JILL CARTER

R3

report

Runcible I

Vol. I Series V

Revised Edition

March, 1959

Staff

Computing Center

CASE INSTITUTE OF TECHNOLOGY
UNIVERSITY CIRCLE • CLEVELAND 6, OHIO

Runcible I

**Revised Unified New Compiler with
IT- Basic Language Extended**

"They dined upon mince and slices
of quince,
Which they ate with a runcible
spoon;
And hand in hand, on the edge of
the sand,
They danced by the light of the
moon, —
The moon;
They danced by the light of the
moon."

—Edward Lear

COMPUTING CENTER
Case Institute of Technology
Cleveland, Ohio

Abstract

In recent years several compilers have arisen for various computers as steps towards automation in programming. A compiler, such as RUNCIBLE I, comes as close as possible to the ideal of transforming a flow chart directly onto cards and into a finished program. Coding for Runcible I is done in "IT-language," originally developed by Dr. A. J. Perlis, Mr. J. W. Smith, and Mr. H. R. Van Zoeren at Carnegie Institute of Technology, Pittsburgh. Although the original IT-language will still work correctly with this compiler, it has been expanded and simplified somewhat for greater convenience.

Runcible I will run on a basic 2000-word 650 which has only an alphabetic attachment as an extra feature, but if desired it will give an output program which utilizes the 653 floating point hardware, index registers, and core storage. Neither the special character attachment or any additional selectors are needed by the 533 plugboard, and the input statements may be easily listed on a 402 tabulator. All routines will work with a 12-word buffer if necessary. This program is designed to work together with the SOAP III assembly program, but the user has the option of skipping the SOAP phase and obtaining a five-instruction-per-card machine language deck directly. RUNCIBLE I also introduces a number of devices which greatly facilitate the debugging of a program which is in its initial stages. Hand optimized subroutines used with the running program compare favorably in speed and accuracy with those of any alternative general purpose system for the 650 known to date.

The major purpose of the compiler is thus to enable a program to be run speedily with but a minimum of preparation and headache on the programmer's part. It is specifically designed to be used with algebraic equations arising in typical mathematical and engineering problems.

Acknowledgments

RUNCIBLE I is a project of such a magnitude that it is impossible to acknowledge everyone who has contributed to the effort. As with Compiler II, the entire blame for the whole operation belongs with Mr. Frederick Way III, Assistant Director of the Computing Center, who with George Haynam made the major decisions on the philosophy to be carried out. Bill Lynch did the greater part of the work of converting the compiler to floating point instructions and expanding the language, and he worked hand in hand with the author on producing the final deck in its multipurpose form; he has also done some of the work on extensions. George Haynam is responsible for the clocking feature and most of the coding in the basic packages, and he was also in charge of board wiring. He and George Petznick, Jr. have prepared a large number of extensions which produce extremely accurate results. Don Buyansky drew up the 533 board wiring diagram, and almost every other member of the staff at the computing center has had his hand in at one time or another. Donald Knuth, author of this manual, added the one pass feature and wrote the relocation routine; he also designed the 24-operation-mode program and deck, helped to hand optimize the basic packages and prepared the comments on the symbolic program listings. Basic packages were optimized with HAND SOAP, a modification of SOAP III.

NOTE: Any program which is written correctly in original IT-language or for the SOAP II Compiler (Case Institute, November 1957) will be processed correctly by this routine without any changes except for slight substitutions in the header card and the data cards. No major revisions of RUNCIBLE I are anticipated for several years; now in preparation is "Runcible 0" which will process directly from statements to answers without any intermediate punching of a machine language program.

R U N C I B L E I

INTRODUCTION. The user of RUNCIBLE I need not be familiar with any features of the 650 other than those to be described in this manual. A compiler such as this program is probably the simplest and most convenient way to solve problems on a computer without actually studying the complexities of the machine. Those who state problems should not be required to code them; RUNCIBLE I takes the place of a professional programmer and does the job itself.

THE LANGUAGE. There are many types of languages which enable communication between people. Besides spoken tongues, there are the languages of music, of mathematics, and so on. RUNCIBLE I understands "IT-language," which is very close to both English and mathematical formulae. The program which will solve a problem is put onto cards in "IT-language;" the compiler will transform these into an intermediate "SOAP language" and punch it onto cards in this form. These cards in turn are processed by the SOAP III program to produce the "machine language" which the computer understands directly.[1] The compiler programmer need not know any more of SOAP language than is given in this manual. The first section of the manual deals with the vocabulary and syntax of IT-language.

NUMBERS: FIXED POINT AND FLOATING POINT. An IT-language program is a set of numbers, words, and symbols arranged to give a complete description of some problem. The numbers which are manipulated by RUNCIBLE I are of two categories: fixed point and floating point.

Fixed point numbers are integers whose numeric value is less than one billion. They may be positive or negative, but may never take on fractional values. They are used primarily as indices or subscripts and only rarely for arithmetic calculations.

Floating point numbers are normally used for arithmetical operations because of their greater flexibility. Floating point numbers can be zero or range from 10^{-50} to 10^{49} in numerical value,[2] and are always rounded to eight significant figures.

---

1. If desired, the SOAP phase may be bypassed; see Operation Modes.
2. When using 653 instructions (see Operation Modes) they range from 10^{-51} to 10^{48}.

VARIABLES. Variables normally called x, y, z, μ, etc. are always given the names I, Y, or C with subscripts when using RUNCIBLE I; e.g., I_1, Y_1, Y_2, C_{10}, and so on. (The subscript is to be written next to the letter as I1, Y2, C10.) Every variable must have a subscript.

I- variables are indexing variables and are always given fixed-point (integral) values. Y- and C- variables are the problem variables and always take on floating-point values. There is no difference in arithmetic between a Y- variable and a C- variable; any distinction is made by the programmer himself as an aid to keeping the names of his variables in order.

When working on the problem the machine keeps a table similar to the table below. If, for example, the program uses variable Y3, the machine looks into the table and sees that the floating point number -1.55 is to be used in the calculation. It is possible to have variable subscripts, Y_{I_1}, C_{I_4}, etc. Y_{I_1} (written YI1) will be the variable Y2 in the example of the table below, since I1 equals 2. In a similar manner, CI4 = C1 = 1000.[1] Subscripts must be fixed point numbers or fixed point variables.

TABLE

	I	Y	C
0	+ 5	+ 3.0000000	- 3.1415927
1	+ 2	- 10.000000	+ 1000.0000
2	+ 1	0	$-6.02000000 \times 10^{23}$
3	+ 12	- 1.5500000	+ 2.7182818
4	+ 1	+ 6155397.0	
5	- 1040	$+ 4.1400000 \times 10^{-34}$	
6	+ 704		

1. YII1 means $Y_{I_{I_1}}$ and in this case YII1 = YI2 = Y1 = -10. This hierarchy of subscripts may proceed to any depth. Compound subscripts of the form Y(I2+I3) are also allowable; other examples of permissible subscripts are I(II3xI9) and C[I(I1+2)] and even Y"SINEF,C3". An alternate form for subscripts used in matrices is described in Appendix II. Note that all legal subscripts must be fixed point.

CONSTANTS. Constants used in a RUNCIBLE program are written as follows: Floating-point constants are written as some number by a power of ten, with a "B" to indicate what power of ten.[1] Examples:

6.02 B 23 means 6.0200000×10^{23}

1066 B-11 means $1066.0000 \times 10^{-11}$

The B may be omitted as long as the number can be represented in eight digits without multiplying by a power of ten; for example,

3.1415927; .15625; 0.

When the B is omitted the number must contain a decimal point.

Fixed-point constants are written in the ordinary manner; "123" for example. Notice that "123" is a fixed point constant while "123." is floating point because of the decimal point.

Any number of constants may be used in a program.[2]

MATHEMATICAL OPERATIONS. Mathematical expressions can be converted almost at sight directly into IT-language, but it is best to describe this process carefully so there will be no chance of a mistake.

Five binary operations are standard for RUNCIBLE expressions: addition, subtraction, multiplication, division, and raising to a power; they are called binary because they involve two quantities. These operations are written in the following manner (letting C1 represent x and Y1 stand for y):

Math. language:	IT-language:
$x + y$	C1 + Y1
$x - y$	C1 - Y1
xy	C1 x Y1
$\frac{x}{y}$	C1 / Y1
x^y (x to y power)	C1 P Y1

Note how the IT-language scheme enables the writing of a formula as a string of symbols all on one line. Putting the operations together, with a few constants, we have:

Math. language: $$\frac{[(10.4 \times 10^{28}) + x^4]y}{x - y}$$

IT-language: {[10.4B28 + (C1 P 4)] x Y1} / (C1 - Y1)

1. A floating point exponent (such as 5 B 3.14) is not permitted.
2. Actually 700 is the maximum allowable number but this may safely be considered "infinite" for programs processed by RUNCIBLE. If more than 100 constants are used the last 100 are punched out and the program then continues where it left off.

Notice the use of parentheses in the last example.

When more than two quantities are involved, parentheses are needed to avoid ambiguity. Parentheses are very important in IT-language because there is no difference in priority or scope between any of the binary operations as is usually understood in everyday mathematics. For example, the expression Y1 x Y2 + C3 would mean $Y_1Y_2 + C_3$ to most people but RUNCIBLE understands it to mean $Y_1(Y_2 + C_3)$. As another instance, in order to write x^4y we would have to write (C1 P 4) x Y1 because C1 P 4 x Y1 would mean x^{4y}. There is a very simple MORAL to be learned from this: always place parentheses around the quantities in an IT language expression until it can mean only one thing. This cannot be stressed too heavily, for the vast majority of programming errors for RUNCIBLE are caused by neglecting to place parentheses in the proper way.[1]

As a reference, these are the rules by which RUNCIBLE determines to which quantities each binary operation applies.

1. On the left-hand side of the symbol, the binary operation applies to the variable or constant immediately at its left, unless the character next to the operator is a right parenthesis. In the latter case, the entire quantity between the right parenthesis and its matching left parenthesis is used.

2. On the right-hand side of the symbol, everything up to the end of the expression or to the first unmatched right parenthesis is used.[2]

Here are some examples showing application of these rules.

IT-language:		Math. language:
Y1 / Y2 + Y3	means	$\frac{Y_1}{Y_2 + Y_3}$
(Y1/Y2) + Y3	means	$\frac{Y_1}{Y_2} + Y_3$
Y1P Y2 + 3x Y3	means	$Y_1^{Y_2+3Y_3}$
(Y1PY2)+3xY3	means	$Y_1^{Y_2} + 3Y_3$
(Y1PY2+3)xY3	means	$(Y_1^{Y_2+3})Y_3$
[(Y1PY2)+3]xY3	means	$(Y_1^{Y_2+3})Y_3$
4/Y(I1+I2)-I3	means	$\frac{4}{Y_{(I_1+I_2)}-I_3}$

1. Parentheses, braces, and brackets may be nested within each other not more than nine deep but this limit is rarely met in practice.

2. Exception: the minus (-) is treated first as a unary operator (see below) and then as a plus (+) binary operator.

Remember, it is always best to add parentheses to make your intentions unquestionable--better safe than sorry. Adding parentheses where they are unnecessary will not bother RUNCIBLE and no harm will be done.

In addition to the five binary operations there are two standard "unary" operations: taking the absolute value and taking the negative of a quantity. Letting C1 <=> x and Y1 <=> y as before, we have

Math. language:	IT-language:
$\|x\|$	A C1
$\|x+y\|$	A(C1 + Y1)
$-x$	- C1
$-(x+y)$	-(C1 + Y1)
$-\|y\|$	- A Y1

The unary operator A or - applies to the variable or constant at the immediate right of the symbol only, unless the character to its right is a left parenthesis. In the latter event, it operates on the entire quantity inside the left parenthesis and the matching right parenthesis. Examples:

IT-language:	Math. language:
AY1 + C1 / C2	$\|Y_1\| + \frac{C_1}{C_2}$
A(Y1 + C1) / C2	$\frac{\|Y_1 + C_1\|}{C_2}$
AY1 P -(AC1-Y2)	$\|Y_1\|^{-(\|C_1\|-Y_2)}$
AY(I3+1) + 2	$\|Y_{I_{3+1}}\| + 2$

Arithmetic.[1] Runcible always "unwinds" statements by doing the innermost parentheses first, just as in ordinary high school algebra. Once RUNCIBLE is into the innermost parenthesis it drops the "ordinary" rules and does

---

1. It is possible to intermingle fixed and floating constants and variables--RUNCIBLE will not get mixed up; however this is generally inefficient and of doubtful utility.

2 mistakes here

things from the RIGHT!

Examples:

$Y_1 \leftarrow 3 \times 4 + 6$; Y_1 is given the value 30.0

$Y_1 \leftarrow 6 + 3 \times 4$; Y_1 is given the value 18.0

$Y_1 \leftarrow 6 + (3 \times 4)$; Y_1 is given the value 18.0

$Y_1 \leftarrow (3 \times 4) + 6$; Y_1 is given the value 18.0

$Y_1 \leftarrow 6 - 3 \times 4 + 2$; Y_1 is given the value -12.0

$Y_1 \leftarrow 6 - (3 \times 4) + 2$; Y_1 is given the value -8.0

$Y_1 \leftarrow 6 / 4 / 2$; Y_1 is given the value 3.0 ? $\frac{1}{3}$ or $\frac{1}{2}$

$Y_1 \leftarrow (6 / 4) / 2$; Y_1 is given the value 0.0 ? .75

$Y_1 \leftarrow (6.0 / 4.0) / 2.0$; Y_1 is given the value 0.75

$Y_1 \leftarrow (6.0 / 4.0) / 2$; Y_1 is given the value 0.75

$Y_1 \leftarrow (6 / 4) / 2.0$; Y_1 is given the value 0.5

From the above examples we can see that RUNCIBLE attempts to use the arithmetic (fixed or floating) of the innermost parentheses and then tries to keep on using that kind of arithmetic until it must do floating point - at this stage, and from this stage on, it does everything floating point at this parenthesis level. The final substitution into the left hand side is always forced to agree with the arithmetic of the left hand side.

Examples:

$I_2 \leftarrow 6.0 + 3 - 7 \times 2.0$; I_2 is given the value -5

$I_2 \leftarrow 6.0 + 3 - 7 \times 2$; I_2 is given the value -5

$Y_2 \leftarrow 6.0 + 3 - 7 \times 2$; Y_2 is given the value -5.0

$C_6 \leftarrow 6.0 + 3.0 - (1 / 4)$; C_6 is given the value 9.0

$I_6 \leftarrow 6.0 - 3.0 - (1 / 4)$; I_6 is given the value 9

$C_6 \leftarrow 3.0 - (1 / 4) + 6.0$; C_6 is given the value 9.0

$C_6 \leftarrow (9 / 10) + 1.0 - (7 / 16)$; C_6 is given the value 1.0

$I_6 \leftarrow (9 / 10) + 1.0 - (7 / 16)$; I_6 is given the value 1

In summary then, if you "mix" arithmetic by mixing fixed and floating constants or variables, the rule is that RUNCIBLE always initializes its arithmetic to FIXED at EACH parenthesis level and continues that way until it encounters a floating variable or constant at the same level in which case the arithmetic stays floating at that parenthesis level. The search takes place from the RIGHT. The final substitution is done in the arithmetic of the left hand side variable. Floating point answers are always rounded to eight significant figures, but fixed point numbers are never rounded -- all figures to the right of the decimal point are dropped. (Thus, 8/9 = 0 in fixed point division!)[2]

---

2. The remainder of a fixed point division is usually dropped but it can be saved if desired as described in Appendix III.

Extensions: In addition to these basic operations, a wide variety of "extensions" can be added to the compiler, making it extremely versatile. Rules for their use are given in the writeup supplied with each individual extension which might be in the subroutine library at your installation, but a few general rules and examples will be illustrated here for clarity.

Reference to an extension is made with quotation marks followed by the name of the extension and a comma; these are followed by one or more inputs and closing quotation marks. (The quotation marks are treated just as parentheses by Runcible except that they identify extensions.) The name of each extension will end with either an "E" or "F," the former signifying floating point output and the latter signifying a fixed point result.[1] Three commonly used extensions are illustrated below, letting C1 stand for x:

Math. language:	IT-language:
$\sqrt{x}$	"RT2 E, C1"
$\sin x$	"SINE, C1"
e^x	"EXP E, C1"

As a final example for this section on operations, here is the IT-language representation of the "Wolontis function"

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

We have f(x) = "SINE, C1" / "RT2 E, 1. + "EXP E,(-C1)P 3." "

STATEMENTS. The numbers, variables, and operators we have now learned are put together into meaningful statements as instructions to Runcible. Each statement is given a number which is some integer less than 1000. The order in which statements are executed has no relation at all to the numbers on the statements -- they are eventually carried out in essentially the same order in which the original deck was compiled. It is sometimes helpful, however, to make the numbers consecutive in case the cards should get mixed up. Statement number zero is special -- this number is reserved for statements which are not going to be referred to in the program, and as many statements can be numbered zero as desired. Each nonzero statement

1. When a subroutine is identified by number it is floating point if the number is below 500.

number, however, must never appear on more than one statement in a program -- it must be unique.

Substitution statements. There are many types of statements; perhaps the most frequently used is a substitution statement. In this statement a variable is given the value of any mathematical expression; a new value is "substituted" for its former one. The substitution operation is denoted by a backwards arrow (←). Examples:

```
Y1 ← 0.
C10 ← Y1 / (Y2 x Y2)
I3 ← I3 + 1
YI4 ← I8 - 3
```

In the first case Y_1 is set to zero. The second example sets C_{10} equal to Y_1 divided by Y_2 squared. In the third illustration, I_3 becomes equal to one greater than its former value. The last case takes three less than the value of I_8 and gives it to the Y- variable which has subscript equal to the current value of I_4. Note that in the last example RUNCIBLE will convert the fixed point right-hand side automatically into a floating point number before inserting it into YI4. Variables always retain their values until being changed by a substitution statement or a READ statement (see below) or perhaps an extension statement (see below).

JUMP statements. RUNCIBLE normally executes statements in the order it receives them, but this sequence can be broken with a JUMP statement which tells the compiler to jump to a certain statement and continue from there. A JUMP statement is written simply

```
JUMP TO  k
```

where k is the number of the statement which should be executed next. A variable or a parenthesized arithmetic expression may also be used instead of k as long as it is fixed point; e.g., JUMP TO I2 or JUMP TO (I2 + 3 x I1). k must never be zero or negative.

READ statements. A READ statement is written

```
READ
```

This statement will cause the machine to read in one or more cards of data for the problem. All data for a program other than constants enter

the 650 via a READ statement. The form of data cards is described later in the section on formats.

Output statements. Answers are punched onto cards when an output statement is given. There are two different kinds of PUNCH statements:

1. PUNCH statement type one will put the current values of up to four variables onto one card. The statement

PUNCH Y3

will cause the current value of variable Y3 to be imprinted on a card;

PUNCH C4 PUNCH I6 PUNCH Y9

will put the values of C_4, I_6, and Y_9 all on the same card. Up to four variables can be punched at a time in this manner.

PUNCH Y(I4+1)

is also allowable -- it will type the value of the variable specified by (I4+1). It is not legal, however, to give a statement like PUNCH (Y1+Y2) or PUNCH AY7.

2. If a large number of consecutive variables are to be punched, a statement such as

PUNCH Y1 THRU Y15

may be given. When the word "THRU" is used like this, up to seven answers will be put onto each card. The statement above, for example, will put Y1 - Y7 on the first card, Y8 - Y14 on the second card, and Y15 on a third card.

Formats of the output cards from a PUNCH statement are described in a later section. They are identical to the formats required by the READ statement, so answers may be used as data to another program.

When an output statement has statement number zero it will be bypassed when the console switch of the 650 is set to plus during the running phase; it will be obeyed only when this switch is minus. This is a handy device for obtaining intermediate answers when checking a program out in its first few trial runs.

HALT statement. A HALT statement will stop the 650 if the programmed switch on the console is set to STOP.[1] It is written simply

HALT

1. Control will proceed to the next statement (if any) if the PROGRAM START switch is depressed after a HALT. A programmed HALT can be identified by its data address of 8003.

A number may be written after the word HALT like this:

HALT 12

in this case the machine will stop displaying the number 12. This technique may be used to differentiate between several HALTs in the same program.

BYPASS statement. The BYPASS statement, written (as might be guessed)

BYPASS

does absolutely nothing; it is simply bypassed during the running phase. (Believe it or not, this statement can be useful!)

Extension statements. Some extensions which may be used with the compiler have no specific output. The form taken on by such statements is specialized and varied; rules are given in the writeup for each individual subroutine.[1]

Conditions. Any of the above statements may be made conditional and executed only if a certain relation holds true. There are three allowable relations: =, >, ≥ (equals, greater than, and greater than or equal). The condition is preceded by the word IF, thus:

IF Y9 = C9

IF I4 ≥ 2

IF A(C8-Y8) > 1 B -8

In the last example the relation is satisfied if the magnitude of $C_8 - Y_8$ is greater than 10^{-8}. Here are some examples of statements made conditional:

1. JUMP TO 1 IF Y3 = 0.
2. READ IF 1 ≥ C10
3. HALT IF Y(I5+2) = Y(I5+1)
4. PUNCH C2 IF -(I5/5) ≥ 4 x I5
5. C4 ← Y4/10. IF C3 x C3 > Y4 P 8.4
6. JUMP TO I3 IF I1 = C1
7. Y5 ← 3.1415927 IF "SINE, Y5" = 0. IF Y5 > 1.571

The lefthand portion of the statement is executed only if the relation is satisfied; when the condition is not fulfilled the statement is treated like a BYPASS. Note that fixed and floating point arithmetic may be mixed as in example 6. The seventh case is interesting because the substitution

1. Examples of extension statements: STATISTICAL READ 1,2; READ PROGRAM; EDIT C1 THRU C2, 1231231234, 1112223334; CONVERT Y1 THRU Y50.

will be done only if both conditions hold.

Iteration statement. The last type of statement to be discussed in this section is a handy programmer's convenience for a sequence of operations which occurs quite often in typical problems. An "iteration statement" causes a number of other statements to be repeated over and over as a variable is changed by a specified amount.[1]

Iteration statements look something like this:

n, v1, v2, v3, v4,

v1 is the variable (I, Y, or C) which is to be changed; v2 is its starting value and v4 is its finishing value; and v3 is the amount by which the variable is to be changed before repeating the sequence of statements again.[2] The n in this statement is a certain statement number: All statements after the iteration statement up to and including statement n will be repeated for every value of the variable. (n must not be zero, nor should the statement immediately following the iteration statement have number zero.) Please note the comma after v4 -- it must be included!

For example, the statement

5, Y1, 1, 2, 13,

means: execute all statements from the next one to statement 5 for Y1 taking on the values 1, 3, 5, 7, 9, 11, and 13.

It is possible to include an iteration statement within the scope of another iteration statement -- all the details will be handled automatically by RUNCIBLE; this "nesting" of iteration statements may proceed to four deep.[3] Iteration statements cannot be made conditional.

v2, v3, and v4 need not be constants; they may be variables or even mathematical expressions. For example, the starting value might be something like Y4/2. and the finishing value, C8P I3. A lot of liberties are allowable here, the only restrictions being:

1. No more than five characters (excluding spaces) may appear between any two adjacent commas in an iteration statement. Thus the statement

2, Y1, (Y2 x Y3), -6., 3.1415927,

is illegal on two counts: v2 has seven characters -- this may be cured by dropping the parentheses; v4 has nine characters -- the remedy is to let some

1. This process can also be programmed, of course, without using an iteration statement.
2. If v4 - v2 is not exactly divisible by v3 the iteration procedure will be discontinued just before the value v4 is passed.
3. The scope of every iteration statement must be contained in the scope of any other iteration which uses it; that is, if a certain statement iterates on statements 1 through 5, say, no meaningful iteration statement within these bounds will terminate at any statement after five.

Example Problem:

Flowchart of houses to det. cost

	sq ft area	no. of stories	const	no. of rooms
1	C_1	C_2	C_3	C_4
2	C_5	C_6	C_7	C_8
3	C_9	C_{10}	C_{11}	C_{12}
4	C_{13}	C_{14}	C_{15}	C_{16}

	area1	area2	area3	area4
sq ft.	4,	46	49	413
stories	42	44	410	414
const	43	47	411	415
room	44	48	412	416

Answers:

	area1	area2	area3	area4
hs 1	C_{17}	C_{18}	C_{19}	C_{20}
hs 2	C_{21}	C_{22}	C_{23}	C_{24}
hs 3	C_{25}	C_{26}	C_{27}	C_{28}
hs 4	C_{29}	C_{30}	C_{31}	C_{32}

(can be done 5 statements)

use approx 30

1st subscript = row
2nd " = col.

$$A \cdot B = C \qquad C_{ij} = \sum_{k=1}^{4} a_{ik} b_{kj}$$

i.e., $C_{34} = a_{31} b_{14} + a_{32} b_{24} + a_{33} b_{34} + a_{34} b_{44}$

(3 nested iterations
3 counters)

$y_1, y_2, y_3, y_4, y_5 = y_6$

N1
N2

variable, say Y4, be set equal to 3.1415927 first and then use Y4 in the iteration statement.

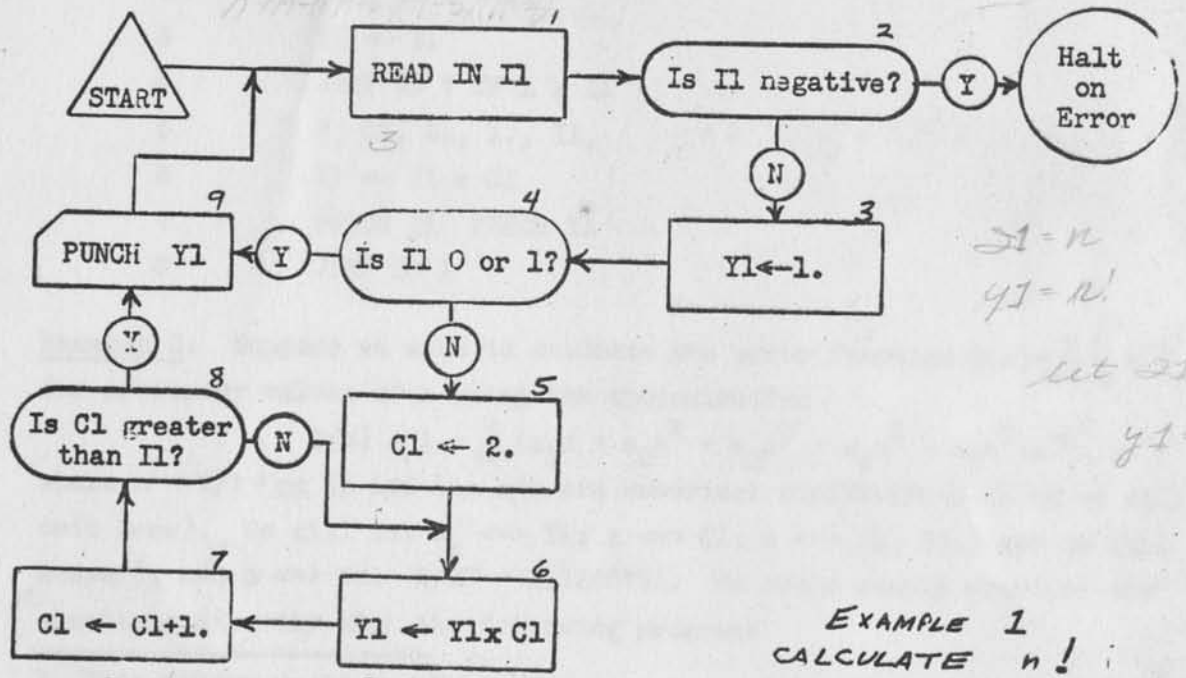
2. If v3 is negative it must start with the character -; if it is positive it must not start with a minus.[1] The example above illustrates proper use of this rule; if v3 had been written (-6.) erroneous operation would have resulted.

3. n must be a positive fixed point constant and v1 must be a variable.

EXAMPLE PROBLEMS. Two sample problems will be given here to demonstrate some portions of IT-language as it has been described; more examples may be found in Appendix VI.

Example 1. Calculate n! where n is a non-negative integer.

Solution -- since is to be an integer we shall let I1 represent n in this case for input data to the problem; n! (the answer) will be floating point and we will call it Y1. We will do successive multiplications by a variable C1 which will run through the integers up to n. A flow chart to solve the problem would look something like this:



EXAMPLE 1
CALCULATE n!

1. If v3 is zero the programmer should visit a psychiatrist.

row index i
col. index j

C_{17} C_{18} C_{19} C_{20}
C_{21} C_{22} C_{23} C_{24}
C_{25} C_{26} C_{27} C_{28}
C_{29} C_{30} C_{31} C_{32}

subtract 16 from each sub-script

$$[(i-1)\times 4] + j$$

$$C\{16 + [(i-1)\times 4] + j\}$$ given subscript

$$C'(16 + j + 4 \times i - 1)$$

- C' → subscript of element 1-1
- j → col. index
- 4 → width of matrix
- i → row index

Translation of the flow chart into a series of statements is almost automatic:

Number:	Statement:	Remarks:
1	READ	(Read in I1)
2	HALT IF 0 > I1	(Error stop if n negative)
3	Y1 ← 1.	(Initialize Y1)
4	JUMP TO 9 IF 1 ≥ I1	(0! = 1! = 1.)
5	C1 ← 2.	(Initialize C1)
6	Y1 ← Y1 x C1	(Multiply successively
7	C1 ← C1 + 1.	until C1 = n)
8	JUMP TO 6 IF I1 ≥ C1	
9	PUNCH I1 PUNCH Y1	(Punch answer)
10	JUMP TO 1	(start over again)

Observe that the program parallels almost exactly the instructions you would give to a person[1] telling him what you wish to be done.

The following is the same program using an iteration statement:

1	READ
2	HALT IF 0 > I1
3	Y1 ← 1.
4	JUMP TO 7 IF 1 ≥ I1
5	6, C1, 2., 1., I1,
6	Y1 ← Y1 x C1
7	PUNCH I1 PUNCH Y1
8	JUMP TO 1

Example 2. Suppose we want to evaluate the error function $Q(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$ for arbitrary values of x using the approximation

$$Q(x) = 1 - \frac{2}{\sqrt{\pi}} (a_1 n + a_2 n^2 + a_3 n^3 + a_4 n^4 + a_5 n^5) e^{-x^2}$$

where $n = 1/1 + px$ (p and the a's are numerical coefficients which we will omit here). We will let a_k <=> Yk; x <=> C1; n <=> C2; Q(x) <=> C3 (the answer); and p <=> Y6. $2/\sqrt{\pi} = 1.1283791$. We could simply evaluate the equations directly with the following program:

---

1. This person would be of below average mentality, but would be expert at arithmetic -- just like a computer.

Number:	Statement:	Remarks:
1	READ	(read in a_k, p, and x)
2	C2 ← 1./ [1.+ (Y6 x C1)]	(calculate n)
3	C3 ← 1. -(1.1283791 x {(Y1xC2)+[Y2x(C2P2)]+[Y3x(C2P3)]+ [Y4x(C2P4)]+[Y5x(C2P5)]} x [2.7182818P(-C1xC1)]	(calculate Q(x))[1]
4	PUNCH C1 PUNCH C3	(punch answer)
5	JUMP TO 1	(read in another x and continue)

But the evaluation of the polynomial will be quite a bit more rapid if we rewrite the expression $Q(x) = 1 - \frac{2}{\sqrt{\pi}}(n(a_1+n(a_2+n(a_3+n(a_4+na_5)))))e^{-x^2}$. Now we could evaluate this new equation directly or set up a "loop" type of routine which calculates the polynomial from the inside out. Careful study of the program below will be very instructive.

1	READ	(read in a_k, p, and x)
2	C2 ← 1./[1.+(Y6xC1)]	(calculate n)
3	C4 ← Y5	(initialize C4)
4	5, I1, 4, -1, 1,	{iteration to calculate C4 =
5	C4 ← YI1 + (C2 x C4)	$a_1+n(a_2+n(a_3+n(a_4+na_5)))$ }
6	C3 ← 1- [1.128379lxC2xC4x"EXP E,-(C1xC1)"]	
7	PUNCH C1 PUNCH C3	(punch answer)
8	JUMP TO 1	(read in another x and continue)

Of course an even shorter program would be

1	READ	
2	C3 ← "ERF E,C1"	(error function extension)
3	PUNCH C1 PUNCH C3	
4	JUMP TO 1	

...but this is cheating.

BASIC PACKAGES. Standard subroutines such as floating-point arithmetic and input-output operations have been incorporated into "basic packages" which augment the finished program in its running stage. There are many of these packages, each of which has its own special purpose or goal, and the requirements of each individual program will determine just which one to use. For

1. In this statement and the last not all of the parentheses are necessary. The P operator is rather slow when doing floating point calculations since it always requires finding a logarithm and antilogarithm -- C2xC2xC2 would actually be much faster than C2P3 as written--but the following program speeds it up even more. The extension EXP E might have been used here as it is in the second version of the program.

instance, there are the "A" packages, designed for eight-digit accuracy and legible error display; and the "S" packages, stripped down for speed. The separate writeup for these packages goes into greater detail.

WHICH PACKAGE TO USE ?

If you are going to run your program on an "ordinary" 650, then you must use one of the "A" packages; P1A, P2A, or P3A. If your program is to be run on an augmented 650 (floating point, index registers, etc.) then you use one of the "Y" packages; P1Y, P2Y, or P3Y. All of the packages mentioned so far include the necessary routines for READ, PUNCH, and other various and sundry necessities of life for RUNCIBLE programs. The P1 packages contain a bare minimum of things and are used in most cases. The extra things included in the other two flavors are listed below:

P2 :	P3 :
P operator	P operator
Logarithm (base e) LNE	Log (base e) LNE
Exponential (base e) EXPE	Expon (base e) EXPE
	Sine (radian) SINE
	Cosine (radian) COSE
	Arctangent (rad) ARTE
	Square root RT2E

The decision as to which one of the packages to use is now simply made by examining the above lists. If your program does not use any of the things in either list--then use P1. If your program uses only things listed above the dashed line--then use P2, if your program uses features below the line, then use P3.

CARD PREPARATION AND FORMATS.

1. Data card format.

A. Four entries per card. On the four-per-card input, data is entered on each card in "couples," each consisting of an "IDENT" -- the name of a variable (17, Y1, etc.) -- and the VALUE of the variable itself. The "IDENT" looks like this:

$$\begin{Bmatrix} 01 \\ 02 \\ 03 \end{Bmatrix} \quad (0000 + s) \quad (0000 + k)$$

where 01, 02, 03 are used for I, Y, or C respectively,[1] and the number s is the subscript value. The number k is an arbitrary personal identification which may be attached by the programmer; it should be filled with zeroes if not used for identification.

The VALUE of the variable differs for fixed and floating point entries. For fixed point (I-) variables, write down the integer and add left hand zeroes until it is ten digits long; this is the correct form for VALUE. For floating point (Y- and C-) variables, first write the number as

$$a.bbbbbbb \times 10^{c}$$

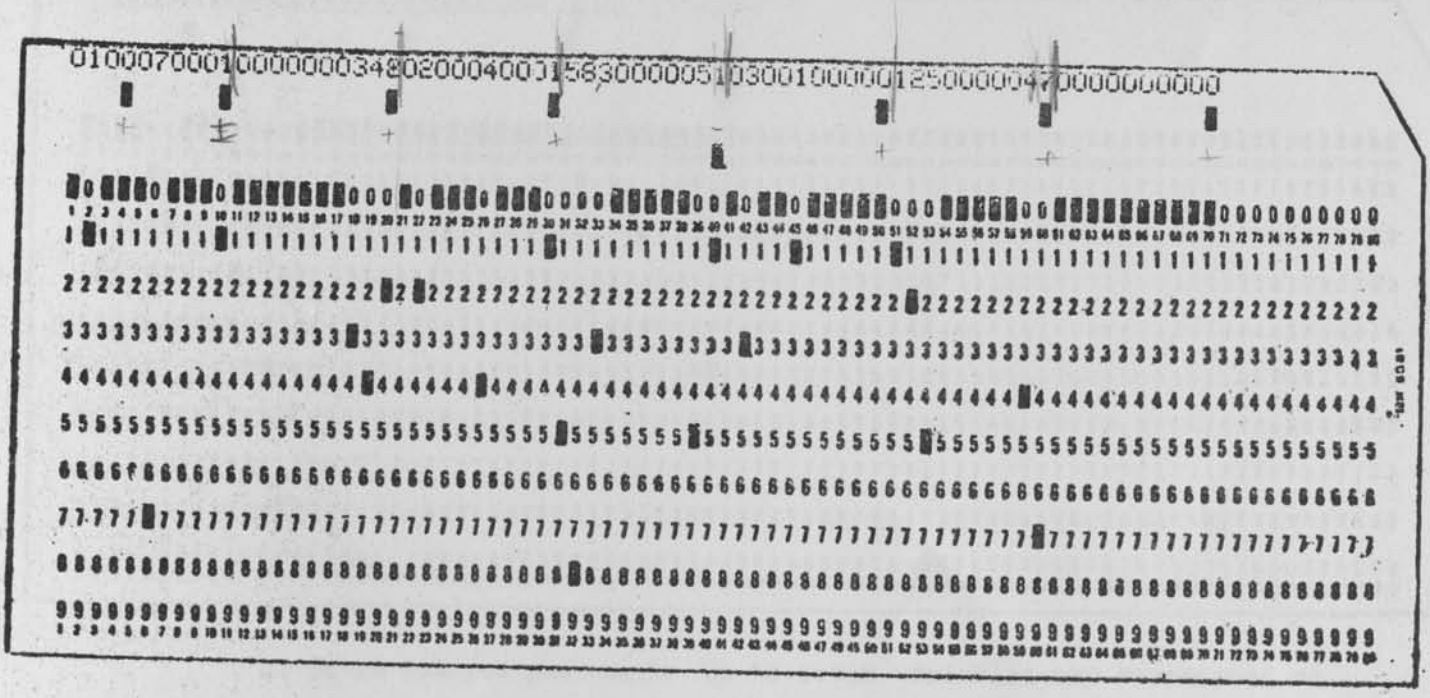
where c is some exponent between -50 and 49, and the decimal point is as shown (with digit "a" non-zero). Then the correct form[2] for the floating point VALUE is

$$(a)\ (bbbbbbb)\ (50 + c)$$

Thus, five would be written 0000000005 in fixed point and 5000000050 in floating point; fifty would be 0000000050 fixed and 5000000051 floating. Zero is always written as ten zeroes, whether fixed or floating.

---

1. In more advanced RUNCIBLE programs, 04, 05, and 06 stand for S, T, and P regions, respectively.
2. These rules must be slightly modified when using the 653 floating point hardware: add fifty-one (51) to the exponent instead of 50; e.g., five is 5000000051 and fifty is 5000000052 for floating point VALUES.



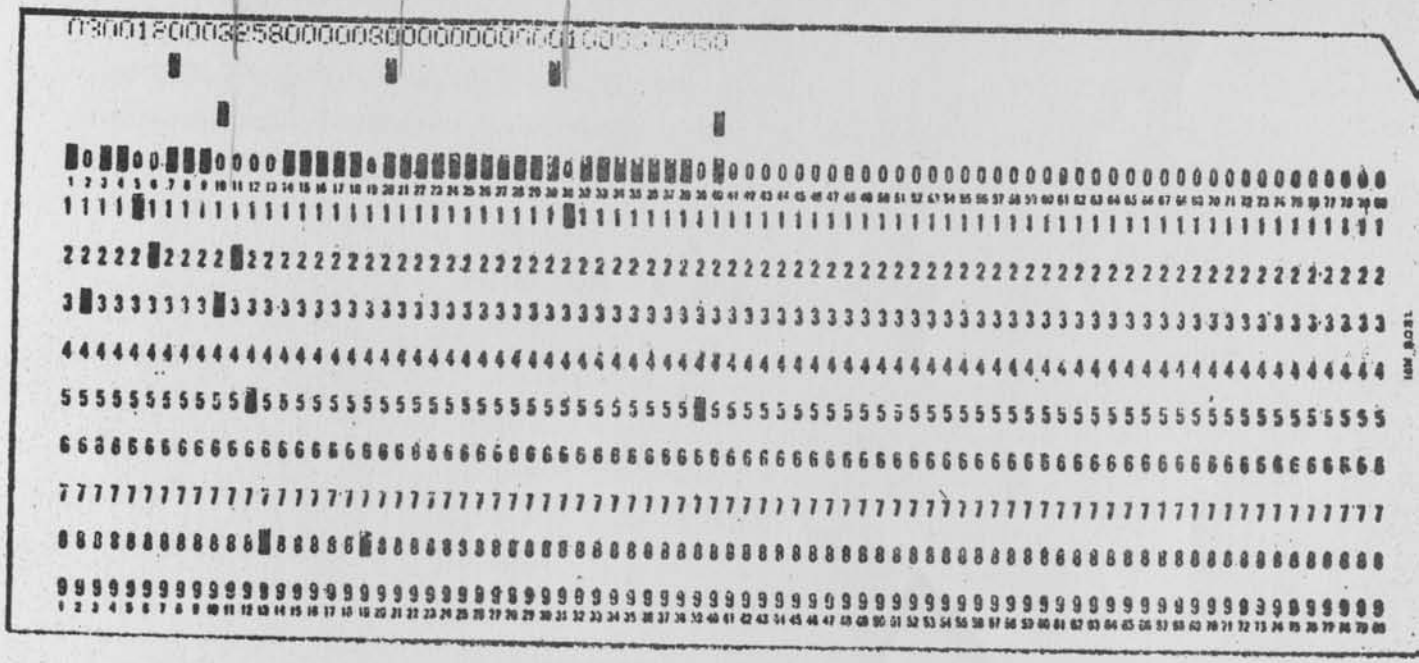
The sign of the number must always be punched over the right-most (units) digit of the VALUE: a 12- (Y) overpunch indicates a plus sign and an 11- (X) punch indicates a minus sign. Signs must also be placed over the units digit of the IDENTs -- either plus or minus is permissible.

Columns 1-20 contain IDENT 1 and VALUE 1
Columns 21-40 contain IDENT 2 and VALUE 2
Columns 41-60 contain IDENT 3 and VALUE 3
Columns 61-80 contain IDENT 4 and VALUE 4

If less than four variables are to be entered on a card, the IDENT following the last VALUE must be filled with all zeroes and the remainder of the card may be blank.

A READ statement in RUNCIBLE will cause cards to be read until a card with IDENT 1 negative is found. Any number of cards having IDENT 1 positive may precede the card having negative IDENT 1, which stops the READ statement and signals for continuation of the program.

A 12- (Y) overpunch in column 4 is necessary to identify the four-per-card format. The sample card shown assigns I7 = 342, Y4 = -58.3, and C10 = .00125; observe the IDENTs and VALUEs carefully.



B. Seven entries per card. Up to seven variables may be entered on one card if the subscripts are all consecutive. The IDENT for the first variable is put in columns 1-10 in the same form as noted above except that k now must be a number between 1 and 7 which specifies how many consecutive values are to be loaded by this card. Then VALUE 1 (corresponding to subscript s of the IDENT) is placed in columns 11-20, VALUE 2 (corresponding to subscript s + 1) is placed in columns 21-30, VALUE 3 (for s + 2) into cols. 31-40, VALUE 4 (for s + 3) into cols. 41-50 and so on, leaving unused columns blank if desired.

A 12- (Y) overpunch in column 7 is necessary to identify the seven-per-card format. The sign of IDENT 1 again controls the reading of cards (see four-per-card format); when IDENT 1 is negative as it is above it signifies the last card of a series.

Seven-per-card and four-per-card formats may be used interchangeably. The sample card shown assigns C12 = 2.58×10^{30}, C13 = zero, and C14 = minus one.

2. Output card format. Output card format is precisely the same as input card format and can therefore be used as input to another program. With four-per-card output, IDENT 1 is made negative on all cards, and the identification k is the statement number of the PUNCH statement being executed. IDENT 1 is negative only on the last card of the seven-per-card output (when the word THRU was used in the compiler statement).

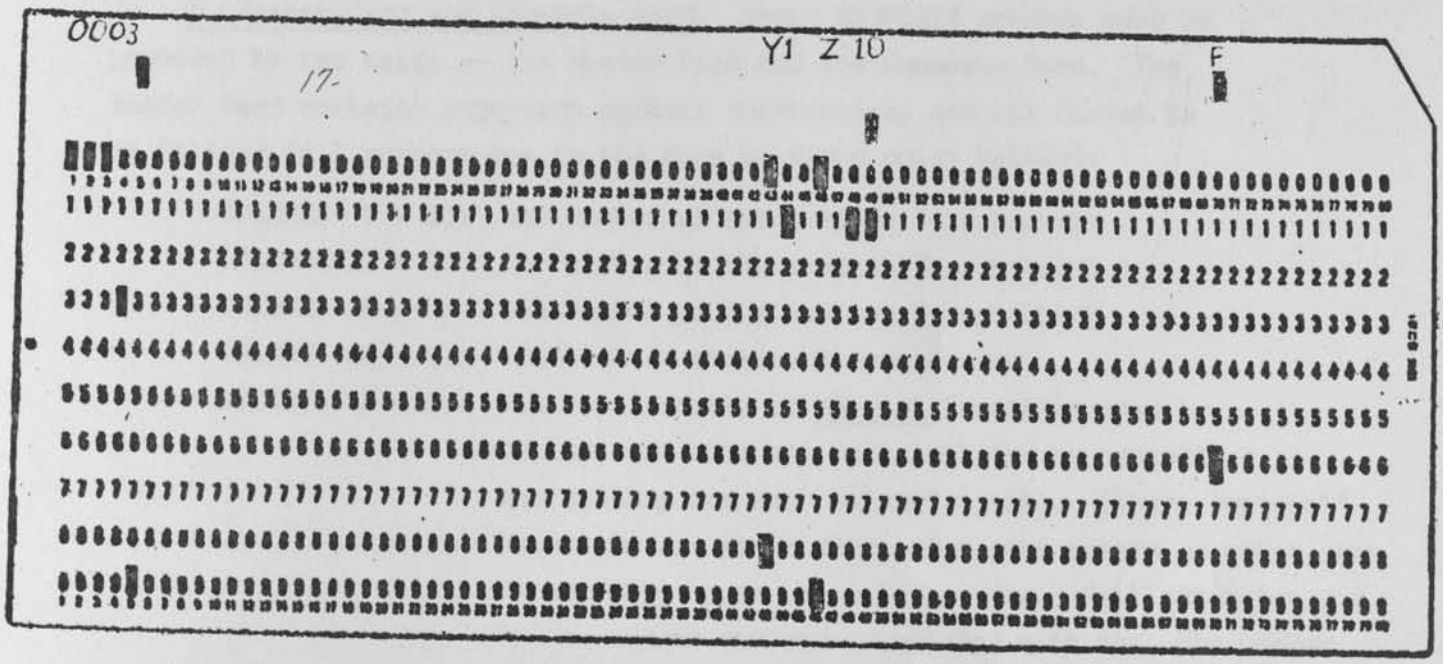
3. RUNCIBLE Statements. Since the basic 650 for which this program is designed recognizes only numbers and alphabetic characters--not equal signs or quotation marks--an alphabetic code is used for these symbols when punching the statements onto cards. Here is a complete list of symbols used and their alphabetic equivalents[1]:

(|	left parenthesis, bracket, or brace...	L
) |	right parenthesis, bracket, or brace..	R
.	decimal point	J
←	substitution	Z
=	equals	U
>	greater than	V
≥	greater than or equal	W
,	comma	K
"	quotation marks (right or left).....	Q
+	plus	S
-	minus	M
x	times	X
/	divided by	D

Each statement must be punctuated by an F ("finish"), which is added at the end. The very last statement of a program must end with an FF ("Final finish").

The statement number, n, is punched as (0000 + n) in columns 1-4. Column 5 must contain both a 12(Y) punch and a 9 punch (this is the code for the letter "I") to identify the format, and a 9-punch may be added in column 5 to be picked up by tabulator board wiring for use in listing the deck. The statement itself is punched in columns 43 through 69, and column 70 must contain the F which terminates it. (If a statement is so long it does not fit on one card, up to five cards may be used. In this case the "F" should appear only on the very last card of the statement, and columns 43 through 72 may be used for characters of the statement on all other cards. A statement may thus contain up to 148 characters, including the final F or FF as the case may be. Columns 1-4 need only be punched on the first card of each statement.) Columns 6-42 and 73-80 are ignored by RUNCIBLE except columns 7 and 41 must not contain 12- (Y) punches[2] -- but these

1. It may prove convenient at your installation to put these symbols directly onto the keys of your keypunch keyboard; e.g., ←Z
2. Column 10 should be blank if the 533 plugboard has the added statistical features.



columns are usually not used.

• Blank columns in the middle of a statement are always ignored, and they may be inserted or deleted at will. A substitution statement must start in column 43. Here is example program one as it would be punched onto cards; statement number three is shown above.

cols 17-20 arbitrary numerics or blanks

STATEMENT NUMBER	STATEMENT
1 2 3 4 5	43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
0 0 0 1	R E A D F
0 0 0 2	H A L T I F Ø V I 1 F
0 0 0 3	Y 1 Z 1 J F
0 0 0 4	J U M P T O 7 I F 1 W I 1 F
0 0 0 5	6 K C 1 K 2 J K 1 J K I 1 K F
0 0 0 6	Y 1 Z Y 1 X C 1 F
0 0 0 7	P U N C H I 1 P U N C H Y 1 F
0 0 0 8	J U M P T O 1 F F

Here Ø represents zero and O an alphabetic letter O. Appendix VII contains another sample program punched on cards including the Header and Comments Cards.

Rules.
1. The 1st character of a statement must be in col. 43.
2. Every statement must end with an "F"
3. The "F" when it appears, must be in col. 70
4. No statement may spend 5 cards.

4. Header Card and Comments Card. Every RUNCIBLE program must be preceded by two cards -- the Header Card and the Comments Card. The Header Card contains important control information, and its format is as follows (all numbers are in the form of fixed point VALUEs):

COLUMNS 1 - 10: the number of the maximum I- subscript

COLUMNS 11 - 20: the number of the maximum Y- subscript

COLUMNS 21 - 30: the number of the maximum C- subscript

COLUMNS 31 - 40: the highest statement number used

COLUMNS 41 - 50: the total number of special locations used by extensions--this may be calculated by consulting the writeup for each extension used. (These "special" locations are those which, for some reason or other, are supposed to be in sequential order.) Each extension now has two magic numbers associated with it: The number of special locations--this goes into columns 41-50--and the number of other locations--this goes into columns 61-70.

COLUMNS 51 - 60: the number of locations used by the basic package.[1] (This is 325 for P1A and 525 for P2A, 751 for P3A the most frequently used packages; for other packages, see their own write-up.

COLUMNS 61 - 70: the total number of locations used by extensions (excluding the basic package) -- this may be calculated by consulting the writeup for each extension used.

COLUMNS 71 - 80: all zeroes (unless using procedure of Appendix IV)

COLUMNS 10, 20, 30, 40, 41, 50, 60, 70, 80: 12- (Y) overpunches. The punch in column 41 is necessary to identify the header card.

Any of these numbers may be made a little larger than the actual value (for safety) but it is extremely important that none of them are smaller than the true values for this is an unchecked error which can lead to mysterious and unfortunate results.

---

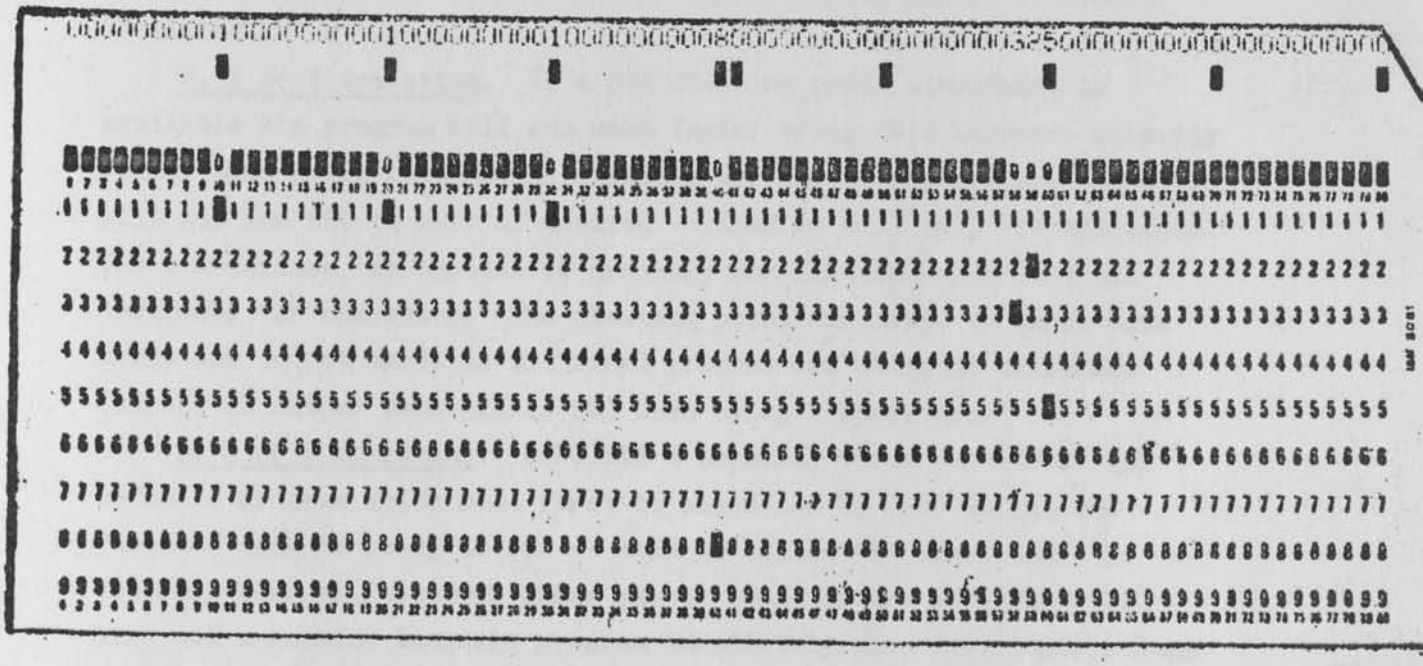
1. When using B-processing (see Operation Modes) this number must be at least 73, or at least 175 if any extensions are used, because of the Relocation Package used to load the program.

DO NOT USE

P1y 185.
P2y 360
P3y 475

The example Header Card which appears on the next page is the one determined by example program one, using basic package PLA.

The Comments Card is generally easier to prepare than the Header Card: columns 1 - 42 and 73 - 80 are blank and the remaining columns 43 - 72 may be filled with anything the programmer's heart desires (as long as it is acceptable to the alphabetic attachment) -- the title of



the program, his own name, or his girlfriend's name; or he may choose to be obstinate about the whole thing and leave the card completely blank.[1]

MODES OF OPERATION. There are twenty-four different modes of operation to choose from while compiling, and these are selected by setting switches on the 650 console.

1. Clocking. RUNCIBLE I's clocking feature is a major step forward for the testing of a compiler program during its trial runs. "Minimum" clocking keeps tabs on the number of the statement which is being executed[2] and will display this number if the computer stops during the running phase (see Stops). "Minimum" clocking does not require any more space in the 650's memory than the program would normally use; it does, however, add approximately .026 seconds machine running time to every non-zero statement executed.

Additional available clocking features (stopping at certain statement numbers, flow tracing, and full tracing) are explained in detail in Appendix I. There is a three-way choice while compiling: to use no clocking, minimum clocking, or the full clocking, which allows any or all of the aforementioned

1. The Comments Card becomes a SOAP III comments card which is the first card of the symbolic output.
2. If it is non-zero.

extra features. The extra features require an additional 98 memory locations.

2. X or Y operation. If a 653 floating point attachment is available the program will run much faster using this hardware directly rather than going through the floating arithmetic extensions. RUNCIBLE will put the 653 to work if desired -- this we will call "Y" operation for convenience, as opposed to ordinary compilation which we shall designate "X" operation. The floating point exponents on input data cards and output cards in the final program are slightly different (excess-51 rather than excess-50) when using Y-operation.[1]

3. A or B operation. RUNCIBLE I normally turns out a SOAP III symbolic program which must first be assembled by SOAP before the finished deck is ready to run -- this is called "A" operation. If desired, however, the SOAP phase may be bypassed and RUNCIBLE will turn out a machine language program immediately with five instructions on each card ("B" operation).

B-operation is to be recommended for smaller scale problems and when conservation of cards is more important than conservation of time. A-operation is more versatile and is recommended for programs which will be long runs. A B-program requires less than one-sixth the cards of an A-type (multipass) program during the IT-language to machine language translation, and takes approximately one-half the time to assemble; however, it uses about 10 percent more time while running. (More accurate timing has yet to be made for this comparison.)

Extensions: Programming is exactly the same for A or B processing except that there are different maximum limits for extensions. In type A, no more than nine named extensions may be given in any one statement; type B restricts the programmer to a total of ten different extensions (named or not in P1 or P2) in his entire program.

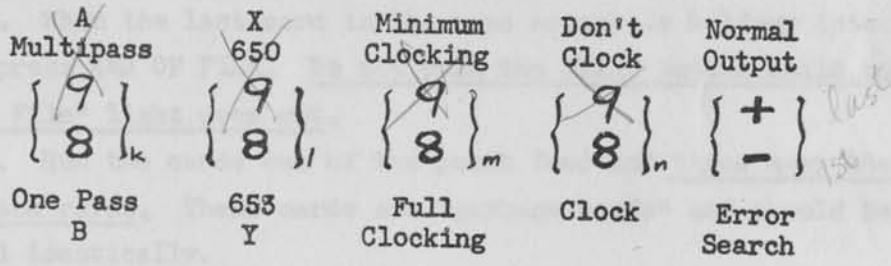
4. Error searching. There are many checks on programming errors built into the RUNCIBLE program (see STOPS); these are sometimes awkward to correct without beginning compilation over again. The error search mode feature in RUNCIBLE I will check an entire program

1. Index register A (8005) is used to calculate variable subscripts when Y-operation is used, but all index registers are free for use by extensions.

for goofs without punching any of its normal output cards. In this case any error which would normally have halted compilation will not usually stop the computer -- it will be punched onto a card instead.

Selecting the modes. These various modes of operation are selected in the following manner: we will "build" a four-digit number klmn+. Digit n is 9 if not clocking[1] and 8 if clocking; and if clocking is used, digit m is 9 for minimum clocking only and 8 for the expanded clocking features described in Appendix I. For X-operation, digit l is set to 9, while it is 8 when Y-operation is desired. Digit k is 9 when multipass (A) operation is to be used and 8 when using the B (single pass) feature. The number klmn finally is plus when normal output is called for and minus when searching out errors.

All of these rules can be remembered more easily by the general pattern that all four digits are 9 for basic operation, and changing any one of them to 8 calls out a more special feature. Four eights is more or less asking RUNCIBLE for the "works." The rules are summarized in this chart:



OPERATOR INSTRUCTIONS.

Step 1. Insert RUNCIBLE I plugboard into 533 unit. This board will be used during the entire operation.

Step 2. Clear any cards out of the read feed and punch feed and ready the punch feed with blank cards.

Step 3. Place the RUNCIBLE I deck in the read hopper, face down, and place your program deck on top of it. The program deck consists of 1) Header Card 2) Comments Card and 3) Statements (in that order).

---

1. When not clocking, digit m should also be a 9.

Step 4. Set the 650 console to the following:

SWITCH	SETTING
STORAGE ENTRY	70 1951 klmn ± (see Operation Modes)
PROGRAMMED	STOP
HALF CYCLE	RUN
ADDRESS SELECTION	anything -- 1888 recommended
CONTROL	RUN
DISPLAY	LOWER ACCUM
OVERFLOW	SENSE
ERROR	STOP

Step 5. Depress the following buttons in order:

1) COMPUTER RESET
2) PROGRAM START

Step 6. Press both START buttons on the 533 unit.

Step 7. When the last card in the read hopper is halfway into the machine, depress END OF FILE. Do not push the START button again until the "End of File" light goes out.

Step 8. Run the cards out of the punch feed and throw away the top and bottom cards. These cards are "garbage cards" and should have been punched identically.

Step 9. Clear out the read feed and put the RUNCIBLE I deck back into the file.

Step 10. If using B-operation (see Operation Modes) skip down to step 14. Otherwise repeat step 5 and then run the SOAP III deck into the machine (by pressing START and END OF FILE at the proper times).

Step 11. If your program uses extensions which are not included in the basic package you are using, lift the first few cards from the output of step 8 which have 12- (Y) punches in columns 76 - 80 and insert the extensions in symbolic form into the output deck at this place. Now, place in the read feed face down (in order):

1) The Reservation Package which accompanies your basic package; e.g., R1A corresponds to P1A.
2) The output of step 8 -- with extensions inserted if they are used.
3) A BLANK CARD.

Step 12. Repeat steps 6, 7, and 8; clear out the read feed and put the SOAP deck, Reservation Package, and extensions back into the file. The output you now have is a running program assembled at five instructions per card.

Step 13. Repeat step 5 and run in your basic package followed by the output of step 12. You are doing fine. Skip down to step 16!

Step 14. Place in the read feed, face down, in order

1) The Relocation Package (standard for type B operation)
2) The output from step 8
3) Any extensions used in relocatable form
4) The basic package you want
5) A BLANK CARD

Step 15. Repeat steps 5, 6, and 7.

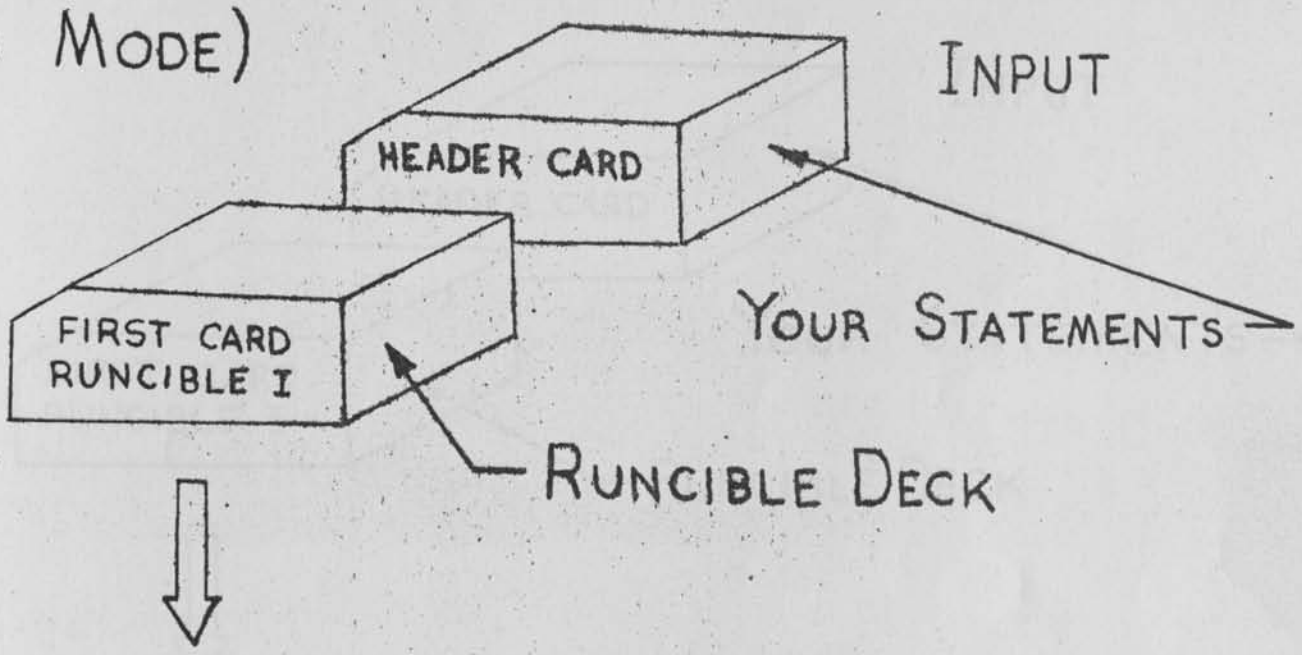
Step 16. When your deck is all in, the machine should be stopped with 1999 in the ADDRESS lights. Turn the CONTROL switch to ADDRESS STOP. Set the ADDRESS SELECTION switches to 1888. If you are using any of the expanded clocking special features, change the storage entry switches to the setting described in Appendix I; if you are running on ERROR SENSE (Appendix III) make the desired console setting; otherwise leave everything on the console alone. Now, push PROGRAM START (not COMPUTER RESET this time!).

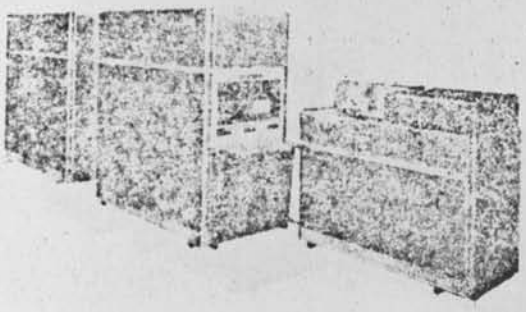
Step 17. If your program contains any READ statements, place the data cards in the read feed in their proper order. Hit the START button.

Step 18. Congratulations. If you have gotten this far, any output may consist of the answers to your problem. Be sure to return all library decks to the files and to remove your other cards from the machine.

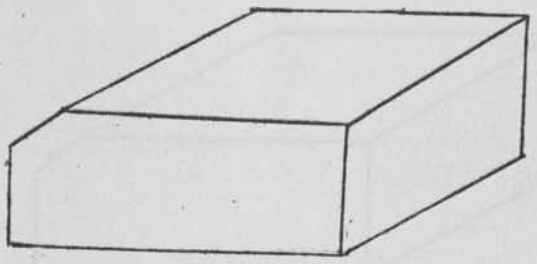
General Information Flow For Runcible I

Compiling Phase
(A Mode)



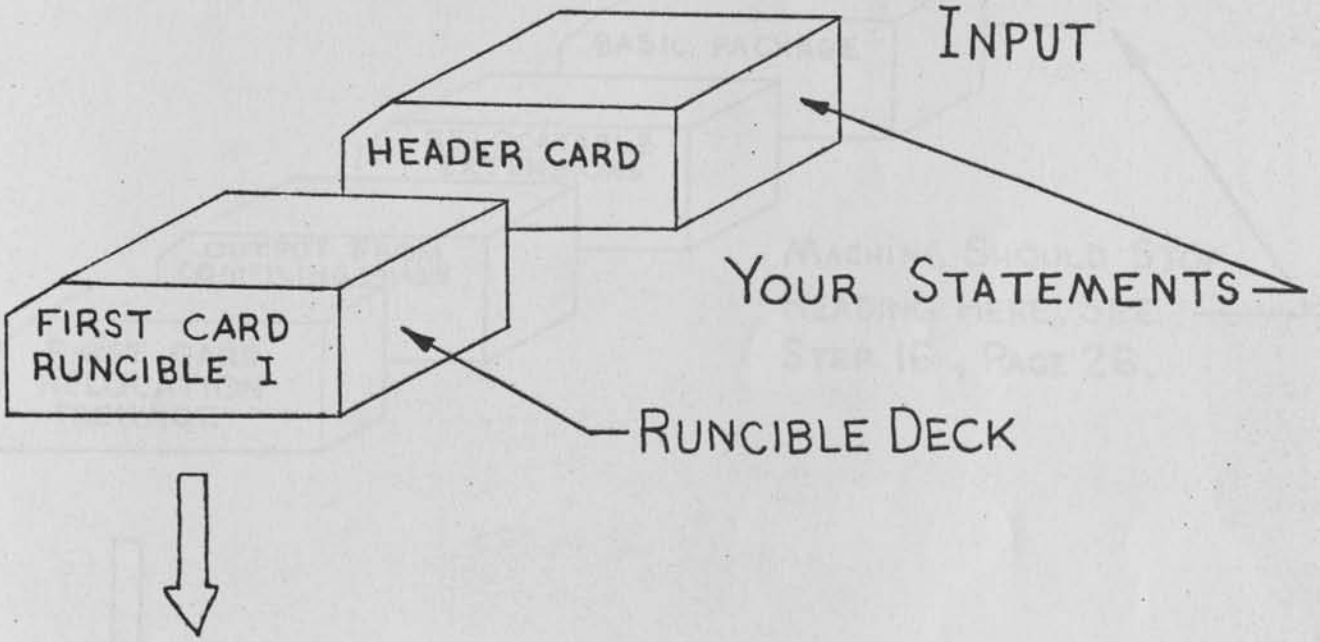


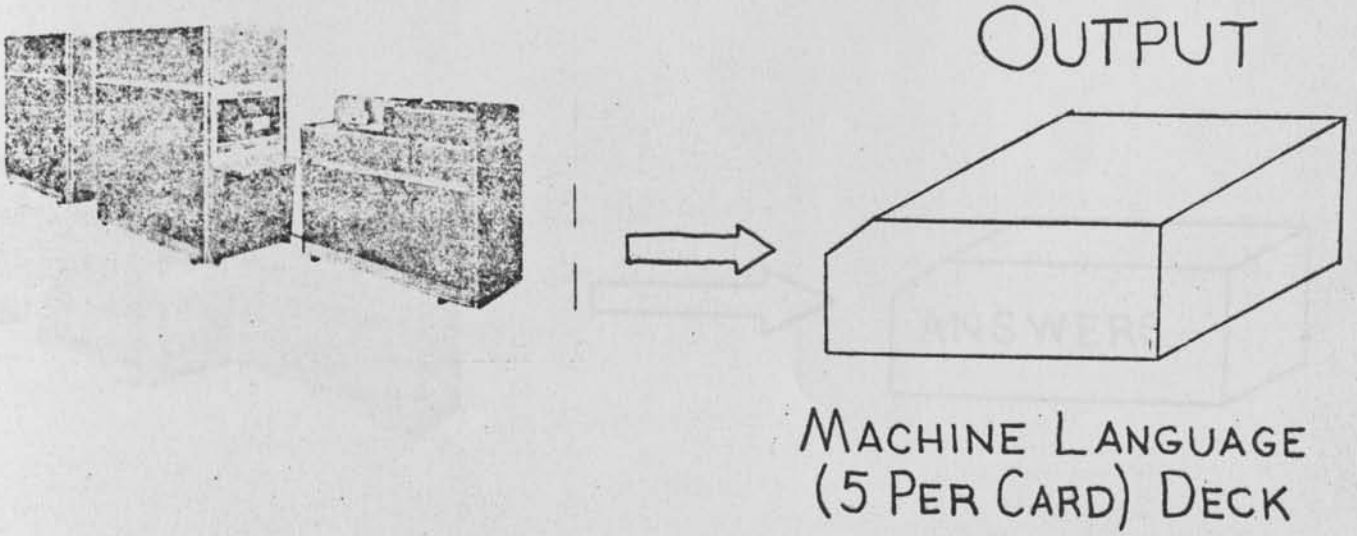
OUTPUT



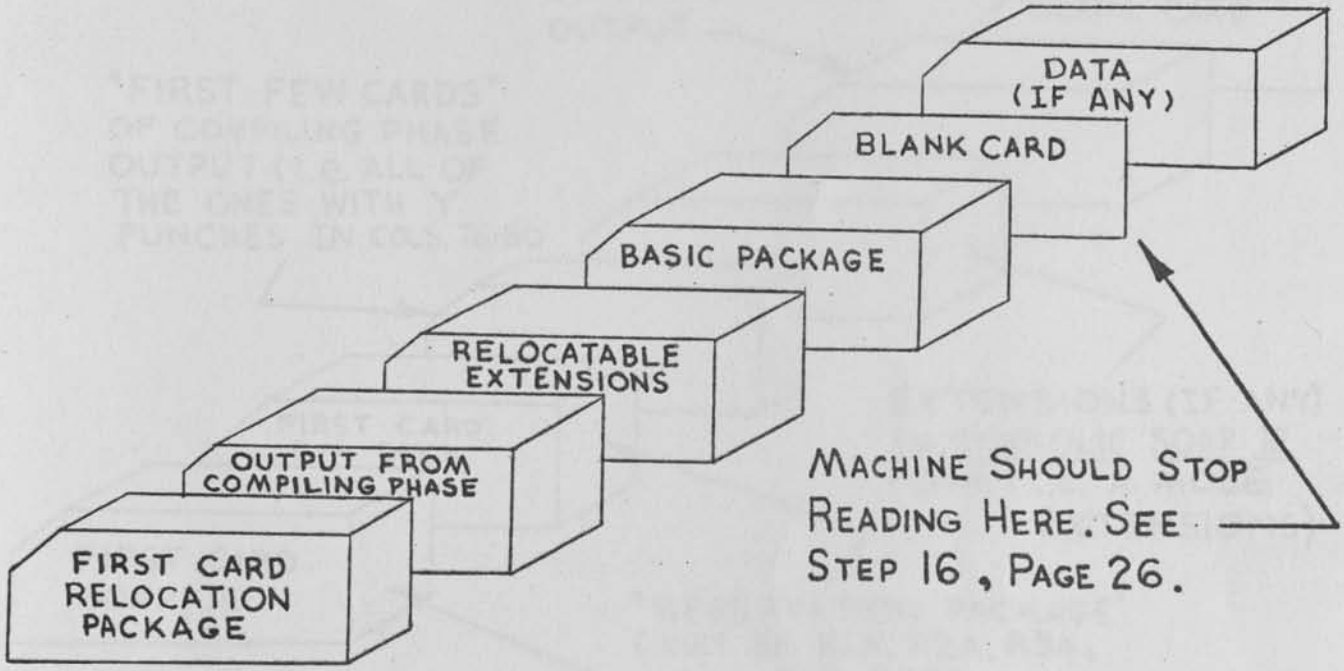
SOAP III
(SYMBOLIC) DECK

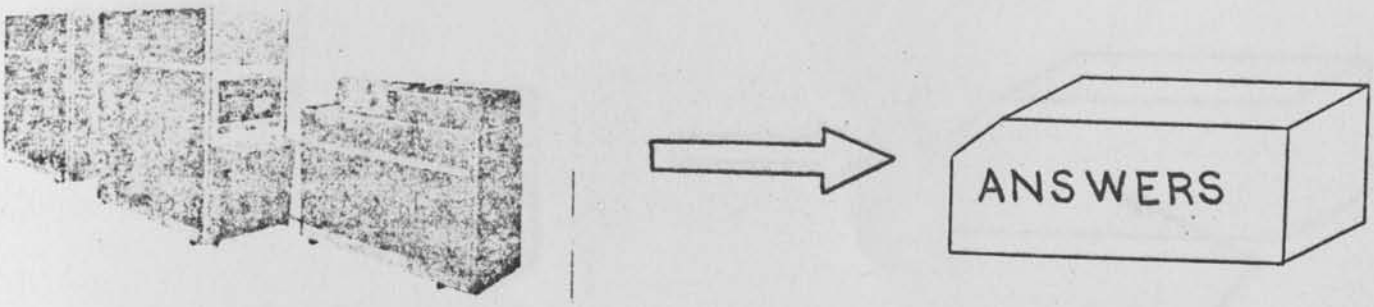
COMPILING PHASE
(B MODE)



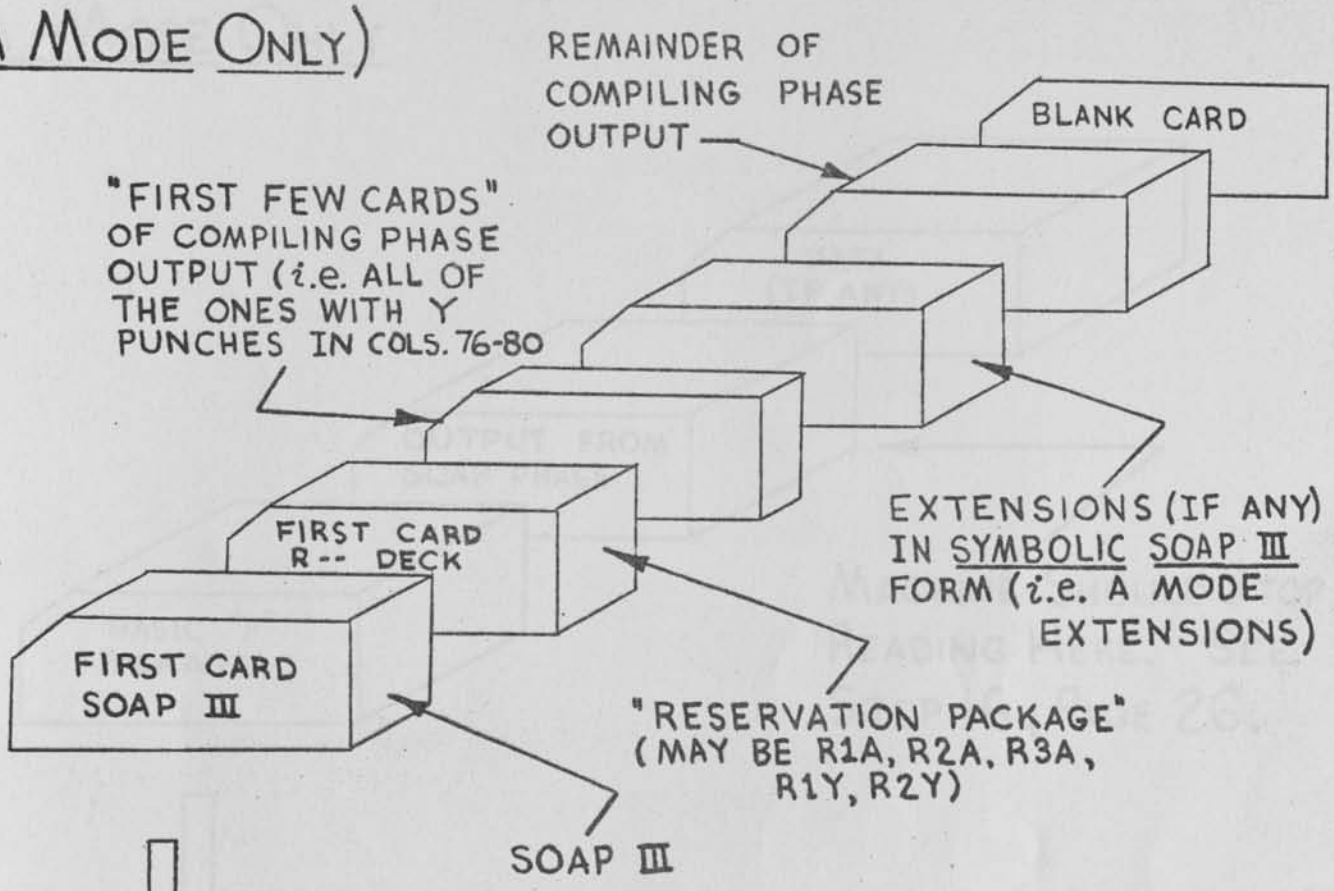


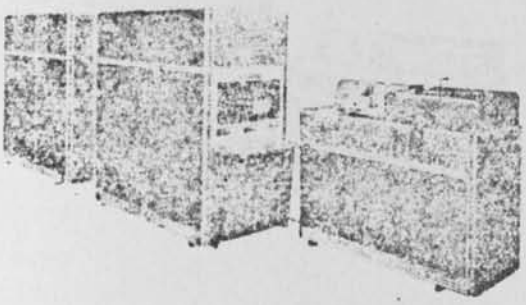
RUNNING PHASE
B MODE ONLY

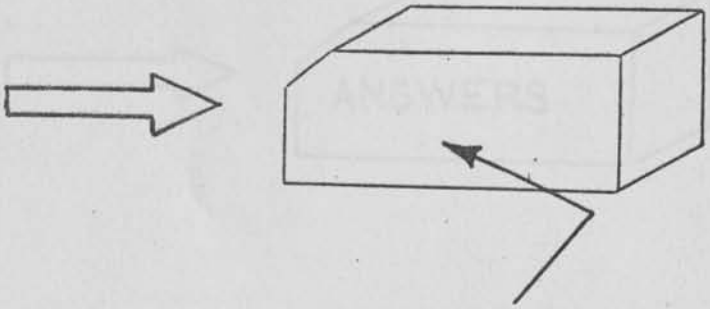




SOAP PHASE
(A MODE ONLY)

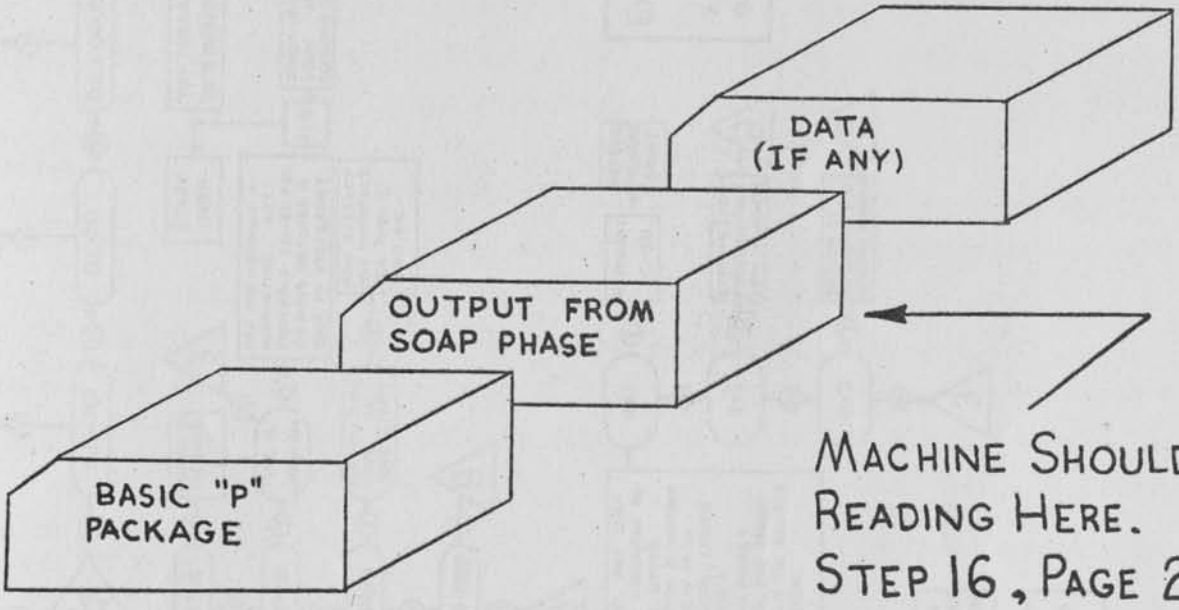




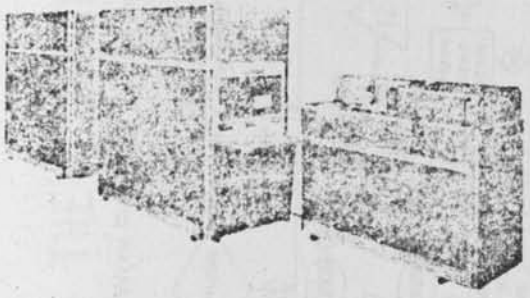


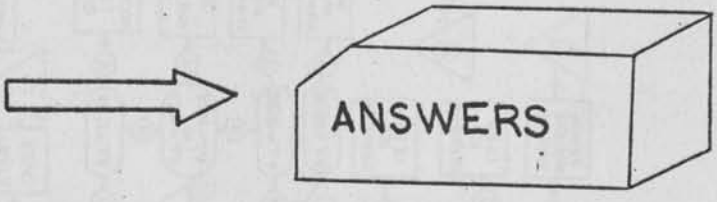
MACHINE LANGUAGE PROGRAM
5 INSTRUCTIONS/CARD

RUNNING PHASE
A MODE ONLY



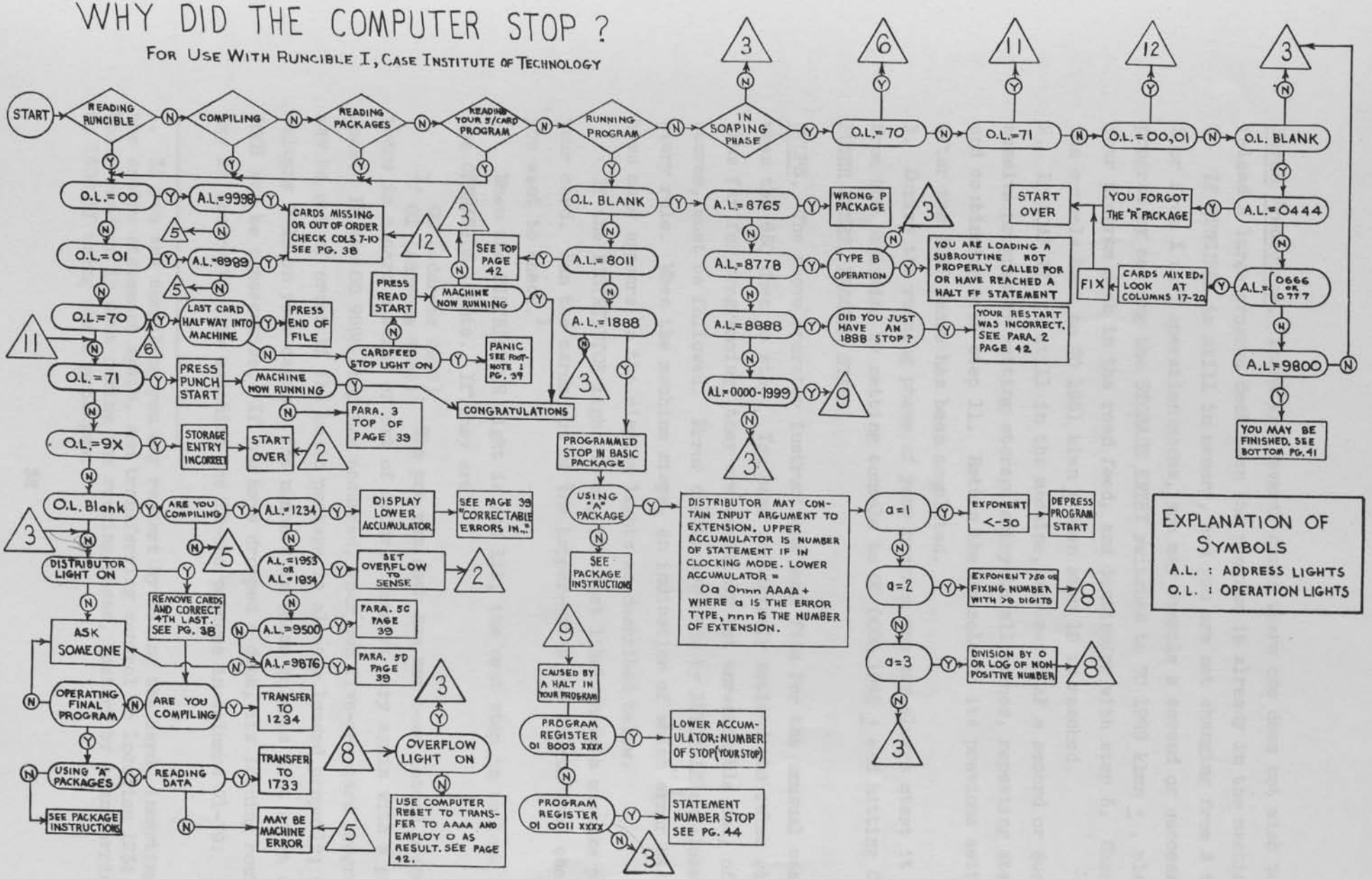
MACHINE SHOULD STOP READING HERE. SEE STEP 16, PAGE 26.





WHY DID THE COMPUTER STOP ?

FOR USE WITH RUNCIBLE I, CASE INSTITUTE OF TECHNOLOGY



Rerun Procedures. There are several cases where one does not wish to reload a large program deck when the program is already in the machine.

1. If RUNCIBLE is still in memory, and you are not changing from A to B or from X to Y operation modes, you may compile a second or successive program by setting the STORAGE ENTRY switches to 70 1999 klmn ± , placing your program deck in the read feed, and continuing with step 5. Change the console back to 70 1951 klmn ± when step 10 is reached.

2. If SOAP III is still in the machine, you may SOAP a second or successive program by setting storage entry to all zeroes, repeating step 5 and continuing with step 11. Return the console to its previous setting after the SOAP phase has been completed.

3. During the running phase of your program you may always start it over from the beginning by setting console to 00 0000 1999 ± and hitting COMPUTER RESET, PROGRAM START.

STOPS. The above operator instructions work fine for the unusual case when the 650 does no stop. You have undoubtedly noticed the rules given thus far for programming; they are certainly not unreasonable and, of course, must be followed. Error checks are made by RUNCIBLE on almost every rule. When the machine stops, an indication of which error has been made appears in the display lights as described below.

If the DISTRIBUTOR light is on, you most likely have a mistake punching your card. Run the cards out of the hopper--it's the fourth last one out you want to check.[1]

When the DISTRIBUTOR light is not lit, the next step is to look at the OPERATION lights. If they are

1. 00 (address 9998)
01 (address 8989): The program deck has not been loaded properly; there is a card missing or out of order. Remedy: try again with a good deck. If the 00 9998 stop has occurred, your own five-per-card program may be out of order if the basic package is already loaded properly; check columns 7-10 on your cards. The numbers in these columns must start at 0001 and be consecutive. If you have dropped a deck, its loading routine may be unscrambled by watching the 12-overpunches in columns 71-79.

---

1. If you are compiling you may restart by fixing the card, inserting the entire statement again, and transferring control to location 1234. While reading in data during the running phase, restart by transferring to 1698 if using the A packages.

2. 70: The machine wants to read a card. Depress read START, or END OF FILE if the last card in the hopper is halfway in; however, if the CARD FEED STOP light is on, panic.[1]

3. 71: The machine wants to punch a card. Depress punch START. If this doesn't work, there is a bent card in the punch feed or the feed is filled to capacity. In the latter case, relax and take all cards out of the feed as the first portion of your output. But if there was a bent card, remove it and play around until the unlabeled light goes out. You will have now one or more extra garbage cards (see step 8 in operator instructions) in the middle of your output which must be removed.

4. In the 90's: You have not set the storage entry switches correctly.

5. Blank: Now you must check the ADDRESS lights, as follows: Compilation phase. A. 1234: This is a correctable error in a RUNCIBLE statement; error cards are punched for this type of mistake when in error search mode (see Operation Modes). A list of these goofs will be given below. To restart, correct the offending statement, place it first into the read hopper, and depress PROGRAM START and read START.

B. 1953, 1954: Turn the OVERFLOW switch to SENSE and run the deck again.

C. 9500: Columns 31-40 of your Header Card are negative or zero; or storage exceeded and not in error search mode.[2] When tight for memory space in a long program, always check by an error search first. Remedy: change your header card and start over.

D. 9876: If your entire deck of statements has been processed, cheer up, for this is no mistake; the compilation mode has been successfully completed. If not, however, an FF occurred on the last statement processed; or you are using B-operation and columns 71-80 of your Header Card are not zero. Remedy: start again.

CORRECTABLE ERRORS IN STATEMENTS: (1234 stop or error search output) If a 1234 stop occurs and the DISPLAY switch is turned to LOWER ACCUM, your error indication will be as follows:

0 n n n 0 a a 0 b b

where nnn is the number of the offending statement and aa and bb are

1. You have a bent card. Clear the read feed, and replace the last two cards out into the hopper. Fix up the bent card and replace the deck; depress STOP and then START (both keys).
2. On this error a display is given as in the 1234 stops (see 02 02 stop).

explained in the table below. You may correct your statement immediately by making up the new card or cards necessary and replacing the incorrect statement, putting your cards back in the read hopper, pushing read START on the 533 and PROGRAM START on the 650.

If you are in error search mode, list any output cards on a tabulator with the SOAP III control panel. Your statement number will appear, followed by an entry in the table below, followed by that portion of your statement which caused the trouble.

Here is the error table — if your error indication does not appear to make any sense, check to see if you have punched the F's correctly in column 70.

Letters:	aa	bb	Error type:
	00	00	A zero statement number immediately following an iteration statement; or more than five cards in a statement; or statement too long--over 100 machine language instructions were necessary to execute your statement (remedy: break it into two statements).
	01	01	Subscript or statement number over 999.
	02	02	Problem too long--2000 memory locations exceeded. You must start over if not in error search mode. Either cut down numbers on header card to bare minimum or shorten or segment your program.
	04	04	Constant larger or smaller than allowable. (fixed point $\geq$ 1 billion or floating point $< 10^{-50}$ or $\geq 10^{50}$)
	06	06	More than ten subroutines used in addition to the basic package when using type B operation.
	07	07	Column 43 on first card of a substitution statement not I, Y, or C; or too many named subroutines used in one statement; or parentheses and quotation marks nested over nine deep.
	08	08	Unmatched parentheses or quotation marks; or a floating point number when it should have been fixed; i.e., in a statement number, an exponent, or a subscript.
E R	65	79	"CONVERT" when not using type A operation.
G F	67	66	"ARITHMETIC FLOATING" when not using type AX operation.

Letters:		aa	bb	Error types (continued)
I	F	69	66	"IF" not followed properly by one of the relations U, V, or W.
K	F	72	66	Iteration statements nested over four deep; or more than five symbols between commas in an iteration statement; or too many commas in an iteration statement; or bad spelling at the beginning of a matrix definition statement.
L	F	73	66	"ARITHMETIC DECIMAL" when not using type AX operation.
R	L	79	73	Improper use of REMAINDER (see Appendix III); or the invalid pair)(in a statement.
O	K	90	72	PUNCHing more than four variables in one statement.
O	O	90	90	Erroneous substitution statement; or statement does nothing. Examples: I1⇐1⇐1; I1+1⇐1; I1.

..... and if not in the list so far,

?	K	??	72	Too many multiple input extensions nested together; or HALT followed by a variable.
?	?	αα	ββ	The code for αα and ββ is explained in the following table; they are two letters which should not appear next to each other in any meaningful RUNCIBLE statement:

61	A		76	O	
62	B		77	P	
63	C		78	Q	"
64	D	/	79	R	)]} or U, V, W
65	E		82	S	+ or M, U, V, W
66	F	or the word IF	83	T	or the word PUNCH
67	G		84	U	=
68	H		85	V	>
69	I		86	W	≥
71	J	.	87	X	x
72	K	, or T or the word PUNCH	88	Y	
73	L	([{ or A,C,I,Q,Y, or R...R	89	Z	⇐
74	M	-	90	lefthand end of statement[1]	
75	N		99	any integer 0-9 or a J	

SOAPing phase. Assuming that properly written extension are being used, no stops should ever occur until the end of the SOAP III phase. However, if the OPERATION lights are blank you may have a stop of

0444: You forgot the reservation package.

0666, 0777): You got the compiler output jumbled up somehow.

9800: You have successfully finished SOAPing; or (if all cards are not processed) you forgot to remove a garbage card (caused by a punch jam) from the middle of your deck. See 71 stop.

1. In this case, ββ represents the first symbol of the statement.

Running Phase. 1. OPERATION lights not blank. ADDRESS lights:

8011: Overflow or underflow has occurred somewhere during floating point arithmetic while using Y operation. The statement on which it occurred is in machine location 0018 if clocking.

1888: Programmed stop in basic package. When using "A" packages, the DISPLAY switch may be dialed to indicate:

DISTRIBUTOR — May or may not contain the input argument to the extension; when it does not, it is the same as the UPPER ACCUM.

UPPER ACCUM — The number of the statement being executed; zero if clocking not used.

LOWER ACCUM — Displays the ten-digit number

0a 0nnn AAAA +

nnn is the number of the extension;[1] a is the error type which may be found by referring to the writeup of the extension. In the basic packages,

a = 1: Floating point exponent less than -50; pushing PROGRAM START will continue calculation using zero as a result.

a = 2: Floating point exponent more than +49; or attempting to fix a number of more than eight digits.

a = 3: Division by zero or logarithm of a non-positive quantity.

When the OVERFLOW light is not on, the program may be restarted by setting the ADDRESS SELECTION switches equal to AAAA, turning CONTROL to MANUAL OPERATION, punching COMPUTER RESET and TRANSFER, turning CONTROL back to ADDRESS STOP, resetting address switches to 1888, and hitting PROGRAM START. Zero will be employed as an answer to the calculation.

xxxx: Operation lights not blank, address not 1888 or 8011: If Overflow light is on and DISTRIBUTOR contains zero, you are attempting fixed point division by zero.

2. OPERATION lights blank. ADDRESS lights:

8765: Wrong basic package for your program. Dial PROGRAM REGISTER -- it contains 01 0nnn 8765 where nnn is the extension missing from the package you are using.

8778: (Type B Operation) You are attempting to load a subroutine in relocatable form which you did not properly call for in your program, or HALT FF statement.

---

1. See next page for extension numbers in the basic packages, and extension writeups for the number of a named extension.

8888: You had a 1888 stop with OVERFLOW light not on and did not restart as described above, or you forgot to turn the switch to Address Stop at Step 16.

0000-1999: Stop caused by HALT statement in your program; if you are displaying a number it is in LOWER ACCUM.[1] PROG REGISTER should contain 01 8003 xxxx.

Extensions in the basic packages which include error stops have the following numbers:

nnn =
- 001: Log to base 10--not in P1
- 002: 10 to the X--not in P1
- 006: Floating point division
- 008: Floating point addition and subtraction
- 009: Floating point multiplication
- 010: Fl. pt. number to fl. pt. power (P)--not in P1
- 011: Clocking
- 018: Natural logarithm--not in P1
- 019: e to the X--not in P1
- 020: Square root--not in P1 or P2
- 021: Sine--not in P1 or P2
- 022: Cosine--not in P1 or P2
- 023: Tangent--not in P1 or P2
- 033: Arctangent--not in P1 or P2
- 500: Fix a floating point number
- 501: Fx. pt. number to fx. pt. power (P)--not in P1

"IF NOTHING ELSE WORKS, READ THE INSTRUCTIONS"

---

1. If this number is 01 0011 xxxx, you have a statement number stop (see Appendix I.

APPENDIX I

EXTRA CLOCKING FEATURES. While doing minimum clocking, the present statement number is kept in machine location 0017; the previous statement number may be found in location 0018. The following features may also be used while clocking -- if any or all of these are used, an additional 98 drum locations are needed. (These features are primarily intended for those who have a more complete knowledge of 650 machine language.)

1. Statement number stopping. The computer may be stopped just before executing any statement whose number is nonzero.

2. Flow tracing. A card will be punched just before executing every non-zero statement showing the contents of the accumulators.

3. Complete tracing. A card will be punched for every machine language instruction executed except those in extensions, showing contents of the accumulators and distributor.[1] Extensions will be run at full speed. Tracing may be stopped or started at any statement number.

CAUTION: Iteration statements generate several zero statements which when executed are not clocked or flow traced.

To use any of these extra features, digits m and n (see Operation Modes) should be set to 8 while compiling, and the following console setting is to be made while running the machine language program:

q r s f A A A B B B ±

The sign controls conditional PUNCH statements as before, and digit f is free for some use which is as yet unforeseen. Clocking is independent of the sign or of digit f. Digit q tells what features are to be used:

q = 0 or q > 4: Minimum clocking only.

q = 1: Statement number stopping and clocking.

q = 2: Flow tracing, statement number stopping, and clocking.

q = 3: Complete tracing, statement number stopping, and clocking.

q = 4: Complete tracing, flow tracing, statement number stopping, and clocking.

---

1. Complete tracing will correctly trace all RUNCIBLE output instructions but will not handle branch-distributor-8 commands, references to 8001 or 8003 for instructions, or division with opposite signs. When "doctoring" of compiler output is done, therefore, care should be used to stay away from these instructions if complete tracing is also desired.

Statement number stops: The machine will halt just before executing statement number AAA if digit r is 8, and r must be 9 when this stop is not desired. The machine will halt before executing statement BBB if digit s is 8, and s must always be 9 when this stop is not desired. The halt command will have data address 0011, and the DISTRIBUTOR will contain the statement number on which the stop occurred. The accumulators do not contain anything of interest. To continue, depress PROGRAM START.

Complete tracing will start up when statement AAA is encountered and will be discontinued just before executing statement BBB. (If AAA equals BBB, no tracing will occur.) Digit q may be changed while running the program, but its effect will be noticeable only after the present statement has been completed. Never change the console setting unless the 650 is stopped.

Card formats:	Flow tracing:	Complete tracing:
Word 1:	FLO the letters	COM
Word 2:	000000nnnn (nnnn is statement number; kkkk is location of instruction.)	00kkkknnnn
Word 3:	0000000000	Instruction
Word 4:	UPPER ACCUMULATOR	
Word 5:	LOWER ACCUMULATOR	
Word 6:	0000000000	
Word 7:	0000000000	DISTRIBUTOR
Word 8:	0000000000	

When flow tracing with type Y operation, word 7 of the output cards will contain the contents of the DISTRIBUTOR if complete tracing is not being used simultaneously. Results of every substitution statement may be found in the distributor, while only certain types of substitution statements leave the answer in the lower.

APPENDIX II

MATRIX NOTATION. There are two types of doubly-subscripted notation available in RUNCIBLE I, and since they have somewhat different rules, they will be discussed separately here. It is possible to program matrix problems without this notation, but the machine language output on matrix notation is usually much better and the notation is convenient and quite easy to use.

1. VARIABLE-SIZE MATRICES. Two matrices having variable row length may be used: the YN matrix and the CN matrix. The number of columns in the YN matrix, whenever it is used, is always contained in the current value of variable I1, and I2 contains the number of columns of the CN matrix. Notation used in statements is

YN(v1 , v2)

where v1 (the row) and v2 (the column) are any fixed point mathematical expressions. Variable-size matrices start with row zero and column zero, and they overlap the other Y- and C- variables. Y0 is the same as YN(0,0) and if I2 = 4, say, CN(1,0) is variable C4. This may be seen more clearly in the following equivalence diagram, showing in this case a 4 x 3 matrix:

$$\left\| \begin{matrix} YN_{0,0} & YN_{0,1} & YN_{0,2} \\ YN_{1,0} & YN_{1,1} & YN_{1,2} \\ YN_{2,0} & YN_{2,1} & YN_{2,2} \\ YN_{3,0} & YN_{3,1} & YN_{3,2} \end{matrix} \right\| \quad I_1 = 3 \quad \left\| \begin{matrix} Y_0 & Y_1 & Y_2 \\ Y_3 & Y_4 & Y_5 \\ Y_6 & Y_7 & Y_8 \\ Y_9 & Y_{10} & Y_{11} \end{matrix} \right\|$$

2. FIXED-SIZE MATRICES. Fixed-size matrices are the type most frequently found in compilers prepared for other computers; the row size must be specified by the statements of the program. There are ten of these matrices available: YN1, YN2, ..., YN5, CN1, ..., CN4, and CN5. These matrices start with the more conventional element 1,1 in the upper left-hand corner. Notation used in statements is

YN2(v1 , v2)

where v1 (the row) and v2 (the column) are any fixed point mathematical expressions. Y- and C- variables are overlapped by the fixed-size matrices; the subscript number where each matrix will start (element 1,1 of the matrix) is supplied by the programmer.

A matrix definition statement must precede the use of any fixed-size matrices, or erroneous (undetected) operation will occur. Such a statement looks like this:

YNn, COLS, c, START, s,
or CNn, COLS, c, START, s,

where n is a number from 1 through 5, c is the number of columns in the matrix, and s is the subscript number which should correspond to element YNn(1,1). Note the commas and English words in the statement; c and s must be fixed-point numbers without leading zeroes. A matrix definition statement cannot be made conditional, of course; when the program is running it is treated like a BYPASS statement, and so it is usually given statement number zero.

The equivalence diagram below illustrates the overlay of a 4 x 3 matrix of fixed size.

$$\left\| \begin{matrix} CN3_{1,1} & CN3_{1,2} & CN3_{1,3} \\ CN3_{2,1} & CN3_{2,2} & CN3_{2,3} \\ CN3_{3,1} & CN3_{3,2} & CN3_{3,3} \\ CN3_{4,1} & CN3_{4,2} & CN3_{4,3} \end{matrix} \right\| \left\| \begin{matrix} \text{COLS, 3,} \\ \text{<=======>} \\ \text{START, 32,} \end{matrix} \right\| \left\| \begin{matrix} C_{32} & C_{33} & C_{34} \\ C_{35} & C_{36} & C_{37} \\ C_{38} & C_{39} & C_{40} \\ C_{41} & C_{42} & C_{43} \end{matrix} \right\|$$

A sample matrix problem is given in Appendix VI.

APPENDIX III

EXTRA PROGRAMMING FEATURES. RUNCIBLE I contains, in addition to those features described in the main portion of this manual, some secret hidden tricks which are extra added attractions for the more advanced programmer.

1. Remainder. The remainder of a fixed point division may be saved instead of the quotient by enclosing the division and parentheses and preceding it with the word REMAINDER; e.g.,

REMAINDER (I3 / 5)

The expression inside the parentheses must be some fixed point division — it may have a more complicated dividend and divisor as

REMAINDER [(I8+2)/(I3P6)]

When used in a statement it is treated by RUNCIBLE as though the entire thing were enclosed in parentheses, so parentheses need not be placed again around the entire remainder expression. REMDR or simply RR may be used as an abbreviation for REMAINDER.

2. Variables in the core. Your variables may be placed into immediate access (core) storage for faster operation by adjusting the Header Card. Word 1, 2, or 3 of the header may be made negative (an 11-punch replacing the 12-punch) and the corresponding variables, I, Y, or C respectively, will be placed into the core.[1] There are a few restrictions, however: no matter which variables, I, Y, or C, are placed in core, that one with subscript zero is put into 9000, subscript one into 9001, etc., so that no subscript greater than 59 may be used. Also if more than one type of variable is put into core, overlapping occurs between those having equal subscripts, and so care must be taken that no subscripts are duplicated between these two variables. For example, if both the I's and the Y's are placed in core, and I1 is used in the program, Y1 may not be used at the same time since the two variables are both kept in location 9001.

3. Switching arithmetic. This is another step forward in compiler offerings -- the floating point variables may be changed into a fixed decimal point form with the number of decimal places to be saved

1. The other numbers in a negative word are ignored.

left as a variable. This is useful in some calculations for speed of operation and for editing answers before punching, making them more easily read. Switching arithmetic can be done only when using type A and type X operation modes simultaneously. A group of extensions is supplied for this operation; a full description of how to use these features completely is given in their writeup. The compiler statements ARITHMETIC DECIMAL and ARITHMETIC FLOATING will change from floating decimal to fixed decimal arithmetic or back from fixed to floating, respectively; they may not be made conditional. Only the arithmetic of Y- and C- variables is affected. During the running phase, variable IO (I-zero) will specify the number of decimal digits desired; this can conveniently be changed from one run to another until the correct scale for the problem is determined.

4. Error Sense Running. For production runs it may be found economical to leave the ERROR switch on SENSE and then have a built-in error restarting procedure in the program to bypass machine errors when they occur. This may be done with RUNCIBLE, as follows:

The error correction procedure is programmed as a block of statements, and on the first statement of the restarting program the number of the statement on which the error occurred is by now in location 0018 if using minimum clocking. This value may be programmed by asking for variable I(-1) if the I's are not in core; if they are in core, call for I(-8982). For example, the first statement might be

I6 ← I(-1)

The error restart routine which you program will continue then perhaps by attempting to correct any harm done, keeping count to see if the error occurs repeatedly, and then going back into the main program.

This operating mode is set up by the statement

SET ERROR CORRECTION TO n

where n is the (fixed-point constant) number of the statement at which the correction procedure should start if an error is sensed. This number may be reset during different portions of the program by other similar statements. The SET ERROR CORRECTION statement must

precede any part of the program it is supposed to be operative on; it may be made conditional. During the running phase of the program the console switches must be set to NOP to 1998 to utilize this error sense operation.[1]

5. Spelling liberties. RUNCIBLE is not always very fussy about the programmer's spelling of English words (this will come as a relief to many people):

READ is any statement ending with a D (e.g., READ A CARD, or DONT READ)

HALT is any statement ending with a T or H; HALT n is identified by the T-integer combination.

BYPASS is any statement ending with an S.

PUNCH may be replaced by the letter T; it may also be any group of letters or numbers starting with P and ending with H as long as there is only one P. (e.g., PINCH, or PEANUT GOULASH)

REMAINDER is anything beginning and ending with R without any R's in the middle.

JUMP TO is anything ending in G or O not containing an E.

SET ERROR CORRECTION TO is anything ending in G or O containing the letter E. (e.g., RESTARTING)

IF and THRU must be spelled exactly.

COLS and START are not looked at but they must be less than six symbols in length.

ARITHMETIC DECIMAL and ARITHMETIC FLOATING are anything ending in L or G, respectively.

READ PROGRAM is anything ending in M.[2]

STATISTICAL READ is identified by the letters EAD or ED in sequence followed by a number or variable.[2]

EDIT is anything ending in IT.[2]

CONVERT is anything ending with the letters ERT in sequence.[2]

6. Passing Soap Cards. (Type A Operation Only) Any number of cards written in standard SOAP III coding may be placed just after the Comments Card of a RUNCIBLE input program to be reproduced in the output. They will appear in the place where extensions are normally to be inserted when Soaping (see Operator Instructions, Step II). Column 5 must not contain a 12-punch on these cards.

---

1. Notice that digit f (see clocking) still is arbitrary if we ever find a use for it.
2. See these extensions for use of the words.

APPENDIX IV

CORRECTING ERRORS AFTER COMPLETING COMPILATION. Notes this procedure may be used only with type A operation. If only a small number of statements in a large program have to be changed after compilation, first make a listing of the SOAP III symbolic program. At the very end will be a list of constants terminating with ABC00 (or ABD00, etc.) -- the symbolic locations will be in decreasing order. Take the largest number following the ABC[1] and put it into columns 71-80 of a Header Card which is identical in all other respects with an ordinary Header Card. (The control information in the other columns of this Header Card may have to be changed; this information must apply to the whole program, not just the corrections.)

Now compile in the regular manner the statements which correct or supplement your original output. Put FF on the card of the final statement compiled. The first few cards of your output -- identified by 12- punches in columns 76-80 -- should now replace the corresponding cards of the original program. Remove also the old symbolic program cards corresponding to the statements you want to replace -- they can be located by checking the comments portion of the listing -- and insert the new material into its proper place. Each statement begins with the characteristic "statement dictionary entry" which has location address (S0000+n) where n is the statement number. Append any new constants (identified by location address ABC, ABD, ...) at the end of the program. Continue then with the SOAP phase.

Caution: iteration statements become three statements when compiling, one to initialize the variable and the other two (numbered zero) to increment and test it, occurring just after statement n, the end of the iteration scope.

When compiled with clocking, linkages from statement to statement are made with statement numbers, and so the first statement in any block of consecutive correcting statements must have the same number as the first statement it replaces, and the final statement in this block must be followed with a BYPASS statement having the

1. If it is ABD, add an extra hundred; ABE, two hundred; and so on.

same number as the statement following the replaced statements. Therefore a single statement may not be merely added into the listing when compiled with clocking; one statement must always be removed (and recompiled in this case) whenever a change is made. Example: suppose it is desired to insert a new statement 20 between old statements 4 and 5. Remove statement 4 from the listing; compile statements 4 and 20 and a dummy statement 5: BYPASS in that order. Put the output from statements 4 and 20 into the place formerly occupied by statement 4.

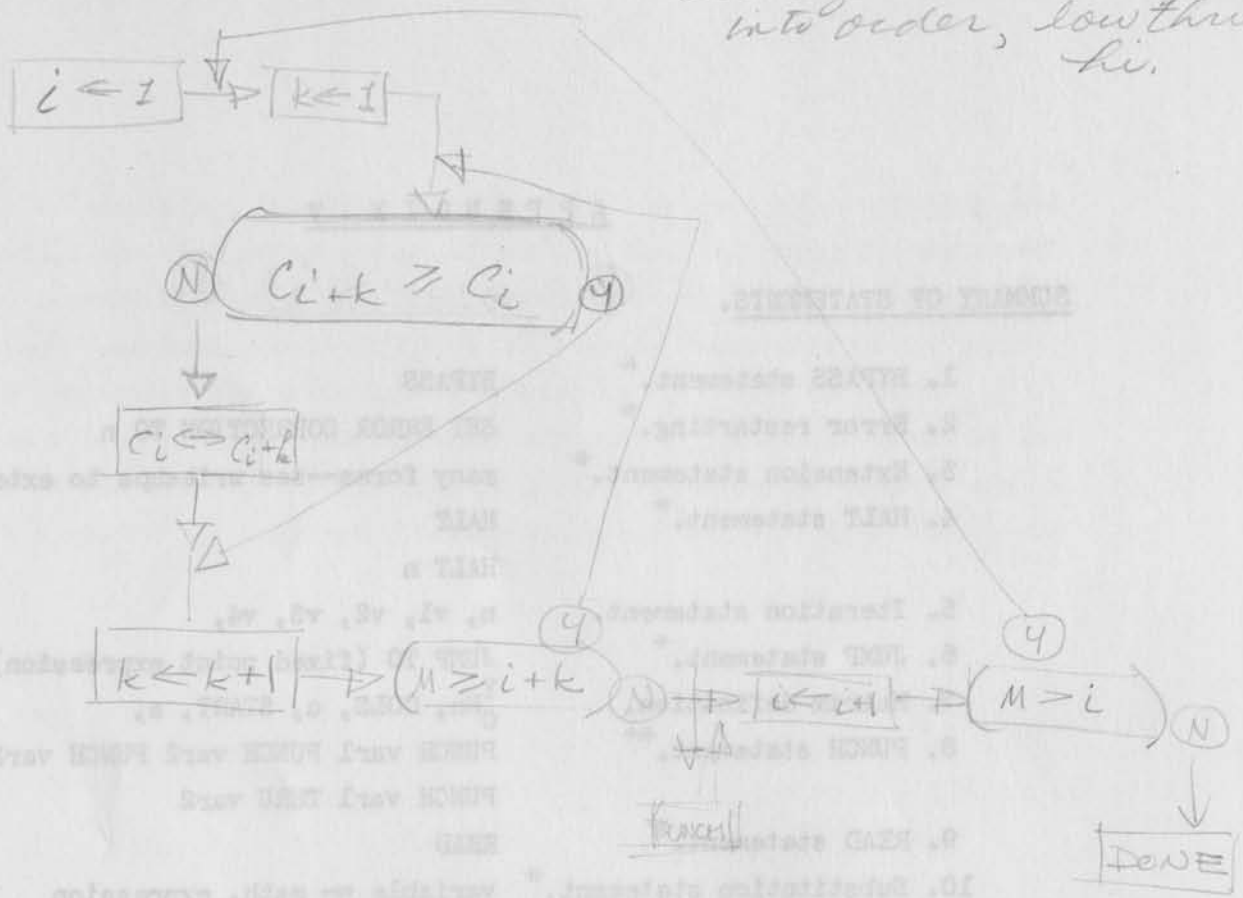
APPENDIX V

SUMMARY OF STATEMENTS.

1. BYPASS statement.*	BYPASS
2. Error restarting.*	SET ERROR CORRECTION TO n
3. Extension statement.*	many forms--see writeups to extensions
4. HALT statement.*	HALT
	HALT n
5. Iteration statement.	n, v1, v2, v3, v4,
6. JUMP statement.*	JUMP TO (fixed point expression)
7. Matrix definition.	$\begin{matrix} Y \\ C \end{matrix}$Nn, COLS, c, START, s,
8. PUNCH statement.*+	PUNCH var1 PUNCH var2 PUNCH var3 PUNCH var4
	PUNCH var1 THRU var2
9. READ statement.*	READ
10. Substitution statement.*	variable ← math. expression
11. Switch of arithmetic.	ARITHMETIC DECIMAL
	ARITHMETIC FLOATING

* May be made conditional by adding IF expression $\begin{matrix} = \\ > \\ \geq \end{matrix}$ expression*
+ May be made conditional on sign of console switches by giving statement number zero.

arrange M numbers into order, low thru hi.



A P P E N D I X V I

ADDITIONAL SAMPLE PROGRAMS. (These programs are partly pilfered from the original Carnegie Tech compiler manual; they are chosen for their educational value.)

Example Sub-Program 1. Separate the integral and fractional parts of a floating point variable Y1.

```
1       I1 ← Y1
2       C1 ← I1
3       C2 ← Y1 - C1
```

I1 is the integral part of Y1 in fixed point form; C1 is the same in floating point form; and C2 is the fractional (decimal) part of Y1 in floating point form.

Example Sub-Program 2. Represent the eight significant digits of a floating point variable Y1 as a fixed point integer I1.

```
1       JUMP TO 5 IF Y1 ≥ 1 B 8
2       JUMP TO 7 IF 1 B 7 > Y1
3       I1 ← Y1
4       HALT
5       Y1 ← Y1 / 10.
6       JUMP TO 1
7       Y1 ← Y1 x 10.
8       JUMP TO 2
```

Trickier, shorter, and probably slower, would be

```
1       JUMP TO 5 IF 1 B 8 > Y1 IF Y1 ≥ 1 B 7
2       Y1 ← Y1 / 10. IF Y1 ≥ 1 B 8
3       Y1 ← Y1 x 10. IF 1 B 7 > Y1
4       JUMP TO 1
5       I1 ← Y1
6       HALT
```

Example Sub-Program 3. Find the maximum of a set of I1 numbers; the numbers are Y- variables with consecutive subscripts; YI2 is the first variable of the group. I1 > 1.

```
1    C1 ← YI2
2    I3 ← I2
3    I1 ← I2 + I1 - 1
4    8, I4, I2 + 1, 1, I1,
5    JUMP TO 8 IF C1 ≥ YI4
6    C1 ← YI4
7    I3 ← I4
8    BYPASS
9    JUMP TO I5
```

C1 is the desired maximum and I3 is the subscript of the maximum Y of the set. Statement 8 was necessary to end each iteration on a common statement. Statement 3 was necessary because the expression I2 + I1 - 1 was too long to include in iteration statement 4. Note statement 9 at the end -- a variable JUMP statement enabling the main program to use this sub-program several times and exit to different statements for continuation.

Example Program 3. Generalized matrix multiplication of up to a 20 x 20 matrix. Multiply the I6 x I2 matrix YN1 by the I4 x I6 matrix CN1 obtaining the product elements by the relation

$$Y_1 = \sum_{I5=1}^{I6} CN1(I3,I5) \times YN1(I5,I1)$$

punching the elements as they are being computed.

```
0    CN1, COLS, 20, START, 25,
0    YN1, COLS, 20, START, 25,
0    READ
7    4, I1, 1, 1, I2,
1    4, I3, 1, 1, I4,
6    Y1 ← 0.
2    3, I5, 1, 1, I6,
3    Y1 ← Y1 + CN1(I3,I5) x YN1(I5,I1)
4    PUNCH Y1 PUNCH I3 PUNCH I1
8    HALT
```

The various outputs that may be obtained with this program are illustrated in Appendix VII. Notice nesting of iteration statements.

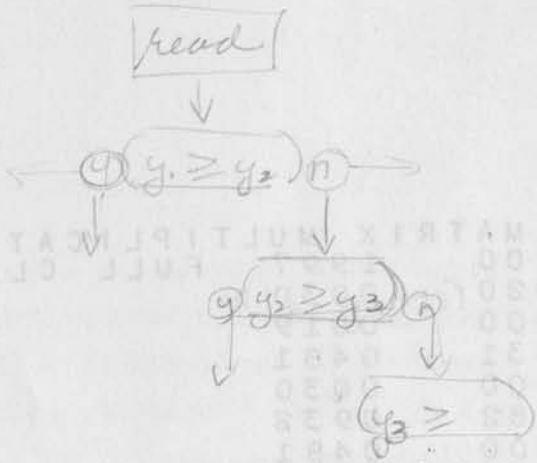
A P P E N D I X   V I I

LISTINGS OF OUTPUT. On the following pages are shown the input to and output from RUNCIBLE I as it works on Example Problem 3. First the program deck is shown. Due to the shortness of the problem, no attempt was made to keep the number of variables down to a minimum. Word 1 of the header card was made negative for the type Y operation. Statement number 3 would have fit onto a single card but the extra spaces were added for clarity.

```
         -10       -20       -30       -40       -50       -60       -70       -80
00000000100000000450000000045000000000100000000000000000325000000000000000000000

                                             EXAMPLE3 MATRIX MULTIPLICATION
0000I                                        CN1K COLSK 20K STARTK 25K  F
00001                                        YN1K COLSK 20K STARTK 25K  F
0000I                                        READ                       F
0007I                                        4K I1K 1K 1K I2K           F
0001I                                        4K I3K 1K 1K 14K           F
0006I                                        Y1 Z 0J                    F
0002I                                        3K I5K 1K 1K I6K           F
0003I                                        Y1 Z Y1 S CN1LI3K I5R X YN1LI5
    I                                        K I1R                      F
0004I                                        PUNCH Y1 PUNCH I3 PUNCH I1 F
0008I                                        HALT                      FF
```

```
1          EXAMPLE3 MATRIX MULTIPLICATION
                BLR    1900      1997    FULL  CLOCK
                REG    I0020     0030
     0001         00     0000     0019
                REG    Y0031     0481
     0002         00     0000     0030
                REG    C0482     0932
     0003         00     0000     0481
                REG    S0933     0942
     0004         00     0000     0932
                REG    T0943     0952
     0005         00     0000     0942
   GO  TO       NOP     8003     1F
   S0000        LOD    1         E  011    CN1K  COLSK
   1            NOP     8003     1F          20K  START
   S0000        LOD    1         E  011    YN1K  COLSK
   1            NOP     8003     1F          20K  START
   S0000        LOD    1         E  011    READ
   1            LOD    S0007     E  016
   S0007        LOD    1         E  011    I1     Z
   T0007        LOD    ABC00                1
                STD    I0001     S0001
   S0001        LOD    1         E  011    I3     Z
   T0001        LOD    ABC00                1
                STD    I0003     S0006
   S0006        LOD    1         E  011    Y1  Z  0J
   T0006        LOD    ABC01
                STD    Y0001     S0002                F
   S0002        LOD    1         E  011    I5     Z
   T0002        LOD    ABC00                1
                STD    I0005     S0003
   S0003        LOD    1         E  011    Y1  Z  Y1  S
   T0003        RAU    I0005                CN1LI3K  I5
                MPY    ABC02                R  X  YN1LI5
                ALO    I0001                K  I1R
                SLT     0004
                ALO                8002              F
                LOD    Y0004
                STD    W0000
                RAU    I0003
                MPY    ABC02
                ALO    I0005
                SLT     0004
                ALO                8002
                RAL    C0004
```


read
$y_1 \geq y_2$
y
n
$y_2 \geq y_3$
y
n
$y_3 \geq$

```
        LOD  W0000
        STD  ACC
        LOD            E 009
        RAL  Y0001
        LOD            E 008
        STL  Y0001     1F
S0000   LOD  1         E 011    I5     Z     L
1       RAL  I0005              1      R   S
        ALO  ABC00              I5
        STL  I0005     1F
S0000   LOD  1         E 011    GO  TO  0003
1       RSL  I0005              IF    LI6
        ALO  I0006              R   W  I5
        BMI  S0004     S0003
S0004   LOD  1         E 011    PUNCH  Y1  P
T0004   LOD  ABC03              UNCH  I3  PU
        STD  P0000              NCH  I1  F
        LOD  ABC04
        STD  P0001
        LOD  ABC05
        STD  P0002
        RAL  ABC06
        LOD  1F        E 017
S0000   LOD  1         E 011    I3     Z     L
1       RAL  I0003              1      R   S
        ALO  ABC00              I3
        STL  I0003     1F
S0000   LOD  1         E 011    GO  TO  0006
1       RSL  I0003              IF    LI4
        ALO  I0004              R   W  I3
        BMI  1F        S0006
S0000   LOD  1         E 011    I1     Z     L
1       RAL  I0001              1      R   S
        ALO  ABC00              I1
        STL  I0001     1F
S0000   LOD  1         E 011    GO  TO  0001
1       RSL  I0001              IF    LI2
        ALO  I0002              R   W  I1
        BMI  S0008     S0001
S0008   LOD  1         E 011    HALT
T0008   HLT    8003      8778
ABC06     00     0003      0004
ABC05     00     0002      0001
ABC04     00     0001      0003
ABC03     00     0001      0001
ABC02     00     0000      0020
ABC01     00     0000      0000
ABC00     00     0000      0001
```

```
1          EXAMPLE3 MATRIX MULTIPLICATION
               REG  I9001
    0001         00    0000      9000
               REG  Y0020        0470
    0002         00    0000      0019
               REG  C0471        0921
    0003         00    0000      0470
               REG  S0922        0931
    0004         00    0000      0921
               REG  T0932
    0005         00    0000      0931
   GO  T0      NOP    8003    1F
   S0000       NOP  1          1F        CN1K  COLSK
   1           NOP    8003     1F          20K  START
   S0000       NOP  1          1F        YN1K  COLSK
   1           NOP    8003     1F          20K  START
   S0000       NOP  1          1F        READ
   1           LOD  1F         E  016
   S0007       NOP  1          1F        I1    Z
   1           LOD  ABC00                1
               STD  I0001      1F
   S0001       NOP  1          1F        I3    Z
   1           LOD  ABC00                1
               STD  I0003      1F
   S0006       NOP  1          1F        Y1  Z  0J
   1           LOD  ABC01
               STD  Y0001      1F                  F
   S0002       NOP  1          1F        I5    Z
   1           LOD  ABC00                1
               STD  I0005      1F
   S0003       NOP  1          1F        Y1  Z  Y1  S
   1           RAU  I0005                CN1L I3K  I5
               MPY  ABC02                R  X  YN1L I5
               ALO  I0001                K  I1R
               RAA    8002
               LOD  Y0004A                     F
               STD  W0000
               RAU  I0003
               MPY  ABC02
               ALO  I0005
               RAA    8002
               RAU  C0004A
               FMP  W0000
               FAD  Y0001
               STU  Y0001      1F
```

```
S0000   NOP  1          1F          I5     Z     L
1       RAL  I0005      1           1      R     S
        ALO  ABC00      I5
        STL  I0005      1F
S0000   NOP  1          1F          GO  TO  0003
1       RSL  I0005                  IF    LI6
        ALO  I0006                  R   W  I5
        BMI  1F         S0003
S0004   NOP  1          1F          PUNCH  Y1  P
1       LOD  ABC03                  UNCH  I3  PU
        STD  P0000                  NCH  I1  F
        LOD  ABC04
        STD  P0001
        LOD  ABC05
        STD  P0002
        RAL  ABC06
        LOD  1F         E  017
S0000   NOP  1          1F          I3     Z     L
1       RAL  I0003                  1      R     S
        ALO  ABC00                  I3
        STL  I0003      1F
S0000   NOP  1          1F          GO  TO  0006
1       RSL  I0003                  IF    LI4
        ALO  I0004                  R   W  I3
        BMI  1F         S0006
S0000   NOP  1          1F          I1     Z     L
1       RAL  I0001                  1      R     S
        ALO  ABC00                  I1
        STL  I0001      1F
S0000   NOP  1          1F          GO  TO  0001
1       RSL  I0001                  IF    LI2
        ALO  I0002                  R   W  I1
        BMI  1F         S0001
S0008   NOP  1          1F          HALT
1       HLT   8003       8778
ABC06    00   0003       0004
ABC05    00   0002       0001
ABC04    00   0001       0003
ABC03    00   0001       0001
ABC02    00   0000       0020
ABC01    00   0000       0000
ABC00    00   0000       0001
```

```
1          EXAMPLE3 MATRIX MULTIPLICATION                                1
                REG  I9001                                               2
     0001         OO   0000     9000                                     3
                REG  Y0020     0470                                      4
     0002         OO   0000     0019                                     5
                REG  C0471     0921                                      6
     0003         OO   0000     0470                                     7
                REG  S0922     0931                                      8
     0004         OO   0000     0921                                     9
                REG  T0932     0941                                     10
     0005         OO   0000     0931                                     11
   GO  TO   NOP    8003   1F                                             12
   S0000    RAA  1         E  011   CN1K  COLSK                         13
   1        NOP    8003   1F         2OK  START                         14
   S0000    RAA  1         E  011   YN1K  COLSK                         15
   1        NOP    8003   1F         2OK  START                         16
   S0000    RAA  1         E  011   READ                                17
   1        LOD  S0007     E  016                                       18
   S0007    RAA  1         E  011   I1    Z                             19
   T0007    LOD  ABC00             1                                    20
            STD  I0001     S0001                                        21
   S0001    RAA  1         E  011   I3    Z                             22
   T0001    LOD  ABC00             1                                    23
            STD  I0003     S0006                                        24
   S0006    RAA  1         E  011   Y1  Z  OJ                          25
   T0006    LOD  ABC01                                                  26
            STD  Y0001     S0002                 F                     27
   S0002    RAA  1         E  011   I5    Z                             28
   T0002    LOD  ABC00             1                                    29
            STD  I0005     S0003                                        30
   S0003    RAA  1         E  011   Y1  Z  Y1  S                       31
   T0003    RAU  I0005             CN1LI3K  I5                          32
            MPY  ABC02             R  X  YN1LI5                         33
            ALO  I0001             K  I1R                               34
            RAA    8002                                                 35
            LOD  Y0004A                  F                             36
            STD  W0000                                                  37
            RAU  I0003                                                  38
            MPY  ABC02                                                  39
            ALO  I0005                                                  40
            RAA    8002                                                 41
            RAU  C0004A                                                 42
            FMP  W0000                                                  43
            FAD  Y0001                                                  44
            STU  Y0001   1F                                             45
```

```
S0000  RAA  1          E  011    I5     Z     L
1      RAL  I0005                1      R   S
       ALO  ABC00                I5
       STL  I0005    1F
S0000  RAA  1          E  011    GO  TO  0003
1      RSL  I0005                IF    LI6
       ALO  I0006                R   W  I5
       BMI  S0004    S0003
S0004  RAA  1          E  011    PUNCH  Y1  P
T0004  LOD  ABC03                UNCH  I3  PU
       STD  P0000                NCH  I1  F
       LOD  ABC04
       STD  P0001
       LOD  ABC05
       STD  P0002
       RAL  ABC06
       LOD  1F         E  017
S0000  RAA  1          E  011    I3     Z     L
1      RAL  I0003                1      R   S
       ALO  ABC00                I3
       STL  I0003    1F
S0000  RAA  1          E  011    GO  TO  0006
1      RSL  I0003                IF    LI4
       ALO  I0004                R   W  I3
       BMI  1F       S0006
S0000  RAA  1          E  011    I1     Z     L
1      RAL  I0001                1      R   S
       ALO  ABC00                I1
       STL  I0001    1F
S0000  RAA  1          E  011    GO  TO  0001
1      RSL  I0001                IF    LI2
       ALO  I0002                R   W  I1
       BMI  S0008    S0001
S0008  RAA  1          E  011    HALT
T0008  HLT   8003       8778
ABC06    00   0003     0004
ABC05    00   0002     0001
ABC04    00   0001     0003
ABC03    00   0001     0001
ABC02    00   0000     0020
ABC01    00   0000     0000
ABC00    00   0000     0001
```

70 1951 8999+

1	0001	00000000019	0002	00000000030	0003		481	4		932	5			
2	0000	00000000944	1999	00800031997	1997		800031996	1996	6919951790		939		9391995	
3	1995	69094419944	1994	24002019930	0933		9331993	1993	6909441992		1992	24002219910		
4	0938	00093819911	1991	69094519900	1990	24003119890		934		9341989	1989	6909441988		
5	1988	24002419870	0935	00093519870	1987	60002419861		1986	1909461985		1985	15002019840		
6	1984	35000419830	1983	15198280021	1982	69003419811		1981	2418771980		1980	60002219790		
7	1979	19094619780	1978	15002419770	1977	35000419761		1976	1519758002		1975	65048519740		
8	1974	69187719730	1973	24000019720	1972	69197117580		1971	6500311970		1970	69196918080		
9	1969	20003119680	1968	65002419670	1967	15094419660		1966	2000241965		1965	66002419640		
10	1964	15002519630	1963	46196209350	0936	9361962		1962	6909471961		1961	24185019600		
11	1960	69094819590	1959	24185119580	1958	69094919570		1957	2418521956		1956	65095019550		
12	1955	69195418400	1954	65002219530	1953	15094419520		1952	2000221951		1951	66002219500		
13	1950	15002319110	1911	46192209380	1922	65002019330		1933	1509441944		1944	20002219050		
14	1905	66002019160	1916	15002119270	1927	46193809330		940		9401938	1938	1800387780		
15	0950	00000300040	0949	00000200010	0948		10003	947		10001	946		20	
16	0945	00000000000	0944	00000000010	0944		1	944		1	944		1	

70 1951 8998+

1	0001	00000000019	0002	00000000030	0003		481	4		932	5		943	
2	0000	00000009540	1999	00800319970	1997		800319960	1996	6909391790		939	6909391891		
3	0950	69095419950	1995	24002009330	0933	69093318910		944	6909541994		1994	24002209380		
4	0938	69093818910	0949	69095519930	1993	24003109340		934	6909341891		945	69095419920		
5	1992	24002409350	0935	69093518910	0946	60002419910		1991	1909561990		1990	15002019890		
6	1989	35000419880	1988	15198780021	1987	69003419861		1986	2418771985		1985	60002219840		
7	1984	19095619830	1983	15002419820	1982	35000419810		1981	1519808002		1980	65048519790		
8	1979	69187719780	1978	24000019770	1977	69197617580		1976	6500311975		1975	69197418080		
9	1974	20003119730	1973	65002419720	1972	15095419710		1971	2000241970		1970	66002419690		
10	1969	15002519680	1968	46093609350	0936	69093618910		947	6909571967		1967	24185019660		
11	1966	69095819650	1965	24185119640	1964	69095919630		1963	2418521962		1962	65096019610		
12	1961	69196018400	1960	65002219590	1959	15095419580		1958	2000221957		1957	66002219560		
13	1956	15002319550	1955	46195409380	1954	65002019530		1953	1509541952		1952	20002019510		
14	1951	66002019500	1950	15002119110	1911	46094009330		940	6909401891		951	1800387780		
15	0960	00000300040	0959	00000200010	0958		10003	957		10001	956		20	
16	0955	00000000000	0954	00000000010	0954		1	954		1	954		1	

70 1951 8899+

1	0001	00000009000	0002	00000000020	0003		471	4		922	5			
2	0000	00000009340	1999	00800319970	1997		800319960	1996	6919951790		929		9291995	
3	1995	69093419940	1994	24900119930	0923		9231993	1993	6909341992		1992	24900319910		
4	0928	00092819910	1991	69093519900	1990	24002119890		924		9241989	1989	6909341988		
5	1988	24900519870	0925	00092519870	1987	60900519861		1986	1909361985		1985	15900119840		
6	1984	80800219830	1983	69202419820	1982	24187719810		1981	6090031980		1980	19093619790		
7	1979	15900519780	1978	80800219770	1977	60247519760		1976	3918771975		1975	32002119740		
8	1974	21002119730	1973	65900519720	1972	15093419710		1971	2090051970		1970	66900519690		
9	1969	15900619680	1968	46196709250	0926	9261967		1967	6909371966		1966	24185019650		
10	1965	69093819640	1964	24185119630	1963	69093919620		1962	2418521961		1961	65094019600		
11	1960	69195918400	1959	65900319580	1958	15093419570		1957	2090031956		1956	66900319550		
12	1955	15900419540	1954	46195309280	1953	65900119520		1952	1509341951		1951	20900119500		
13	1950	66900119110	1911	15900219220	1922	46193309230		930		9301933	1933	1800387780		
14	0940	00000300040	0939	00000200010	0938		10003	937		10001	936		20	
15	0935	00000000000	0934	00000000010	0934		1	934		1	934		1	

70 1951 8888+

1	0001	00000009000	0002	00000000020	0003		471	4		922	5		933	
2	0000	00000009440	1999	00800315740	1574		800315730	1573	6909291790		929	8009291891		
3	0940	69094415720	1572	24900109230	0923	80092318910		934	6909441571		1571	24900309280		
4	0928	80092818910	0939	69094515700	1570	24002109240		924	8009241891		935	69094415690		
5	1569	24900509250	0925	80092518910	0936	60900515680		1568	1909461567		1567	15900115660		
6	1566	80800215650	1565	69202415640	1564	24187715630		1563	6090031562		1562	19094615610		
7	1561	15900515600	1560	80800215590	1559	60247515580		1558	3918771557		1557	32002115560		
8	1556	21002115550	1555	65900515540	1554	15094415530		1553	2090051552		1552	66900515510		
9	1551	15900615500	1550	46092609250	0926	80092618910		937	6909471511		1511	24185015220		
10	1522	69094815330	1533	24185115440	1544	69094915050		1505	2418521516		1516	65095015270		
11	1527	69153818400	1538	65900315490	1549	15094415100		1510	2090031521		1521	66900315320		
12	1532	15900415430	1543	46150409280	1504	65900115150		1515	1509441526		1526	20900115370		
13	1537	66900115480	1548	15900215090	1509	46093009230		930	8009301891		941	1800387780		
14	0950	00000300040	0949	00000200010	0948		10003	947		10001	946		20	
15	0945	00000000000	0944	00000000010	0944		1	944		1	944		1	

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

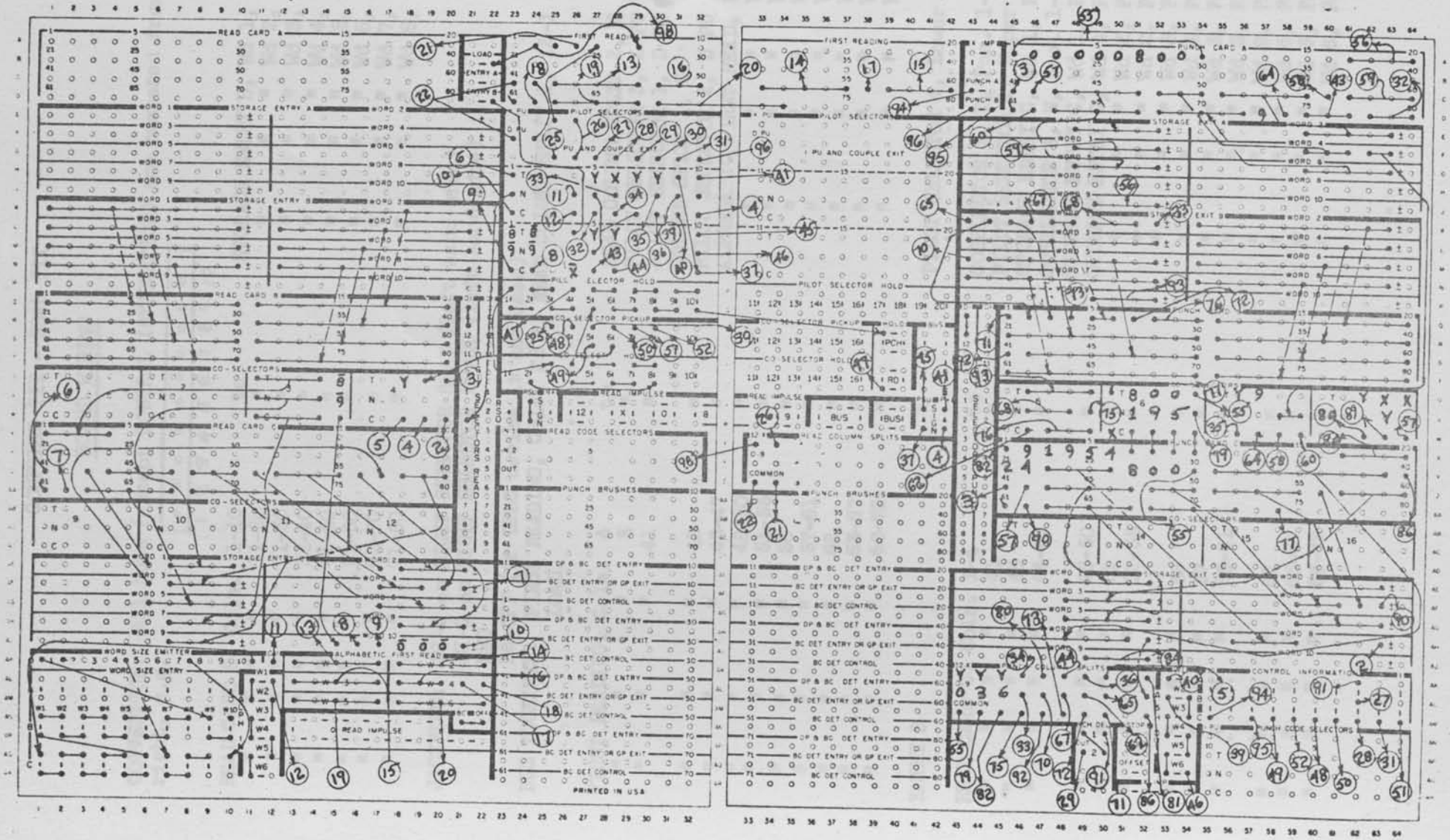
READ-PUNCH UNIT, TYPE 533 CONTROL PANEL

(USED WITH TYPE 650 MAGNETIC DRUM DATA-PROCESSING MACHINE)

Case Institute of Technology
RUNCIBLE I, SOAP III
4-22-58

Form No. 22-6207-1
Printed in U.S.A.

Note: Bar(–) over digit means "timed to read feed"

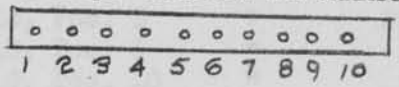


Remove wire from AN 49 to AP 50

COMPILER III - SOAP III

BOARD WIRING

CONVENTION: All hubs (except CONTROL INFORMATION) are numbered from left to right, e.g.



READ CARD B,	Cols.	1-10	to	STORAGE ENTRY B,	Wd 1,	Pos. 1-10
" " "	"	11-20	"	" " "	Wd 2,	" 1-10
" " "	"	21-30	"	" " "	Wd 3,	" 1-10
" " "	"	31-40	"	" " "	Wd 4,	" 1-10
" " "	"	41-50	"	" " "	Wd 5,	" 1-10
" " "	"	51-60	"	" " "	Wd 6,	" 1-10
" " "	"	61-70	"	" " "	Wd 7,	" 1-10
" " "	"	71-80	"	" " "	Wd 8,	" 1-10

JACKPLUG: DI-C on DIGIT SELECTOR READ (left hand side)
DI-C on DIGIT SELECTOR PUNCH (left hand side)
RSU
BC-OFF (if DPBC device is not used)

BUS TOGETHER: FIRST READING, Cols. 2 & 7, READ COLUMN SPLIT 1, 12-X and wire to LOAD.

FIRST READING,	Cols.	4	to	READ COLUMN SPLIT 2, C		
" "	"	5	"	PILOT SELECTOR 3, X PU		
" "	"	41	"	PILOT SELECTOR 1, D PU & READ		
				COLUMN SPLIT 1, C		
" "	"	42	"	PILOT SELECTOR 2, D PU		
" "	"	43-47	"	ALPHABETIC FIRST READ,	Wd 1	
" "	"	48-50	"	" " "	Wd 4,	Pos. 1-3
" "	"	51-55	"	" " "	Wd 2	
" "	"	56	"	" " "	Wd 4,	Pos. 4
" "	"	57-61	"	" " "	Wd 3	
" "	"	62	"	" " "	Wd 4,	Pos. 5
" "	"	63-67	"	" " "	Wd 5	
" "	"	68-72	"	" " "	Wd 6	
ENTRY B			"	PILOT SELECTOR 4, Lower T		
READ CARD C,	Cols.	1-4	to	COSELECTOR 3, T, Pos. 1-4		
" " "	"	41	"	PILOT SELECTOR 1, Upper T		
" " "	"	43	"	STORAGE ENTRY C,	Wd 1,	Pos. 6
" " "	"	44-47	"	COSELECTOR 3, N, Pos. 1-4 &		
" " "	"	48-50	"	STORAGE ENTRY C,	Wd 1,	Pos. 7-10
" " "	"	51	"	STORAGE ENTRY C,	Wd 4,	Pos. 6-8
" " "	"	52-55	"	" " "	Wd 2,	Pos. 6
" " "	"	56	"	" " "	Wd 2,	Pos. 7-10 &
" " "	"	57	"	" " "	Wd 8,	Pos. 7-10
" " "	"	58-61	"	" " "	Wd 4,	Pos. 9
				" " "	Wd 3,	Pos. 6
" " "	"	62	"	" " "	Wd 3,	Pos. 7-10 &
" " "	"	63-67	"	" " "	Wd 9,	Pos. 7-10
" " "	"	68-72	"	" " "	Wd 4,	Pos. 10
				" " "	Wd 5,	Pos. 6-10
				" " "	Wd 6,	Pos. 6-10

STORAGE ENTRY C, Wd 10, Pos. 4	to	COSELECTOR 3, C, Pos. 5
" " " Wd 10, Pos. 5	"	PILOT SELECTOR 2, Lower C
" " " Wd 10, Pos. 6	"	" " 1, Lower C
" " " Wd 10, Pos. 10	"	" " 1, Upper C
" " " Wd 7, Pos. 7-10	"	COSELECTOR 3, C, Pos. 1-4

CAI to PILOT SELECTOR 4, Upper N
ALPHA IN, W1 to PILOT SELECTOR 4, Upper C

JACKPLUG: On ALPH IN, W1-W2; W2-W3; W3-W4; W4-W5; W5-W6

WORD SIZE 10 for B, Wds 1-8 C, Wds 1-6
" " 0 for B, Wd 9 & Wd 10
" " 4 for C, Wds 7-9
" " 7 for C, Wd 10

COSELECTOR 4, T, Pos. 4	to	COSELECTOR PICKUP 6
" " " " 5	"	PUNCH CARD A, Cols. 41
		PUNCH CARD C, Cols. 41
" " C " 3	"	PUNCH COLUMN SPLIT 8, 12-X
" " C " 4	"	P+ & PILOT SELECTOR 10, Upper C
" " C " 5	"	STORAGE EXIT C, Wd 8, Pos. 10
PILOT SELECTOR 3, I PU	"	COSELECTOR PICKUP 3
" " 4, I PU & X PU	"	READ COLUMN SPLIT 2, 12-X
" " 5, I PU	"	CONTROL INFORMATION 3(numbered from Rt to
" " 6, I PU	"	CONTROL INFORMATION 3 Lt.)
" " 7, I PU	"	PUNCH DELAY 2, OUT
" " 8, I PU	"	COSELECTOR PICKUP 5
" " 9, I PU	"	CONTROL INFORMATION 2
" " 10, I PU	"	PUNCH B

Emit READ DIGIT 2 to PILOT SELECTOR 1, Upper N

PILOT SELECTOR 5, Upper N	to	PILOT SELECTOR 5, Lower N &
		" " 7, Upper C
" " " Upper C	"	PUNCH CARD A, Cols. 78-80
" " 6, Upper N	"	PUNCH COLUMN SPLIT 4, 0-9 &
		STORAGE EXIT B, Wd 1, Pos. 10
" " " Upper C	"	PUNCH COLUMN SPLIT 4, 12-X
" " 8, Upper N	"	COSELECTOR 7, C, Pos. 2
" " " Upper C	"	PUNCH COLUMN SPLIT 7, 12-X
" " 9 Upper T	"	PILOT SELECTOR 10, Lower C & PSU
" " " Upper N	"	ALPH OUT, Wd 2
" " " Upper C	"	ALPH OUT, Wd 1
" " 10, Upper T	"	P+
" " 5, Lower C	"	PUNCH CARD A, Cols. 75-77
" " 6, Lower C	"	PUNCH COLUMN SPLIT 6, 12-X
" " 10, Lower T	"	PSU
" " " Lower N	"	ALPH OUT, Wd 6

COSELECTOR PICKUP 4	to	CONTROL INFORMATION 5
" " 5	"	" " 7
" " 6	"	" " 4
" " 7	"	" " 1
" " 8	"	" " 6
PILOT SELECTOR HOLD 1	"	COSELECTOR HOLD 3
" " " 5	"	" " 4

READ HOLD TO PILOT SELECTOR HOLD 4
PUNCH HOLD TO PILOT " " 10

JACKPLUG: PILOT SELECTOR HOLD, 1-2; 2-3; 3-4; 5-6; 6-7; 7-8; 8-9; 9-10;
COSELECTOR HOLD, 4-5; 5-6; 6-7; 7-8

COSELECTOR HOLD 2 to COSELECTOR HOLD 8

PUNCH CARD A, Cols. 1 & 10	to	COSELECTOR 6, T, Pos. 5
" " " Cols. 17-20	"	STORAGE EXIT A, Wd 9, Pos. 7-10
42	"	COSELECTOR 8, C, Pos. 5 #
" " " 43-47	"	PUNCH CARD C, Cols. 42
" " " 48-50	"	STORAGE EXIT A, Wd 1, Pos. 6-10
" " " 51-55	"	" " " Wd 4, Pos. 6-8
" " " 56	"	" " " Wd 2, Pos. 6-10
" " " 57-61	"	COSELECTOR 7, C, Pos. 4
" " " 62	"	STORAGE EXIT A, Wd 3, Pos. 6-10
" " " 63-67	"	COSELECTOR 7, C, Pos. 5
" " " 68-72	"	STORAGE EXIT A, Wd 5, Pos. 6-10
" " " 74	"	" " Wd 6, Pos. 6-10
		COSELECTOR 7, C, Pos. 3
STORAGE EXIT A, Wd 4, Pos. 9	"	COSELECTOR 7, N, Pos. 4
" " Wd 4, Pos. 10	"	" " N, Pos. 5
PUNCH CARD B, Cols. 1	"	PUNCH COLUMN SPLIT 8, C
" " " 2	"	STORAGE EXIT B, Wd. 1, Pos. 2
" " " 3	"	" " " Wd 1, Pos. 3
" " " 4	"	PUNCH COLUMN SPLIT 6, C
" " " 5-6	"	STORAGE EXIT B, Wd 1, Pos. 5-6
" " " 7	"	PUNCH COLUMN SPLIT 7, C
" " " 8-10	"	COSELECTOR 5, C, Pos. 3-5
" " " 41	"	PUNCH COLUMN SPLIT 5, C
" " " 11-20	"	STORAGE EXIT B, Wd 2, Pos. 1-10
" " " 21-30	"	" " " Wd 3, Pos. 1-10
" " " 31-40	"	" " " Wd 4, Pos. 1-10
" " " 42-50	"	" " " Wd 5, Pos. 2-10
" " " 51-60	"	" " " Wd 6, Pos. 1-10
" " " 61-70	"	" " " Wd 7, Pos. 1-10
" " " 71-80	"	" " " Wd 8, Pos. 1-10
STORAGE EXIT B, Wd 1, Pos. 1	"	PUNCH COLUMN SPLIT 8, 0-9
" " " Wd 1, Pos. 4	"	" " " 6, 0-9
" " " Wd 1, Pos. 7-9	"	COSELECTOR 5, N, Pos. 2-4
" " " Wd 5, Pos. 1	"	PUNCH COLUMN SPLIT 5, 0-9
" " " Wd 9, Pos. 7-10	"	COSELECTOR 5, T, Pos. 2-5

COSELECTOR 5, N, Pos. 5 to
" " C, Pos. 2 "
" 6, T, Pos. 1 "
" " T, Pos. 5 "
" " N, Pos. 5 "
" " C, Pos. 2-5 "
" 8, C, Pos. 3 "
" " C, Pos. 4 "

PUNCH CARD C, Cols. 1 "
" " " " 17-20 "
" " " " 23-26 "
" " " " 31-39 "
" " " " 40 "
" " " " 43-47 "
" " " " 48-50 "
" " " " 51-55 "
" " " " 56 "
" " " " 57-61 "
" " " " 62 "
" " " " 63-67 "
" " " " 68-72 "
STORAGE EXIT C, Wd 7, Pos 10 "

PUNCH COLUMN SPLIT 4, C
" " " 7, 0-9
PUNCH CARD C, Cols. 74
PUNCH COLUMN SPLIT 1, C
" " " 2, C
PUNCH CARD C, Cols. 7-10
PUNCH COLUMN SPLIT 5, 12-X
" " " 10, 12-X

PUNCH COLUMN SPLIT 3, C
STORAGE EXIT C, Wd 9, Pos. 7-10
" " " Wd 8, Pos. 3-6
" " " Wd 7, Pos. 1-9
PUNCH COLUMN SPLIT 10, C
STORAGE EXIT C, Wd 1, Pos. 6-10
" " " Wd 4, Pos. 6-8
" " " Wd 2, Pos. 6-10
" " " Wd 4, Pos. 9
" " " Wd 3, Pos. 6-10
" " " Wd 4, Pos. 10
" " " Wd 5, Pos. 6-10
" " " Wd 6, Pos. 6-10
PUNCH COL. SPLIT 10, 0-9

PUNCH DELAY 1, IN to CONTROL INFORMATION 3
PUNCH A to CONTROL INFORMATION 10
PUNCH B to CONTROL INFORMATION 8

JACKPLUG: ALPH OUT, W2-W3; W3-W4; W4-W5; W5-W6

EMIT READ-TIMED DIGITS:

8 in COSELECTOR 3, T, Pos. 5
8 in PILOT SELECTOR 1 ¢ 2, Lower T
9 in COSELECTOR 3, N, Pos. 5
9 in PILOT SELECTOR 1 ¢ 2, Lower N
X in PILOT SELECTOR 4, Lower C
0 in STORAGE ENTRY C, Wd 10, Pos. 7-9

EMIT PUNCH-TIMED DIGITS:

Y in COSELECTOR 4, T, Pos. 3
Y in PILOT SELECTOR 5, 7 ¢ 8, Upper T
Y in " " 5¢6, Lower T
Y in COSELECTOR 7, T, Pos. 2
Y in " 8, T, Pos. 3
Y in " " N, Pos. 4
Y in PUNCH COLUMN SPLIT 1, 2¢3, 12-X
X in PILOT SELECTOR 6, Upper T
X in COSELECTOR 8, T, Pos. 4¢5
X in " 6, C, Pos. 1
9 in PUNCH CARD A, Cols. 73
9 in COSELECTOR 7, T, Pos. 3

In PUNCH CARD A, Cols. 2-9 emit the following punch-timed digits--00000800
" PUNCH CARD C, " 2-6 " " " " " " --91954
" " " " " 21-22 " " " " " " --24
" " " " " 27-29 " " " " " " --800

In COSELECTOR 6, T, Pos. 2-4 emit the following punch-timed digits--800
" " " N, Pos. 2-4 " " " " " " --195
" PUNCH COLUMN SPLIT 1, 2 & 3, (0-9) " " " " --036

INDEX