THE A-2 COMPILER SYSTEM

OPERATIONS MANUAL

15 November 1953

The A-2 Compiler System Operation Manual is a working paper intended to provide the reader (assumed to be a UNIVAC programmer) with all the information necessary to make use of the existing system. The descriptive material has been included only as necessary to introduce concepts or terminology peculiar to automatic programming or to the A-2 System.

The A-2 Compiler System has been developed by Richard K. Ridgway and Margaret H. Harper under the direction of Dr. Grace M. Hopper, Programming Research Section, Electronic Computer Department, Remington Rand, Inc.

TABLE OF CONTENTS

GENERAL DESCRIPTION

The A-2 Compiler is a programming system for UNIVAC which produces, as its output, the complete coding necessary for the solution of a specific problem. If the problem has been correctly described to the compiler the coding will be correct and checked (by UNIVAC) and the program tape may be immediately run <u>without any debugging</u>.

The compiler must have available a library of subroutines. Since the A-2 Compiler was designed primarily for use in mathematical problem solving, the only library presently available is a two-word floating-decimal library containing the arithmetic, sin, cos, arctan, exponential and nth root routines. Several non-computational routines are also included as well as generalized routines for generating the data-handling subroutines. Experience gained in working with earlier compilers advised against the inclusion of specialized data-handling routines in the library.

The coding necessary for the solution of a specific problem is ordered by the programmer in a pseudo-code. This pseudo-code is called "information", and it is the information which tells the compiler how to proceed just as C-10 Code tells UNIVAC how to proceed. This pseudo-code is a new language which is easier to learn and much shorter and quicker to write. Logical errors are more easily found in information than in UNIVAC coding because of the smaller volume.

The A-2 Compiler produces a "program" which utilizes the memory as follows:

000 - 059	initial and special read instructions
060 - 179	data blocks (usually input)
180 - 239	working storage

240 - 779	compiled program (9 blocks per segment)
780 - 899	floating decimal arithmetic
900 - 939	common constants
940 - 999	data block (usually output)

The nine block section of compiled program is called a "segment". Additional segments are read into this part of the memory if the compiled program exceeds nine blocks. Control may be transferred forward or backward from one segment to another as required by the problem.

Relatively simple changes in the compiler will make it applicable for problems needing more data or working storage.

A compiled program may approach the best hand-tailored coding, and for a large class of problems, the A-2 Compiler, with suitable libraries, will cut programming and debugging time to a point where a slight inefficiency in running time can be tolerated.

ELEMENTS OF THE A-2 COMPILER SYSTEM

1. COMPILER

The executive routine whose function is to coordinate the other elements of the programming system.

2. LIBRARY

An ordered set of subroutines and generative routines. Since the subroutines are stored in the library in relative form, they must be transformed before they can be used.

3. GENERATED LIBRARY

A set of subroutines prepared for a specific problem as ordered by the information. This library is built up from two sources.

a) Coding produced from specifications in the information by generative routines which are in the library.

b) New coding peculiar to this problem and of no value in other problems. This is merely copied from element 4 and built into a subroutine.

4. INFORMATION

An ordered set of instructions stated in a pseudo-code. These instructions fully describe a solution to the problem specified in a language which the compiler understands.

5. RECORD

A list in operation number order prepared by the compiler as an intermediate output, necessary in a later stage of compilation for control transfers and desirable for the programmer as a general check of the compilation or as a locator of specific sections of the final output program. There is an item in the record list for each operation which contains:

a) Call-word of subroutine used in operation;

b) Operation number;

c) Segment serial number;

d) Starting line number within segment.

6. RUNNING PROGRAM

The complete set of instructions in C-10 Code for solving the problem; the final output of the compilation process.

RULES FOR SUBROUTINE CONSTRUCTION

The subroutine is the logical unit or building-block. It operates in a specific way on input data, arguments, to produce results. The compiler forces certain limitations on the form of subroutines.

The requirements for subroutines to be used with the A-2 Compiler are stated below. Those marked with an asterisk (*) are requirements peculiar to the two-word floating-decimal library.

1. Subroutines are written in relative coding starting in line 000 of the subroutine which contains the call-word. The call-word consists of four 3-digit fields.

   xxx 111 f00 00⁄

   xxx = The identification code (alphabetic) of the subroutine,

   111 = The number of lines in the subroutine, exclusive of the call-word and sentinel word (word of ignores) at the end.

   * f00 = A floating-decimal point subroutines.

   00⁄ = A sentinel which serves to identify the start of a subroutine.

2. Line 001 of the subroutine is the only entrance (exclusive of R U or generalized overflow reentries).

3. In referring to arguments, 1RG, 2RG, etc. are used instead of memory locations; i.e., B 1RG. The output of the subroutine, the results, are referred to by 1RS, 2RS, etc.

4. Normal exit from a subroutine is considered to occur when the line preceding the ending sentinel is the last line in the subroutine to be executed, or when control is transferred from any line in the subroutine to the ending sentinel. Under these conditions the next line to be executed will be the entrance line of the next compiled subroutine.

5. Any exit from the subroutine other than one of the type described above (4), is called a controlled exit and is written T 1CN or U 2CN, etc. A skip must precede such instructions (i.e., 000000 Q 3CN).

6. References to other lines within the subroutine, require the use of an "M" in the digit position (digit 3 or 9) preceding the 3-digit relative line number.

7. A set of the most used constants will be the memory in locations 900 - 939 when the problem is being run, and these may be referred to by fixed memory locations (see list of Permanent Constants). Constants not on this list must be included in the subroutine.

* 8. There are ten temporary storage locations in 890 - 899 which may be used by any subroutine.

* 9. The floating decimal arithmetic routines are fixed in the memory in locations 780 - 889. Any subroutine can use this section by placing A in 890, 891 and B in 892, 893, and by selecting the proper R U instruction. A and B are floating-decimal operands with integers in even and exponents in odd-numbered locations.

A + B = C	R 810	U 780
A - B = C	R 810	U 828
A x B = C	R 810	U 847
A ÷ B = C	R 810	U 830

C is the result in 894, 895.

* 10. This section is also available to normalize numbers if the operand is placed in 894, 895, zero in rL and R 810 U 808. The result will be placed in 894, 895.

* 11. All subroutines must consist of an even number of lines excluding call-number and ending sentinel.

Example: (non-functional)

000	XYZ012	F0000/	Call-word
001	V 1RG	W 890	1RG = A
002	V 3RG	W 892	3RG = B
003	R 810	U 847	A x B = C
004	V 894	W 1RS	1RS = C
005	B 894	L 911	
006	00 000	T 1CN	if C > 0 —> Controlled exit
007	A-M010	HOM011	Internal subroutine references
008	V 894	QOM012	
009	W 1RG	UOM013	Normal exit
010	000000	000009	Constant
011	[	]	Temporary storage
012	W 3RS	00 000	Normal exit
013	//////	//////	Ending sentinel

PERMANENT CONSTANTS

900	R 904	U 901	This goes to 000
901	C 905	B 904	
902	A- 939	C 904	Generalized Overflow
903	B 905	00 000	
904	[00 000	U (~~111~~)]	
905	[contents of rA]		
906	000800	000000	
907	003000	000000	
908	002000	000000	Required by the special read instructions
909	001000	000000	
910	040000	000000	
911	000000	000000	plus zero
912	-00000	000000	minus zero - sign extractor
913	010000	000000	.1, SL_1
914	-10000	000000	-.1, SR_1
915	001000	000000	.01
916	000001	000000	unit, left instruction
917	000000	000001	unit, right instruction, counter unit, 1×10^{-11}
918	000001	000001	unit, both instructions
919	020000	000000	.2

920	(unused)		
921	050000	000000	.5
922	099999	999999	$\simeq$ 1
923	(unused)		
924	016666	666667	1/3!
925	041666	666667	1/4! x 10
926	083333	333333	1/5! x 10^2
927	013888	888889	1/6! x 10^2
928	019841	269841	1/7! x 10^3
929	024801	587302	1/8! x 10^4
930	000000	000000	
931	027557	319224	1/9! x 10^5
932	078539	816340	$\pi/4$
933	015915	494309	$1/2\pi$
934	025052	108385	1/11! x 10^7
935	020876	756988	1/12! x 10^8
936	000111	000000	address extractor left
937	000000	000111	address extractor right
938	043429	448190	$\log_{10}e$
939	000000	000001	CF constant

RULES FOR GENERATIVE ROUTINES

Some routines in the library are called generative routines rather than subroutines. If the information calls for one of this class of routines, the normal compilation process is interrupted and the generative routine is executed rather than copied. The requirements placed on these generative routines by the compiler are as follows:

1. The first line of the generative routine is 360. This must be a call-word identifying the generator, i.e., GMIOOO FOOOO1. Only a generative routine may be identified by a call-word, beginning with "G".
2. Line 361 is the entrance line. The compiler will read the 360 - 419 block into the memory and then transfer control to line 361 leaving the input register empty.
3. The library, including all generative routines, is on Servo 6. If the generative routine is more than a block long it will read itself into 420 - 999 leaving rI empty.
4. The exit line of the generative routine must read (00 000 U 002). When control is transferred to 002 the compiler will be reread into the memory.
5. The specifications for the generative routine are included with the information in the line(s) immediately following the call-word for the generative routines. The section of the compiler coding which picks up successive words of information is used by the generative routine to obtain these specifications. The instruction pair, R 073 U 069, will place successive words of specifications in 103, until the requirements of the generator are met.
6. Each finished line of generated coding should be placed in 101. The instruction pair, R 064 U 059, will enable this line to be transferred

to the output block. Lines of generated coding are counted and completed blocks are written on the generated library tape by the compiler.

7. The output of the generative routine will ordinarily be C-10 Code relative to 000, or may be any modifications thereof which are understood by the compiler. (See rules for Subroutine Construction)

8. The compiler will put a call-word in the 000 position of the first block of generated coding. It will insert a pair of skip orders at the end of the coding if necessary to make it an even number of lines, then insert a word of ignores, and write the routine on the generated library tape.

INFORMATION WRITING

In order to produce a running program for the solution of a problem, the compiler must be supplied with information describing the sequence of operations in the solution. Every running program will have certain standard blocks necessary to perform the functions required by all problems.

There are five blocks written on the running tape before compilation takes place.

Block 1 - An initial read block which will read blocks 2 through 5 into the running program memory in fixed locations for the duration of the problem. This block is replaced in 000-059 by the special read instructions.

Block 2 - A partial block of permanent constants to occupy 900-939.

Block 3 - Floating-decimal arithmetic, 780-839

Block 4 - Floating-decimal arithmetic and temporary storage, 840-899.

Block 5 - Special read instructions, 000-059. This block contains the forward and backward reads which make possible the transfers from one segment to another.

As the running program is executed, the arguments are first placed in the working storage block. Each computational step operates on these arguments and places the results in the working storage block. Accordingly, planning the computation largely consists of planning the use of this block. Each operation, ordered by a call-word, must include words which specify the location of operands and the location where the result is to be placed in working storage.

Information is written, operation by operation, using the following types of call-words with appropriate additional words: standard call-words using three-address form; call-words for subroutines requiring single-address form; generative routine call-words; new coding call-words; and sentinel call-words. These types of call-words are described in detail below.

Three-address Call-words

Many mathematical subroutines call for two or less arguments and one result. In referring to such subroutines, only one word of information is required.

Example:

A + B = C	AAO(A)	(B)(C)
sin x = Y	TSO(x)	OOO(y)

The brackets indicate a three-digit field. The programmer fills in any even number in the 000-058 range, which corresponds to the relative position of this quantity in the working storage block.

NOTE: The floating decimal library includes a translator which expands to single address form this line of information, so if new subroutines are added to the floating-decimal library, their full 12-digit call-words must be added to the list incorporated in the translator. (Block 5 of Floating Decimal Library tape)

Single-address Call-words

Some subroutines require other than two arguments and a result as provided for in the three-address call-words. The information calling for such a subroutine must be written in the long (single-address) form.

Example:

9FC004	F0000[illegible]
18G0hh	hhh(x)
1CH000	0(on [illegible])
2CH000	C(on #B)

The QZO subroutine is a four-line routine in the floating-decimal library. When executed it brings one operand from working storage and tests to see if it is a word of sentinels (Zs). It has two controlled exits, (1CN and 2CN) depending upon the results of this test. (h is the current operation number of the sequence, (x) is the address of the argument in working storage, (op # B) is the operation number to which control is to be transferred if the test is met, and (op # A) is for all the other cases.)

Generative Routine Call-words

An operation calling for a routine of the generative type requires three or more lines of information.

Example:

GMI000	000002
ITEM04	WS.006
SERVO2	BLOCKA
1RGOhh	hhh000

The first line is a call for the GMI generative routine. The 002 indicates that this routine requires two lines of specifications. The next two lines are specifications interpreted by the generative routine, and the last line is a "dummy" argument required by the compiler for any operation which cannot be fully described in one line. The dummy argument must include operation number (hhhhh) in the problem sequence.

New Coding Call-words

New coding, if used, requires a minimum of three lines of information.

Example:

OWN.CO	DE.002
810000	820000
830000	900000
1RGOhh	hhh000

The sample problem which follows illustrates all of the different types of information, except the "segment" sentinel.

STATEMENT OF THE OPTICAL RAY PROBLEM

Evaluate the optical rays which will result from passing given rays through given surfaces. The rays are defined by X, Y, Z, L, M, and N. The surfaces are defined by 1/r, n, n', and d.

The equations are as follows:

$$\sin^2 I = [(X/r - 1)^2 + (Y/r)^2 + (Z/r)^2] - [L(X/r - 1) + M(Y/r) + N(Z/r)]^2 \quad (1)$$

$$\sin^2 I' = (n/n')^2 \sin^2 I \quad (2)$$

$$\cos I = (1 - \sin^2 I)^{1/2} \quad (3)$$

$$\cos I' = (1 - \sin^2 I')^{1/2} \quad (4)$$

$$K = (n/n' \cos I) - \cos I' \quad (5)$$

$$p = \frac{[X(X/r - 1) + Y(Y/r) + Z(Z/r) - X}{\cos I - [L(X/r - 1) + M(Y/r) + N(Z/r)} \quad (6)$$

$$\left.\begin{aligned}
X_1 &= X + L_p \\
X' &= X_1 - d \\
Y' &= Y + M_p \\
Z' &= Z + N_p \\
L' &= n/n' \, L + K(X_1/r + 1) \\
M' &= n/n' \, M + K(Y'/r) \\
N' &= n/n' \, N + K(Z'/r)
\end{aligned}\right\} \text{The resultant ray}$$

The given rays are as follows:

	X	Y	Z	L	M	N
1	0	1.245093	.89258	.96126	-.2756435	0
2	0	"	0	"	"	0
3	0	- .840292	.01134	.93969	-.3420274	0
4	0	"	0	"	"	0
5	0	1.245093	.39864	.93969	-.3420274	0
6	0	"	0	"	"	0
7	0	-1.080377	.829435	.99027	-.13916	0
8	0	"	0	"	"	0
9	0	1.245093	.004397	.98027	-.13916	0
10	0	"	0	"	"	0
11	0	- .918856	.053824	.96126	-.2756435	0
12	0	"	0	"	"	0

The given surfaces are as follows:

	1/r	n	n'	d
1	.463823650	1	1.553	.34953530
2	.108705950	1.553	1	.63957197
3	-.0876827	1	1.631	.11948479
4	.453605361	1.631	1	.75944209
5	.17777657	1	1.625	.40187514
6	-.14830923	1.625	1	.857240016
7	0	1	1	0

PROBLEM ANALYSIS

It was decided to use the Floating-Decimal Subroutine Library for the computation.

A cursory glance at the amount of calculation necessary to take one ray through one surface seemed to make it inadvisable to pass all rays through successive surfaces in parallel. To simplify data-handling one tape was prepared with one ray described in each block of data. The surfaces were described, one per block, on a second data tape.

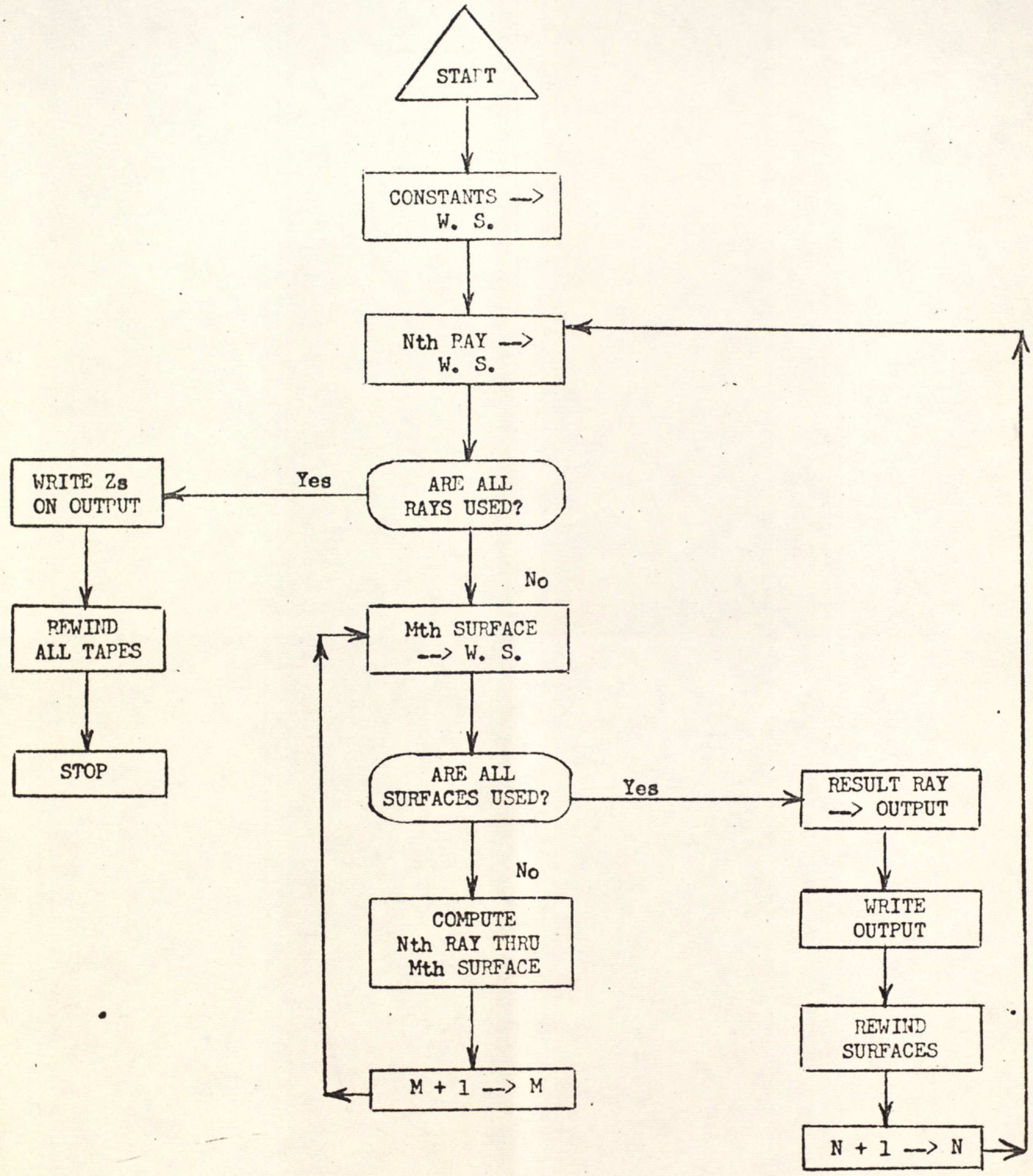
Because the ray tape was longer than the surface tape, it was decided to pass one ray through all surfaces, put out the result, rewind the surface tape, and repeat for successive rays. The flow chart illustrates this plan.

RUNNING PROGRAM - MEMORY ALLOCATION

000 - 059	Initial read and special read instructions (standard)	
060 - 119	Input Block - ray data	
120 - 179	Input Block - surface data	
180 - 239	Working Storage (standard)	
240 - 779	Running Program	
780 - 839	Arithmetic - 2-word floating-decimal	Used by subroutines
840 - 899	Arithmetic and Temporary Storage	Used by subroutines
900 - 939	Permanent Constants	Used by subroutines
940 - 999	Output Block - resultant ray data	

As the running program is executed, the arguments are brought from the working storage block, and the results are deposited in the working storage block. Accordingly, planning the computation consists of planning the use of this block with the capabilities of the subroutine library in mind.

FLOW CHART OF OPTICAL RAY PROBLEM



OPTICAL RAY PROBLEM - USE OF WORKING STORAGE

0	2	4	6	8	10	12
3 - X 45 - X'	3 - Y 46 - Y'	3 - Z 47 - Z'	3 - L 48 - n/n' L 52 - L'	3 - M 53 - n/n' M 56 - M'	3 - N 59 - n/n' N 60 - N'	1 - 1
14	16	18	20	22	24	26
1 - 2	6 - 1/r	6 - n 27 - $(1-\sin^2 I)$	6 - n' 32 - K	6 - d	8 - X/r 9 - (X/r - 1) 49 - X_1/r 50 - $(X_1/r-1)$ 51 - $K(X_1/r-1)$	10 - Y/r 54 - Y'/r 55 - K Y'/r
28	30	32	34	36	38	40
11 - Z/r 57 - Z'/r 58 - K Z'/r	12-$(X/r-1)^2$=A 15-A+B 24-n/n'	13-$(Y/r)^2$=B 16-A+B+C 25-$(n/n')^2$ 30-cos I' 34-Y Y/r=H 37-G+H+I	14-$(Z/r)^2$=C 23-$\sin^2 I$ 29-cos I 40 - ρ 43 - Nρ	17-L(X/r-1)=D 20-D+E 26-$\sin^2 I$ 31-n/n'cos I 35-Z Z/r = I 38 - J	18-M Y/r=E 21-D+E+F 39-0 42-Mρ	19-N Z/r=F 22-$(D+E+F)^2$ 28-1-$\sin^2 I$ 33-K(X/r-1)=G 36 - G+H 41 - Lρ 44 - X_1

An outline of the data-handling, computation, and control steps for the problem solution is given below:

A. The given equations contain certain constants which can be moved into the working storage block and left there.

Op # 0 Read constants into input block

Op # 1 Move constants to working storage

B. The flow chart next shows a ray being moved to working storage.

Op # 2 Read ray into input block

Op # 3 Move ray to working storage

C. The next box calls for a test to see if all rays have been used.

Op # 4 Test for sentinel. If not, go on to the next operation.

D. Next, a surface is moved to the working storage area.

Op # 5 Read surface into input block

Op # 6 Move surface to working storage

E. Test to see if all surfaces have been used, and if not, go on to the next operation.

Op # 7 Test for sentinel

Now the computation can be planned and the operations explicitly stated starting with operation 8. The chart represents working storage area for each 2-word floating-decimal number. The odd-numbered locations are not shown because the numbers are arranged so that the exponent follows each quantity, and the two words are always handled together. The numbers at the left in each box are operation numbers.

F. Compute $\sin^2 I$ from equation (1)

Op # 8 Multiply $X \cdot 1/r = X/r$

Op # 9 Subtrace $X/r - 1$

Op # 10 Multiply $Y \cdot 1/r = Y/r$

Op # 11 Multiply $Z \cdot 1/r = Z/r$

Op # 12 Multiply $(X/r - 1)^2 = A$

.
.
.

G. Compute $\sin^2 I'$ from equation (2)

H. Compute cos I from equation (3)

I. Compute cos I' from equation (4)

J. Compute k from equation (5)

K. Compute X_1 and X', Y', Z', L', M', N': the resultant ray.

L. Return and repeat D

Op # 61 Unconditional transfer to Operation 5

M. Entry point from E when ray has passed through all surfaces.

Op # 62 Transfer ray from working storage to output block

Op # 63 Write output block and rewind surface tape for next ray

Op # 64 Unconditional transfer to Operation 2

N. Entry point from C. All rays have passed through all surfaces.

Operation 3 sent sentinels to working storage instead of ray.

Op # 65 Transfer sentinels from working storage to output block.

O. End of Problem

Op # 66 Write sentinel block twice, rewind all tapes and stop.

The foregoing analysis demonstrates the manner in which the problem is broken down into operations, each of which can be executed by a single subroutine. Now all that is necessary is to translate the operation descriptions into A-2 information by providing the proper call-words and associated arguments, results, etc., as required. Normally, information can be written directly from the flow-charts and the equations.

It may be helpful to explain in detail the meaning of the information for a few of the different operations in the optical ray problem (page 26). Operation 0 is a read routine which does not appear in the library. The first line contains the new-coding sentinel, with the number of lines of coding (001) in digits 10, 11, 12. The second line is the line of new coding to be inserted in the running program. The third line is a required dummy line (the "lrg" has no meaning here) with the operation number (00000) in digits 5, 6, 7, 8, 9.

Operation 1 calls for a generative subroutine which produces the proper instructions for data transfers in the memory, the first line contains the call-word to be used in searching the library tape. Digits 10, 11, 12 indicate the number of lines of specifications (001) which follow the call-word. The specifications (second line) indicate that 4 words (040) are to be transferred starting with line 60 (060), to another area of the memory beginning at line 192. The third line is the dummy word again, and indicates Operation 1.

Operation 4 is an example of information in the single-address code. The first line is the call-word, identifying the subroutine. Digits 4, 5, 6 indicate the number of lines of coding (004) in the subroutine when it is inserted in the running program. Digits 7, 8, 9 state that this is part of the floating-decimal library (F00). The ignore in position 12 is a sentinel signifying a call-word. The second line, in digits 10, 11, 12, states the position in working storage of the argument (lrg) required for this subroutine. Digits 5, 6, 7, 8, 9 of the lrg line always contain the operation number (00004) This is a sentinel test routine; two exits are required. These are specified in lines 3 and 4. If the test is not passed the first exit (1cn) is to operation 5 (00005). If the test is passed the second exit (2cn) is to operation 65 (00065).

Operation 8 shows the three-address type of information, requiring only one word to call for the subroutine and specify the operands. Reading from the left, "AMO" designates "arithmetic, multiply" (this will be a floating-decimal multiplication since we are working with that libfary); "000" is the location of the multiplier in working storage, "X" in this problem; "016" is the location of the multiplicand (1/r): and "024" indicates where the result (X/r) shall be placed.

The last word of information, following Operation 66, is the ending sentinel.

INFORMATION FOR OPTICAL RAY PROBLEM

Op.			
0	QWNΔCO	DEΔ001	Read necessary constants from T_2 into 060, leaving rI empty
	120000	300060	
	1RG000	000000	
1	GMM000	000001	Move constants to W. S. block
	000060	040192	
	1RG000	001000	
2	QWNΔCO	DEΔ001	Read ray into 060 block, leaving rI empty
	120000	300060	
	1RG000	002000	
3	GMM000	000001	Move ray to W. S. block
	000060	120180	
	1RG000	003000	
4	QZ0004	F00001	If 000 holds valid ray, go on to Op. 5, if Z's are in 180, go to Op. 65
	1RG000	004000	
	1CN000	000005	
	2CN000	000065	
5	QWNΔCO	DEΔ001	Read surface into 120 block leaving rI empty
	130000	300120	
	1RG000	005000	
6	GMM000	000001	Move surface to W. S. block
	000120	080196	
	1RG000	006000	

7	Q70004	F00001	If 016 holds valid surface, go on to Op. 8, if Z's are in 016, go to Op. 62
	1RG000	007016	
	1CN000	000008	
	2CN000	000062	
8	AM0000	016024	$X \cdot 1/r = X/r$
9	AS0024	012024	form $X/r - 1$
10	AM0002	016026	$Y \cdot 1/r = Y/r$
11	AM0004	016028	$Z \cdot 1/r = Z/r$
12	AM0024	024030	form $(X/r - 1)^2 = A$
13	AM0026	026032	form $(Y/r)^2 = B$
14	AM0028	028034	form $(Z/r)^2 = C$
15	AA0030	032030	$A + B$
16	AA0030	034032	$A + B + C$
17	AM0006	024036	form $L(X/r - 1) = D$
18	AM0008	026038	form $M(Y/r) = E$
19	AM0010	028040	form $N(Z/r) = F$
20	AA0036	038036	$D + E$
21	AA0036	040038	$D + E + F$
22	AM0038	038040	$(D + E + F)^2$
23	AS0032	040034	$(A + B + C) - (D + E + F)^2 = \sin^2 I$
24	AS0018	020030	form n/n'
25	AM0030	030032	form $(n/n')^2$
26	AM0032	034036	$(n/n')^2 \sin^2 I = \sin^2 I'$
27	AS0012	034018	form $1 - \sin^2 I$
28	AS0012	036040	form $1 - \sin^2 I'$
29	RNA018	014034	$(1 - \sin^2 I)^{1/2} = \cos I$
30	RNA040	014032	$(1-\sin^2 I')^{1/2} = \cos I'$

31	AM0030	034036	form n/n' cos I
32	AS0036	032020	(n/n' cos I) - cos I' = K
33	AM0000	024040	X(X/r - 1) = C
34	AM0002	026032	Y(Y/r) = H
35	AM0004	028036	Z(Z/r) = I
36	AA0040	032040	G + H
37	AA0040	036032	G + H + I
38	AS0032	000036	(G + H + I) - X = J
39	AS0034	038038	cos I - (D + E + F) = Q
40	AD0036	038034	j/o = ρ
41	AM0006	034040	form Lρ
42	AM0008	034038	form Mρ
43	AM0010	034034	form Nρ
44	AA0000	040040	X + Lρ = X_1
45	AS0040	022000	X_1 - d = X'
46	AA0002	038002	Y + Mρ = Y'
47	AA0004	034004	Z + Nρ = Z'
48	AM0030	006006	n/n' L
49	AM0040	016024	X_1/r
50	AS0024	012024	X_1/r - 1
51	AM0020	024024	K(X_1/r - 1)
52	AA0006	024006	n/n' L + K(X_1/r - 1) = L'
53	AM0030	008008	n/n' M
54	AM0002	016026	y'/r
55	AM0026	020026	K(Y'/r)
56	AA0026	008008	K(Y'/r) + n/n' M = M'
57	AM0004	016028	Z'/r
58	AM0020	028028	K(Z'/r)
59	AM0030	010010	n/n' N

60	AA0010	028010	n/r' N + K(Z'/r) = N'
61	U00002	F0000/	Go back to Op. 5
	1RG000	061000	
	1CN000	000005	
62	GMM000	000001	Move resultant ray to output block
	000180	120940	
	1RG000	062000	
63	OWN_CO	DE_001	Write output block and rewind surface tape
	740940	630000	
	1RG000	063000	
64	U00002	F0000/	Go back to Op. 2
	1RG000	064000	
	1CN000	000002	
65	GMM000	000001	Move N's to output block
	000180	010940	
	1RG000	065000	
66	OWN_CO	DE_004	Write 2 blks. of sentinels, rewind all tapes and stop
	740940	740940	
	840000	820000	
	830000	810000	
	900000	900000	
	1RG000	066000	
	ØØEND_	INFO.Ø	

A few statistics on this problem may be of interest.

After the flow-chart was drawn, the information was written in one-half a day. The problem was compiled in 2 1/2 minutes, and was solved in 3 1/2 additional minutes. The running program was just over one segment. (This could have been avoided, but it was designed to test the compiler)

The data-handling is cumbersome because the GMM generative routine was the only one in existence at the time.

OPERATING INSTRUCTIONS

1. Mount tapes as follows:

Servo 1	A-2 Compiler	with ring
Servo 2	blank	no ring
Servo 3	blank	no ring
Servo 4	Information	*
Servo 5	blank	no ring
Servo 6	Subroutine library	with ring

* If the information is to be retained, insert a ring, and retain comma breakpoint.

2. Initial read Servo 6, and let it run on continuous.

* When stopped on the comma breakpoint, remove the tape from Servo 4, and substitute a blank with no ring. Release breakpoint.

3. Normal printouts are:

   "end translation"

   "end Sweep 1"

   "end Sweep 2"

   "end compile"

4. The running program is on Servo 4. Put in a ring, label it, and move it to Servo 1 if the problem is to be run immediately.

<u>Error print-outs which may occur during compilation</u>

1. [word of info.]

   RNOT_S TOREDR

   If this print-out occurs before "end of translation" has been written on the supervisory control, the programmer has written a word of information which has been identified as a call-word to be translated, but the complete 12-digit designation of this call-word is not stored in Block 5 of the floating-decimal library tape.

2. [word of info. ✗NOT␣S TORED✗	If this print out occurs after "end of translation" has been printed on the supervisory control, and before "end S-1" has been printed, the programmer has called for a generative routine which is not in the library.
3. ✗INFO␣ WORD?✗ [word of info.]	A word of information has failed to satisfy any of the tests. There may be a transcription error.
4. ✗OP␣SE Q␣OUT✗ [word of info.]	The word of information carries an operation number (h) which is not in the expected sequence. Line 069 is a variable line which is now prepared to pick up this word in the current block of information. (120-179) A search around this location should help identify the offending word if it is not immediately recognizable.
5. ✗SEGME NT␣␣␣✗ ✗TOO␣L ARGE␣✗ [word of info.] [word of info.]	This print-out indicates that the programmer has enclosed more than the 539 allowable lines with the "segment" sentinels. The two words of information (a call-word and first argument) will locate the subroutine which will not fit in the segment.
6. ✗T␣MCH ␣INFO✗ [word of info.]	The compiler can store no more than sixty descriptive words (i.e., argument, control or

result word) applying to a single call-word. This print-out indicates that the limit has been exceeded. Memory location 660 will contain the call-word, and the 660 block contains the words of information that have been stored.

7. R̸NOT␣I N␣LIBR̸
[word of info.]

The word of information is a call-word for a subroutine which is not in the library.

8. R̸NOT␣S TORED R̸
[Line of Subroutine]
[1RG or 2RS or 3CN etc.]

Memory location 660 contains the call-word for the current subroutine. This subroutine has a line which contains 1RS, for example, in lieu of an address. The information ordering this subroutine did not include a word, 1RS000 000(x), stating the specific address for this result.

9. R̸FULL␣ TAPE␣R̸

The running tape is 1955 blocks long. If this is expected, and there is no danger of running off the tape, SCICR to 529 and let it run.

NOTE: As a general rule, after an error print-out, it will be better to correct this information and recompile. A word of zeros can be inserted to block out incorrect words of information.

All subroutines in the floating-decimal library are built to operate on quantities ($\bar{A}$) represented by two computer words (A and a'), where $\bar{A} = A \cdot 10^{a'}$. The number A has a sign and eleven digits to the right of the decimal point; the number a' has the sign of the exponent and eleven digits to the left of the decimal point.

Examples:

Quantity ($\bar{A}$)	Floating-Decimal Form		
- .006	-60000	000000	= A
	-00000	000002	= a'
- 60	-60000	000000	= A
	000000	000002	= a'
+ 3025.5	030255	000000	= A
	000000	000004	= a'
+.030255	030255	000000	= A
	-00000	000001	= a'

Except where specifically mentioned, the subroutines are designed to produce results accurate to at least ten digits for inputs of any size. The subroutines have been computer-tested for at least thirty different operands covering the complete range of allowable inputs.

A-2 FLOATING-DECIMAL SUBROUTINE LIBRARY

Add	AA0004	F00001̸
Subtract	AS0004	F00001̸
Multiply	AM0004	F00001̸
Divide	AD0004	F00001̸
Add to a limit	AAL012	F000T1̸
Change sign	AN1002	F00001̸
Raise variable to a power	APN048	F00001̸
Input Generator	GMI000	F00001̸
Move Generator	GMM000	000001̸
Output Generator	GMO000	F00001̸
Logarithm	LAU118	F00001̸
Root	RNA154	F00001̸
Arctan	TAT118	F00001̸
Cosine	TC0100	F00001̸
Sine	TS0106	F00001̸
Exponential	X+A096	F00001̸
Test for Sentinel	QZ0004	F00001̸
Unconditional Transfer	U00002	F00001̸
Type-in	BTI002	F00001̸
Print-out	YTO004	F00001̸

AA0004 F0000⁄
AS0004 F0000⁄
AM0004 F0000⁄
AD0004 F0000⁄

Operation Performed - floating-decimal arithmetic

$\overline{A} + \overline{B} = \overline{C}$

$\overline{A} - \overline{B} = \overline{C}$

$\overline{A} \times \overline{B} = \overline{C}$

$\overline{A} \div \overline{B} = \overline{C}$

[$\overline{A}$ = A, a' = normalized number and its exponent]

Allowable Inputs

Full floating-decimal range.

Form of Information

AAO(A) (B)(C)
ASO(A) (B)(C)
AMO(A) (B)(C)
ADO(A) (B)(C)

(A) means to insert the relative address of A in working storage (000 - 058). The normalized number must be in the even location with its exponent in the next higher location.

AAL012 F000T1

Operation Performed - floating-decimal add to a limit

$\bar{x} + \Delta\bar{x} \longrightarrow \bar{x}$ if $\bar{x} = \bar{L}_x$, $\longrightarrow$ 2CN

if $\bar{x} \neq \bar{L}_x$, $\longrightarrow$ 1CN

($\bar{X}$ = X, x')

Form of Information

AAL012	F000T1	
1RG0hh	hhh( )	$\bar{X}$
2RG000	000( )	
3RG000	000( )	$\Delta\bar{X}$
4RG000	000( )	
5RG000	000( )	$L_{\bar{X}}$
6RG000	000( )	
1CN000	0(op≠)	
2CN000	0(op≠)	
1RS000	000( )	$\bar{X}$
2RS000	000( )	

The limit of X should be one increment larger than the largest X on which calculations are to be performed.

AN1002 F0000/

Operation Performed - floating-decimal number transfer

- 1RG --> 1RS

Form of Information

AN1(1RG) 000(1RS)

( ) = The working storage location of the quantity involved.

APN048 F0000/

Operation Performed - raise to an integral power.

$U^n = V$

(U, n, and V all in standard two-word floating-decimal form)

Allowable Inputs

n must be a whole number. If n is greater than 9 the routine asks if it should continue, and then stops. Hit the start bar to continue.

Form of Information

A¬N(U) (n)(V)

Note: In a problem involving this routine, n must be included with the input data, or brought into the working storage block separately before this routine is called for.

GMI000 F0000/

Operation Performed

A generative routine which will produce the coding necessary to move successive items from an input block to working storage, reading (non-continuous) from tape initially and after the last item is transferred out of the input block.

Form of Information

GMI000 000002

ITEMSS WS.(w)

SERVOn BLOCKX

1FGOhh hhh000

SS = 1, 2, 3, 4, 5, 6, 8*, 10, 12, 15, 20 or 30

(* 7 items per block, last four words in block filled with zeros)

w = relative working storage address for first word of item. (000 - 059)

if SS > 1; w must be even

if SS > 9; w must be divisible by 10

n = any servo number

X = A or B only, A = 060 block, B = 120 block

h = the operation serial number

ANALYSIS OF THE TWELVE GENERATED SUBROUTINES

SS	1	2	3	4	5	6	8	10	12	15	20	30
of lines	8	8	80	52	70	48	42	8	42	52	16	14
Av. time millisec. for item (read omitted)	2.59	2.67	5.59	4.78	7.07	5.18	6.65	3.00	8.66	12.67	5.44	6.17

Note: The coding produced by this generative routine does not include the sentinel test which will recognize the end of the data.

GMM000 000001

Operation Performed

A generative routine which will produce the coding necessary to move one or more (consecutive) words from one area within the memory to another area within the memory.

Form of Information

GMM000 000001

000(m_1) dd0(m_2)

1RG0hh hhh000

dd = the number of words to be moved

m_1 = the location of the first word

m_2 = the location to which the first word is to go.

Note: If this routine is used to move a quantity into working storage, the absolute location, 180 - 239, must be used rather than the relative location, 000 - 059.

GM0000 F00001

Operation Performed

A generative routine which will produce the coding necessary to move items from working storage to consecutive lines in the output block, writing the output on tape when the block is full.

Form of Information

GM0000	000002
WS.(w)	ITEMSS
ONSERV	0nPLKX
1RG0hh	hhh000

SS = 1, 2, 3, 4, 5, 6, 8*, 10, 12, 15, 20, 30

(* Block will be written after seven items have been placed in it.)

w = relative working storage address for first word of item (000 - 059)

if SS > 1; w must be even

if SS > 9; w must be divisible by 10

n = any servo number

x = not variable, 940 block is only output block

h = the operation serial number.

LAU118 F0000X

Operation Performed - compute the logarithm

Given: U and $\log_{10}A$

Calculate: $\log_A U$

Allowable Inputs

U must be greater than zero

Form of Information

LAU(U) $(\log_{10}A)(\log_A U)$

( ) means the relative working storage address of the quantity called for.

Note: In a problem involving this routine, $\log_{10}A$ must be included with the input data, or brought into the working storage area separately before the routine is called for.

$\log_{10}2$ =	030102	999566	
	000000	.000000	
$\log_{10}e$ =	043429	448190	(in 938)
	000000	000000	
$\log_{10}10$ =	010000	000000	(in 913)
	000000	000001	(in 917)

Accuracy

This subroutine has not been fully checked.

RNA154 F00001

Operation Performed - compute nth root

Given: N and A

Calculate: $\sqrt[N]{A}$

Allowable Inputs

A may be any size, but N must be no greater than nine.

Form of Information

RNA(A) (N)($\sqrt[N]{A}$)

( ) means the relative working storage address of the quantity called for.

TAT118 F0000/

TC0100 F0000/

TS0106 F0000/

Operation Performed - trigonometric functions

arctan A = V

cosine A = V

sine A = V

Allowable Inputs

The input quantity is in radians , and may be of any size.

Form of Information

TAT(A) 000(V)

TCO(A) 000(V)

TSO(A) 000(V)

( ) means the relative working storage addrwss of the quantity called for.

X+A096 F00001

Operation Performed - raise to a power

Given: U and $\log_{10} A$

Calculate: $A^U = V$

Allowable Inputs

The input quantities may be of any size except the multiplication of U times $\log_{10} A$ must not result in an exponent greater than (000000 000010) or less than (-00000 000010).

Form of Information

X+A(U) $(\log_{10} A)$(V)

( ) means the relative working storage address of the quantity called for.

Accuracy

The routine has not been thoroughly computer checked, but the results are accurate as far as wh have been able to check them with existing logarithm tables.

QZ0004 F0000Z

Operation Performed --

Compares an operand with a word of Zs

If ≠ --> 1CN

If = --> 2CN

Form of Information

QZ0004 F0000Z

1NCOhh hhh(w)

1CN000 0(op#)

1CN000 0(op#)

w = relative working storage address of operand

h = the operation serial number

op# = the operation number to which control is to be transferred.

U00002 F0000/

Operation Performed

An unconditional transfer

Form of Information

```
U00002  F0000/

1FG0hh  hhh000

1CN000  0(op#)
```

h = the operation serial number

op # = the operation to which control is to be transferred.

BTI002 F0000/

YTQ004 F0000/

Operation Performed - type in; print out

BTI calls for four type-ins

YTQ prints out four quantities and then stops

Form of Information

BTI(A) (B)000

YTQ(A) (B)000

A = location of first type-in or print-out, A + 1 is assumed to be the location of the second type-in or print-out

B = location of third type-in or print-out, B + 1 is assumed to be the location of the fourth type-in or print-out.

COMMENTS

The system described herein has a certain amount of built-in flexibility. It is the original A-2 Compiler with modifications I and II. Much greater flexibility can be achieved by programmers familiar with the compiler coding. The following remarks will indicate some of the changes which could be made to the existing compiler.

The running program memory allocation should be variable. Some problems might require only one block of compiled program to be memory-contained, with the remainder of the memory free for data, and/or routines fixed in the memory as are the present floating decimal arithmetic routines.

Once a subroutine has been compiled, it should be possible to modify its exits and its references to working storage, so it can be reused with other operands in another portion of the program. As currently written the A-2 Compiler recompiles such a routine each different time it is called for.

Compilation time can be shortened a great deal. A call for a floating-decimal arithmetic routine necessitates a library search to obtain the proper R U and transfer orders. The compiler could recognize this type of call-word and generate the necessary orders. The library should have generative routines in one section and other routines in a separate section since the two types are searched for at different times in the compilation process. However, it should be emphasized that since compilation time must be compared with human programming time, and since the product is a ready-to-use, checked, program tape, compilation time is not significant.

The compiler should make the working storage memory assignments. This will allow the programmer to use mathematical symbols in information-writing, and relegate to the computer the planning of the working storage area.

Continuous reads on data tapes can now only be used on short problems, because the programmer may not know whether or not the running program will fit in one segment.* The whole problem of data-handling needs a great deal more study.

* Multi-segment programs require that rI be available at all times for the reading of other segments.

GLOSSARY

Note: This list has been included as a reference for terms used in the manual which may be confusing and/or new to the reader. The definitions given pertain to the use of these terms in the manual, although conventional usage and definitions have been considered where possible.

Argument .... An operand or unit of data which must be specified and located prior to execution of an operation.

Call-word .... A computer word used as a descriptive code and identification key in referencing routines in the compiler library.

Executive Routine ..... A program which operates upon input instructions or specifications usually written in a language more convenient that computer code, and produces coding which will be executed by the computer in producing results, either computational or data-processing.

Generated Coding .... See Generative Routines

Generative Routines .... (Generators) Routines which operate upon specifications describing the process for which the computer is to be programmed, to produce (generate) machine (C-10) coding. As an element of an automatic programming system, generative routines must be written in conformance with the requirements of the system, and the generated coding they produce must also conform.

Operation .... The smallest sub-division of a program requiring a single reference to one of the elements in the compiler system, such as a subroutine. Operations are ordered by the programmer and are numbered serially throughout a given program.

Pseudo-code .... Computer words other than the machine (C-10) code, design with regard to facilitating communications between programmer and computer. Since a pseudo-code can not be directly executed by the computer, there must be programmed a modification, interpretation or translation routine which converts the pseudo-codes to machine instruction and routines.

Record .... Intermediate output of compilation process which contains segmenting and memory allocation information in operation number sequence.

Relative Coding .... Coding which may be systematically inserted in any portion of the memory. It depends on the relationship between the memory addresses it contains, rather than fixed, explicit addresses. A compiling system will be capable of automatically transforming relative coding to a fixed location in the memory.

Result .... A numerical quantity or unit of data resulting from the execution of an operation.

Running Program (tape) ..A set of machine (C-10) instructions which, when executed by the computer, produce results. With respect to automatic coding, the running program (tape) is the output of executive type routines, ile., compilers.

Segment .... A part of the complete running-program which can be entirely stored in the memory. A program which exceeds memory capacity is divided into segments. [Segments may be designated by the programmer for optimum efficiency, but if they are not specified, A-2 will automatically segment a program.]

<u>Specifications</u>.......... Computer words other than machine code designed for the convenience of the programmer in describing the characteristics of a problem or process to an executive routine. The executive routine will operate upon the specifications to produce the required machine (C-10) coding.

<u>Subroutine</u>.............. A unit of coding which is complete and performs a specific function and which may be useful in different programs. As one element of an automatic programming system it must conform to the arbitrary requirements of the system which transforms it for a specific applications.

<u>Temporary Storage</u>....... A part of the memory set aside for use during the execution of a subroutine. All temporary storage registers are considered available at the start of each subroutine.

<u>Working Storage</u>......... That part of the memory set aside for the use of <u>all</u> sub-routines as required. Data which is to pass from one subroutine to another is stored in working storage.