During read in by TR1 this constant is modified so that its instruction address digits contain the amount of translation. This is then added to the operation code so that control will be transferred to the translated sub-routine during analysis of the operation code.

The store, add, subtract, and multiply sub-routines are straightforward and simple. The latter part of the add sub-routine is common to both the subtract and multiply sub-routines thus saving memory locations.

The divide sub-routines uses the following approximation to perform double precision division.

$$\frac{A}{B} = \frac{A}{B_1+B_2}$$

$$= \frac{A}{B_1}\left(\frac{1}{1+\frac{B_2}{B_1}}\right)$$

$$\sim \frac{A}{B_1}\left(1-\frac{B_2}{B_1}\right)$$

The sub-routine computes $\frac{A}{B_1}$ and $1-\frac{B_2}{B_1}$. The multiply sub-routine is used to multiply these two factors to obtain the final quotient.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0004	RAL	65	0007	0061	
0061	AL	15	0014	8002	Increase n to n+1 and get instruction
8002	SU	11	n+1	0008	n+1
0008	BR MIN	46	0072	0099	Is n+1 a normal 650 instruction?
0072	AU	10	8001	0045	
0045	AU	10	8001	0053	n+1 is a normal 650 instruction
0053	ST L	20	0007	8003	
8003	Normal 650 Instruction				
0099	ST L	20	0007	0010	n+1 must be interpreted
0010	AU	10	0013	0017	
0017	RAL	65	8003	0025	Is operation code 00?
0025	BR OV	47	0028	0032	
0028	LD	69	0081	0084	
0084	ST DA	22	0087	8001	
8001	LD	69	A_1	0067	
0067	ST D	24	0023	0026	
0026	AL	15	0029	0033	Get (A) and store it in 0023, 0031
0033	LD	69	0036	0039	
0039	ST DA	22	0093	8001	
8001	LD	69	A_2	0071	
0071	ST D	24	0031	0034	
0034	SLT	35	0004	0046	
0046	LD	69	0049	0052	
0052	ST DA	22	0005	8001	
8001	LD	69	B_1	0115	
0115	ST D	24	0006	0009	Get (B) and store it in 0006, 0015
0009	AL	15	0062	0018	
0018	LD	69	0021	0024	
0024	ST DA	22	0077	8001	
8001	LD	69	B_2	0110	
0110	ST D	24	0015	0068	
0068	SRT	30	0002	0075	Analyze operation code and
0075	AU	10	0091	8003	transfer to proper sub-routine.
8003	NO OP	00	0000	Oper.	
0007:	11 n	0008			Address of instruction being interpreted
0014:	00 0001	0000			
0013:	99 0000	0000			
0081:	69 0000	0067			
0029:	00 0001	0000			
0036:	69 0000	0071			Constants
0049:	69 0000	0115			
0062:	00 0001	0000			
0021:	69 0000	0110			
0091:	00 0000	0000			

PAGE _____ OF _____

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: 00: (C) ⟶ A WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0032	AL	15	0037	0041	
0041	LD	69	0044	8002	
8002	ST D	24	A_1	0106	Transfer contents of C to A
0106	AL	15	0043	0048	
0048	LD	69	0051	8002	
8002	ST D	24	A_2	0004	
0037:	25 0000 0106				Constants
0043:	00 0000 9898				

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: 01: (A)+(B) ⟶ C WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0000	RAU	60	0006	0011	
0011	AL	15	0015	0019	$A_1 + B_1 + A_2 + B_2$
0019	AU	10	0023	0027	
0027	AL	15	0031	0035	
0035	BR OV	47	0107	0040	Test for overflow
0040	ST U	21	0044	0047	Store sum in C
0047	ST L	20	0051	0004	
0107	STOP	01	0000	0107	Overflow on addition (or subtraction)

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: 02: (A)-(B) ⟶ C WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0001	RSU	61	0006	0012	
0012	SL	16	0015	0019	$A_1 - B_1 + A_2 - B_2$
0019	AU	10	0023	0027	
0027	AL	15	0031	0035	
0035	BR OV	47	0107	0040	Test for overflow
0040	ST U	21	0044	0047	Store difference in C
0047	ST L	20	0051	0004	
0107	STOP	01	0000	0107	Overflow on subtraction

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: 03: (A)x(B) → C WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0002	RAU	60	0006	0064	
0064	MULT	19	0023	0055	A_1 X B_1 → 0076, 0085
0055	ST U	21	0076	0030	
0030	ST L	20	0085	0100	
0100	RAU	60	0006	0066	
0066	MULT	19	0031	0057	A_2 X B_1 → 0074, 0051
0057	ST L	20	0074	0092	
0092	ST U	21	0051	0054	
0054	RAU	60	0015	0020	
0020	MULT	19	0023	0058	A_1 X B_2 → 0033, 0059
0058	ST L	20	0083	0101	
0101	ST U	21	0059	0112	
0112	RAU	60	0015	0070	
0070	MULT	19	0031	0104	A_2 X B_2
0104	RAL	65	8003	0105	
0105	AL	15	0074	0080	
0080	AL	15	0083	0090	
0090	RAL	65	8003	0098	
0098	AL	15	0051	0056	$A_1 B_1 + A_1 B_2 + A_2 B_1 + A_2 B_2$
0056	AL	15	0059	0063	
0063	AU	10	0076	0082	
0082	AL	15	0085	0040	
0040	ST U	21	0044	0047	Store product in C
0047	ST L	20	0051	0004	

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: 04: (A)/(B)⟶C WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0003	RAU	60	0023	0078	
0078	AL	15	0031	0042	
0042	DIV	14	0006	0108	
0108	ST L	20	0023	0088	$A/B_1 \rightarrow$ 0023, 0031
0088	RAU	60	8003	0095	
0095	DIV RU	64	0006	0109	
0109	ST L	20	0031	0050	
0050	RAU	60	0006	0111	
0111	AL	15	0015	0022	Is B_2= 0? If so, $A/B=A/B_1$
0022	SU	11	8003	0079	
0079	BRNZ	45	0038	0019	
0038	DIV	14	8001	0113	
0113	ST L	20	0089	0096	
0096	RAU	60	8003	0103	
0103	DIV RU	64	0006	0114	
0114	RSL	66	8002	0016	Compute $1- B_2/B_1$
0016	AL	15	0069	0073	
0073	AU	10	8001	0086	
0086	SU	11	0089	0094	
0094	AL	15	0097	0102	
0102	ST U	21	0006	0060	Prepare to go to multiply
0060	ST L	20	0015	0002	routine to compute
					$C= (A/B_1) (1- B_2 /B_1)$
0069:	9999999999				Constants
0097:	0000000001				

A METHOD FOR PERFORMING COMPLEX ARITHMETIC ON THE IBM TYPE 650

G. R. Trimble, Jr.

Introduction

This method for performing complex arithmetic on the 650 has been developed since it is sometimes necessary to perform computations using complex numbers. The basic operations of addition, subtraction, multiplication and division using complex numbers are performed by this method. It was designed with coding convenience as a prime objective and uses an interpretative routine to perform complex arithmetic.

Complex instructions, that is, instructions which are to be interpreted, have a negative sign. A complex instruction consists of a 2 digit operation code, a 4 digit address specifying the location of the first factor and another 4 digit address specifying the location of the second factor. The result of the operation is always stored in a pair of memory locations referred to as C. A 5th operation, "Store", is used to transfer the result from C to the locations desired by the programmer. In order to store the result where desired, it will usually be necessary to follow one of the 4 basic arithmetic operations by a store instruction. The net effect of such a procedure is four 3-address operations.

The address given refers to the real part of the factor to be operated upon. The interpretative routine automatically selects the imaginary part from the next memory location. Thus, the real and imaginary parts of a complex number will always be stored in successive memory locations.

Should a positive instruction appear, it is interpreted as a normal 650 instruction and subsequent instructions are not interpreted. Thus, when a positive instruction occurs the 650 operates in its usual D-I mode. This continues until an instruction address of 0054 is given which causes the control to return to the complex mode of operation at the point of departure.

For example, consider the following sequence:

Location	Contents
n	complex instruction (-)
n+1	complex instruction (-)

n+2 complex instruction (-)

n+3 normal 650 instruction (+)

(the contents of n+3 are interpreted as a normal 650 instruction including the instruction address for sequencing. Normal execution of instructions continues until 0054 is used as an instruction address, at which time the program returns to the complex mode beginning with the instruction in location n+4).

Number Form

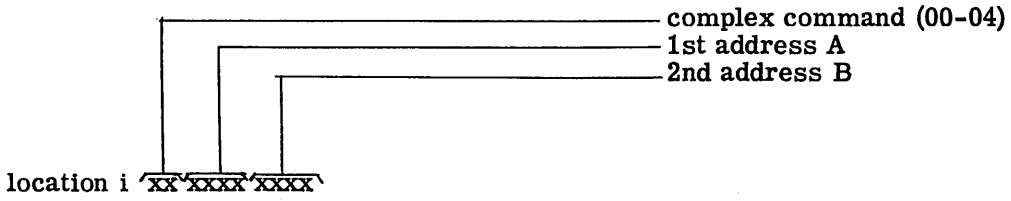
The number form is as follows:

.xxxxxxxxxx+.xxxxxxxxxxi=$A_1 + A_2 i$

The imaginary part of a number is stored in the location immediately following the one containing the real part.

Instruction Form

The instruction form is as follows:



Operations

The following is a list of the operations with their codes and estimated average time.

Code	Operation	Estimated Average Time (ms)
00	B=O, C→A	23.50
01	A+B ⟶ C	45.64
02	A-B ⟶ C	45.64
03	AxB ⟶ C	96.50
04	A/B ⟶ C	175.42

Since the factors are assumed to be less than one, the numbers must be scaled so that the results will be less than one. If any answer should exceed one, the machine will stop. Also, the condition $|A| < |B|$ must be satisfied for the divide operation.

This system requires 126 storage locations. It uses locations 0000 through 0127. Every location within this block is used, thus making it easy to incorporate this program with other programs. The translation routine, TR1, may be used to translate this routine to any desired memory location. It should be translated by an even amount however, to preserve the even-odd conditions.

Explanation of Program

The interpretative routine determines where the next instruction is located, obtains the instruction and analyzes it to see if it must be interpreted or if it is a normal 650 instruction. If it is a normal 650 instruction it is executed as such. If it must be interpreted the interpretative routine continues analyzing it, obtains the required factors, and transfers control to the proper sub-routine.

The constant in location 0091 facilitates use of TR1. During read in by TR1 this constant is modified so that its instruction address digits contain the amount of translation. This is then added to the operation code so that the control will be transferred to the translated sub-routine during analysis of the operation code.

The sub-routines are straightforward and simple. The latter part of the add sub-routine is also used by the subtract sub-routine to save memory locations. The divide sub-routine may require that the dividend and divisor both be shifted right one place if $B_1^2 + B_2^2$ should exceed one.

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: INTERPRETATION WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0054	RAL	65	0007	0011	
0011	AL	15	0014	8002	Increase n to n+1 and get instruction
8002	SU	11	n+1	0122	n+1
0122	BR MIN	46	0106	0125	Is n+1 a normal 650 instruction?
0106	AU	10	8001	0045	
0045	AU	10	8001	0053	n+1 is a normal 650 instruction.
0053	ST L	20	0007	8003	
8003	Normal	650	Instruction		
0125	ST L	20	0007	0060	n+1 must be interpreted
0060	AU	10	0063	0067	
0067	BR OV	47	0020	0026	Is n+1 operation 00?
0020	RAL	65	8003	0027	n+1 is not operation 00.
0027	LD	69	0030	0083	
0083	ST DA	22	0037	8001	
8001	LD	69	A_1	0056	
0056	ST D	24	0013	0016	Get A and store it in 0013, 0037
0016	AL	15	0019	0023	
0023	LD	69	0077	0080	
0080	ST DA	22	0035	8001	
8001	LD	69	A_2	0123	
0123	ST D	24	0037	0040	
0040	SLT	35	0004	0051	
0051	LD	69	0104	0008	
0008	ST DA	22	0061	8001	
8001	LD	69	B_1	0102	
0102	ST D	24	0005	0058	Get B and store it in 0005, 0028
0058	AL	15	0111	0015	
0015	LD	69	0068	0021	
0021	ST DA	22	0076	8001	
8001	LD	69	B_2	0126	
0126	ST D	24	0028	0031	
0031	SRT	30	0002	0038	Interpret operation code and
0038	AU	10	0091	8003	transfer to proper sub-routine.
8003	NO OP	00	0000	Oper.	
0007:	11 n		0122		Address of instruction being inter-preted
0014:	00 0001		0000		
0063:	99 0000		0000		
0030:	69 0000		0056		
0019:	00 0001		0000		
0077:	69 0000		0123		Constants
0104:	69 0000		0102		
0111:	00 0001		0000		
0068:	69 0000		0126		
0091:	00 0000		0000		

PAGE _____ OF _____

IBM

Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: 00: (C) ⟶ A WRITTEN BY:

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0026	AU	10	0029	0034	
0034	LD	69	0046	8003	Store C_2 in A_2
8003	ST D	24	A_2	0121	
0121	SU	11	0055	0059	
0059	LD	69	0022	8003	Store C_1 in A_1
8003	ST D	24	A_1	0124	
0124	BR OV	47	0054	0054	Reset overflow circuit
0029:	25 0001	0121			Constants
0055:	00 0000	9997			

PROBLEM: 01: (A)+(B) ⟶ C WRITTEN BY:

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0000	RAU	60	0005	0009	
0009	AU	10	0013	0017	$C_1 = A_1 + B_1 \longrightarrow 0022$
0017	ST U	21	0022	0025	
0025	RAU	60	0028	0033	
0033	AU	10	0037	0041	$C_2 = A_2 + B_2 \longrightarrow 0046$
0041	ST U	21	0046	0049	
0049	BR OV	47	0120	0054	Test for overflow
0120	STOP	01	0000	0120	

PROBLEM: 02: (A)-(B) ⟶ C WRITTEN BY:

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0001	RSU	61	0005	0010	
0010	AU	10	0013	0018	$C_1 = A_1 - B_1 \longrightarrow 0022$
0018	ST U	21	0022	0075	
0075	RSU	61	0028	0033	
0033	AU	10	0037	0041	$C_2 = A_2 - B_2 \longrightarrow 0046$
0041	ST U	21	0046	0049	
0049	BROV	47	0120	0054	Test for overflow
0120	STOP	01	0000	0120	

PROBLEM: 03: (A)x(B) ⟶ C WRITTEN BY:

LOCATION OF INSTRUCTION	OPERATION ABBRV.	OPERATION CODE	ADDRESS DATA	ADDRESS INSTRUCTION	REMARKS
0002	RAU	60	0005	0109	
0109	MULT	19	0013	0119	$A_1 B_1 \longrightarrow 0057$
0119	ST U	21	0057	0110	
0110	RAU	60	0013	0070	
0070	MULT	19	0028	0115	$A_1 B_2 \longrightarrow 0071$
0115	ST U	21	0071	0024	
0024	RSU	61	0028	0084	
0084	MULT	19	0037	0118	$A_1 B_1 - A_2 B_2 \longrightarrow 0022$
0118	AU	10	0057	0066	
0066	ST U	21	0022	0078	
0078	RAU	60	0037	0047	
0047	MULT	19	0005	0112	$A_1 B_2 + A_2 B_1 \longrightarrow 0046$
0112	AU	10	0071	0042	
0042	ST U	21	0046	0054	

IBM
Trade-Mark

IBM TYPE 650 PROGRAM SHEET

FORM NO. 22-6181-0
PRINTED IN U.S.A.

PROBLEM: 04: (A) / (B) $\longrightarrow$ C WRITTEN BY: _____

LOCATION OF INSTRUCTION	OPERATION		ADDRESS		REMARKS
	ABBRV.	CODE	DATA	INSTRUCTION	
0003	RSU	61	0013	0117	
0117	MULT	19	0028	0098	$-A_1 B_2 \longrightarrow 0105$
0098	ST U	21	0105	0108	
0108	RAU	60	0013	0095	
0095	MULT	19	0005	0103	$A_1 B_1 \longrightarrow 0061$
0103	ST U	21	0061	0127	
0127	RAU	60	0037	0092	
0092	MULT	19	0005	0089	$A_2 B_1 \longrightarrow 0097$
0089	ST U	21	0097	0100	
0100	RAU	60	0005	0062	
0062	MULT	19	8001	0107	$B_1^2 \longrightarrow 0064$
0107	ST U	21	0064	0082	
0082	RAU	60	0037	0093	
0093	MULT	19	0028	0096	$A_2 B_2 \longrightarrow 0094$
0096	ST U	21	0094	0048	
0048	RAU	60	0028	0086	
0086	MULT	19	8001	0113	$B_1^2 + B_2^2$
0113	RAL	65	8003	0006	
0006	AL	15	0064	0069	
0069	BRNZU	44	0073	0081	Is $B_1^2 + B_2^2 \geqslant 1$?
0073	SRT	30	0001	0085	
0085	ST L	20	0090	0043	$B_1^2 + B_2^2 \longrightarrow 0090$
0043	RAL	65	0097	0101	
0101	AL	15	0105	0012	$A_2 B_1 - A_1 B_2 \longrightarrow 0032$
0012	SRT	30	0001	0072	
0072	ST L	20	0032	0039	
0039	RAL	65	0094	0099	
0099	AL	15	0061	0065	$A_1 B_1 + A_2 B_2 \longrightarrow 8003$
0065	SRT	30	0001	0074	
0074	RAU	60	8002	0036	
0081	ST L	20	0090	0044	$B_1^2 + B_2^2 \longrightarrow 0090$
0044	RAU	60	0097	0052	
0052	AU	10	0105	0128	$A_2 B_1 - A_1 B_2 \longrightarrow 0032$
0128	ST U	21	0032	0088	
0088	RAU	60	0094	0004	$A_1 B_1 + A_2 B_2 \longrightarrow 8003$
0004	AU	10	0061	0036	
0036	DIV RU	64	0090	0116	$C_1 = (A_1 B_1 + A_2 B_2) / (B_1^2 + B_2^2) \longrightarrow 0022$
0116	ST L	20	0022	0079	
0079	RAU	60	0032	0087	
0087	DIV RU	64	0090	0114	$C_2 = (A_2 B_1 - A_1 B_2) / (B_1^2 + B_2^2) \longrightarrow 0046$
0114	ST L	20	0046	0054	

PAGE _____ OF _____

INTERNATIONAL BUSINESS MACHINES CORPORATION
590 Madison Avenue, New York 22, N. Y.

Form No. 34-6367-1-2M-A

Litho in USA