

LJED TABSOL
for GE-225 CARD SYSTEM

David T. Schmidt
Adv. Mfg. Eng. Services
and
Paul Margaritis
Large Jet Engine Department

NOVEMBER 1961

GENERAL ELECTRIC
COMPUTER DEPARTMENT
PHOENIX, ARIZONA

PRINTED IN U.S.A.

TABLE OF CONTENTS

	Page
INTRODUCTION	1
I CONSTANTS AND VARIABLES	3
A. Variables (Parameters)	3
B. Constants	4
II THE DICTIONARY	5
III TABLE ENTRIES	6
A. Stub Entries	8
B. Table Row Entries	11
IV STRUCTURE TABLE FORM	15
V OPERATION CODES	20
A. Test Operation Codes	20
B. Result (or Action) Operation Codes	21
1. Arithmetic Operation Codes	21
2. Shifts	23
3. Indexing	23
4. The AND Instruction	28
5. The Convert Instruction (CON)	28
6. Transfer Instructions	28
7. Output Instructions	30
8. The STOP Instruction	30
VI PROBLEM INPUT DATA FORMAT	31
VII OUTPUT FORMAT	32
VIII ERROR IDENTIFICATION	33
IX SETTING UP A TABSOL SYSTEM	33

TABLE OF CONTENTS (CONT.)

	Page	
X	KEYPUNCHING	34
	A. Dictionary	34
	B. Table Forms	34
	1. Header (Descriptor) Card	34
	2. Noise Cards	35
	3. Table Cards	36
	C. Problem Input	36
XI	MACHINE REQUIREMENTS	36
XII	TABCON TABLE TRANSLATOR PROGRAM	37
	A. The Three Phases	37
	B. LJED-225 TABCON Error Print-outs	37
XIII	TABSOL TABLE SOLVING PROGRAM	39
	A. TABSOL Executive Program	39
	B. TABSOL Subroutines	40
	C. TABSOL Error Printouts	40
XIV	TABCON AND TABSOL RUNNING INSTRUCTIONS	41
	A. Deck Set-Up for TABCON	41
	B. Running TABCON	41
	C. Deck Set-Up for TABSOL	41
	D. Running TABSOL	41
	E. TABCON and TABSOL Messages	42

LIST OF FIGURES

Figure No.		Page
1.	Example of Table Entry	6
2.	Example of Table Row	7
3.	Example of Table Column	7
4.	Examples of Stub Entries	9
5.	Corresponding Examples of the Conditions in Figure 4	10
6.	Examples of Table Row Entries	13
7.	Corresponding Examples of the Condition in Figure 6	14
8A.	Sample Structure Table Form	16
8B.	Sample Structure Table Form	17
9.	Example of Indexing Operation Code	25

INTRODUCTION

There has been a continued and increasing interest in TABSOL (Tabular Systems Oriented Language) by functional specialists, systems people, and Computer Department personnel as they apply and extend the decision structure table to more and more problems within the General Electric Company. A parallel development has been the interest in structure tables by the Conference on Data Systems Languages (CODASYL) whose membership includes representatives from all computer manufacturers and many of the leading computer users. As a result of this interest outside the General Electric Company there is strong reason to believe that the systems group of CODASYL will recommend that decision structure tables be the next systems oriented language replacing COBOL.

This report describes a TABSOL processor for the GE-225 Computer developed by the Data Processing Techniques Unit, Large Jet Engine Department (LJED), Evendale, Ohio. Early in 1962 LJED plans to implement an extensive body of structure tables covering complete manufacturing planning for complex gears used in high-speed transmissions for aircraft jet engines. Original feasibility work on the project was done late in 1960 using LJED's existing computer. As work on the project continued, the department made plans to replace the existing computer with a GE 225. The GE 225 was installed in October 1961.

When the decision was made to install the GE 225, it was realized that the GECOM-TABSOL Compiler being written for the GE 225 by the Computer Department would not be available in time for the gear project. In order not to delay the implementation of the project results, the Data Processing Techniques Unit, working jointly with various Services' components and the Computer Department, determined to develop a structure table format and table processor that would be upward compatible with the GECOM-TABSOL program scheduled for release early in 1962. As a result of this decision, LJED successfully ran their structure table processor in August 1961, thus allowing them to continue with their plans for initial implementation of automatic gear planning by January 1962.

The history of TABSOL and the decision structure table, following their conceptual development by the ISP Team in 1958, is extensively covered by TIS Reports and Computer Department publications. It is strongly recommended that the reader, prior to studying this report, gain a better understanding of developments, both technical and historical, in the TABSOL field by reviewing some of the following literature:

TABSOL - The Fundamentals of a Systems
Oriented Language (TIS Report R59MS302)

702 TABCON - A Table Conversion Procedure
(TIS Report R59MS303)

305 TABSOL - A Structure Table Processor for
the 305 RAMAC (TIS Report R59MS310)

305 TABCON - A Structure Table Converter for
the 305 RAMAC (TIS Report R60MS309)

650 TABCON-TABSOL (TIS Report R60MS312)

LOGTAB - A Logic Table Technique (DF-59LS23)

Preliminary TABSOL Manual (CPB-147)
Published by the Computer Department

In addition, application examples in which the decision structure table was employed to document the functional logic are described in the following publications:

Automatic Planning for the Turning Process
(TIS Report R61-X-8)
Published by the X-Ray Department

Medium AC Motor and Generator Department -
Rotor Assembly Planning (TIS Report R60MS401)

TABSOL Application Manual (CPB-147A)
Published by the Computer Department

Much of the work on the LJED-225 TABSOL program was based on previous TABSOL development: primarily the 650 TABSOL program and the GECOM-TABSOL language specification developed by F. D. McNeill, Defense Systems Department, and D. Klick, Computer Department. Messrs. L. Haines and E. Criddle, Data Processing Techniques Unit, Large Jet Engine Department, had primary responsibility for writing the generalized table solving program. Original specifications for the LJED program were developed with the aid of D. F. Langenwalter, Engineering Service, E. F. LaChance, Quality Control Service, T. F. Kavanagh, Production Control Service, and H. W. Nidenberg and J. R. Zinchak, Advanced Manufacturing Engineering Service.

I CONSTANTS AND VARIABLES

A. Variables (Parameters)

A "variable" or "parameter" is a named field containing a value. The value contained is known as the contents of the variable field or the contents of the parameter.

Space in memory is reserved for 499 variables, numbered 001, 002, ----- 499. All input variables (problem input), intermediate variables (stored or created data used within the program and not as output), and output variables must be stored and defined in this space. Special assignment is given to portions of the variable area as follows:

Parameter Number	
001 - 484	Unrestricted variable area
485	Variable clear
486-499	Output area (Variables #498 and #499 contain the problem number from the input data unless destroyed by the coder.)

The variable clear (parameter 485) allows for the holding (non-zeroing) of parameters between problems. A more detailed explanation of the variable clear is contained in Section II, The Dictionary.

All variables are referred to in the structure tables by their name rather than number. Usually the variable names are made mnemonic for ease of interpretation and understanding. Variable names are limited to a maximum of six characters -- each character may be a numeric (0-9), an alphabetic letter (A-Z), or a blank. Each variable name must include at least one alphabetic character. Examples of variable names are given below:

LENGTH
WIDTH
WØRD01
WØRD 2

Variable names must be used consistently throughout the data. If a name does not occupy the full six positions, justification to the right or left must be established and used. Special characters other than blanks and hyphens cannot be used in variable names.

Each variable is stored in two GE-225 memory locations. Numeric variables that will be used for arithmetic calculations must be stored in the binary mode: these variables have a capacity of ten decimal digits internally but are limited to six digits on input. The input data is stored as the six low order positions of the internal ten digit field. All ten digits of the variable are available as output by using the Convert (CON) command described later.

Numeric variables that are not used in arithmetic calculations (testing is not an arithmetic calculation) are stored in the binary coded decimal (BCD) mode and are limited to six characters on input, internally, and on output. Thus, numeric input variables are limited to values from 000000 to 999999. Numbers that could occupy less than the allowed six digits must contain leading zeroes to complete the data field.

Alphabetic variables presumably are never used for arithmetic calculations and hence are stored in the BCD mode. Alphabetic variables are also limited to six characters on input, internally, and on output.

The LJED-225 TABSOL program operates in fixed-point arithmetic. Thus, the table writer must establish and maintain control of the decimal point location (referred to as decimal notation) of each variable at all times.

B. Constants

Constants, as opposed to variables, are fixed values written directly in the structure tables. They may be either numeric, alphabetic, or a combination alpha-numeric. Alphabetic constants and alpha-numeric constants are treated identically.

Numeric constants are six characters in length. Constants normally occupying less than the maximum six digits must have leading zeroes printed, and hence can range in positive values from 000000 through 999999.

Alphabetic or alpha-numeric constants (hereafter referred to as alpha constants) also contain a maximum of six characters. It is not necessary to fill in leading characters or zeroes on alpha constants. All alpha constants, however, must be enclosed in quotation marks on the table form.

One additional comment. The table writer must organize and write tables based on a knowledge of what variables will be used for arithmetic calculations -- and thus must be stored in the binary mode -- and those that will be used only for testing or output and therefore can remain in the BCD mode. Tables should not be written that mix tests and/or arithmetic operations of binary values and alpha-numeric values.

II THE DICTIONARY

A dictionary of all the variable names used in any set of tables is written to define the variables and to assign parameter numbers to them. The assignment of parameter numbers to the variables is arbitrary except for the special assignment of the areas mentioned previously.

Numbers 001-484 are relatively unrestricted, 485 is the variable clear, and 486-499 is the output area. Numbers 498 and 499 contain the problem number from input data. Parameter numbers 001-484 can be used as desired to define input or intermediate variables. In this area the assignment of parameter numbers is not critical unless using indexing, where it is desirable to set up an array or list of variables so that the "ith" variable can be referenced from a known parameter, "i" itself being a variable.

The variable clear parameter (#485) is used for controlling the number of variables to be cleared (set to zero) by the zeroing routine as part of the executive program. The zeroing of parameters by the executive program is normally done between problems. Parameter 485 will contain the parameter number of the last parameter to be cleared (starting with parameter 001) by the initializing routine. The output area (parameter 485-499) will always be cleared. If parameter 485 is left blank (zero), 50 parameters (parameter numbers 435-484) will automatically be treated as the hold area for the problem and parameters 1-434 and 485-499 will be cleared.

When writing the dictionary, it is desirable to group all parameters that require holding and assign parameter numbers 435-484 as they automatically will be held. If more than 50 parameters need holding it is suggested that they be numbered in descending sequence starting with 434, 433, 432, etc. This allows simplicity in establishing the value for the variable clear parameter, normally less than 485.

As previously stated, if the variable clear parameter (485) is left blank (zeroes), then 50 parameters (435-484) will be held. Because the variable clear parameter contains the number of the last parameter to be cleared, and since a zero in 485 indicates hold 50 parameters, it is not possible to hold every parameter. By specifying 001 in parameter 485 all parameters (002-484) except parameter 001 will be held. This is the maximum number of parameters that can be held.

Note that the variable clear parameter is cleared by the initializing routine; thus, if it is being used it must be input with each problem. Note also that the variable clear parameter can vary from problem to problem, if required, by changing the contents of the parameter through new input.

Parameters 486-499 are defined as the output parameters. Any data desired to be output must be stored in the parameters 486-497, usually given a name such as OUTWD1, OUTWD2, OUTWD12, and then "punched" into columns 1-72 (12 words of 6 columns each). Parameters 498 and 499 are normally reserved for the problem number identification and/or problem error identification.

The dictionary format requires the parameter number to be written and punched in columns 9-11, the parameter name in columns 13-18, and a "D" in column 80. The remaining columns are for notes, explanation, or definition of parameters. It is often desirable to indicate the decimal point location of each parameter in this area.

III TABLE ENTRIES

Each table entry is composed of a stub entry, an operation code, and an operand (as shown in figure 1). One stub entry may apply to more than one operation code and operand for multi-row tables.

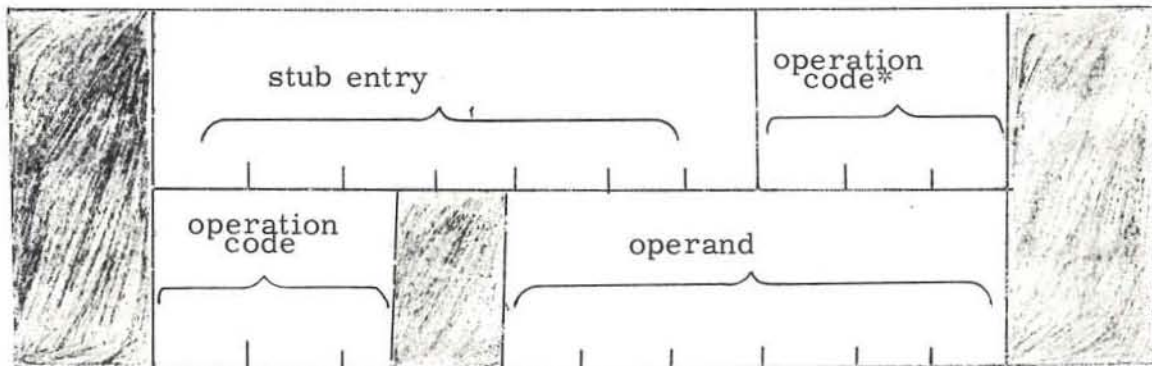


Figure 1. Example of Table Entry

*The operation code may be included in either of the locations indicated, as explained later.

A table row is comprised of many table entries (table blocks) in succession horizontally each having a separate stub entry as shown in figure 2.



Figure 2. Example of Table Row

A table column is defined as one stub entry and one or more operands beneath the stub entry, all referencing the stub entry, as shown in figure 3.

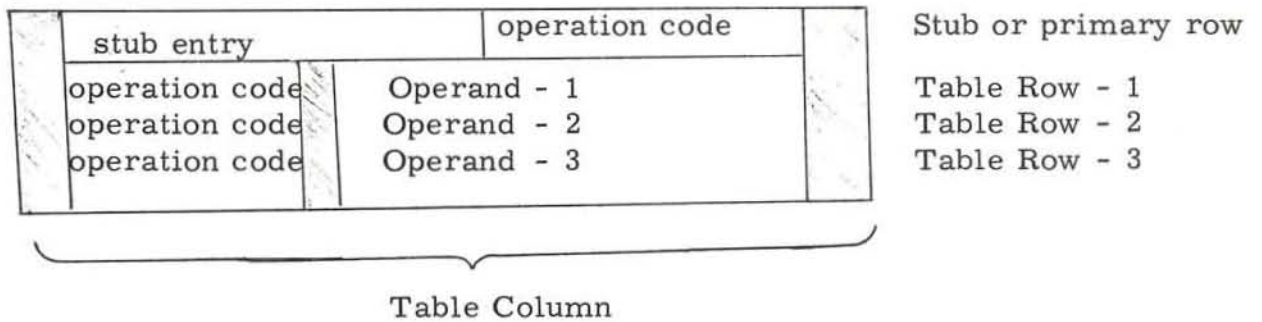


Figure 3. Example of Table Column

A. Stub Entries

The stub entry is always one of the following:

1. A parameter name
2. A parameter name plus operation code
3. A dash (~)
4. "GO TO"
- or 5. "PERFORM"

The parameter name as the stub entry is the general usage. Usually stub parameters are being tested against other values (conditions) or the stub parameters are having results (actions) performed on them such as values stored in the stub parameter, added to the parameter, etc.

The use of the dash (or tilde) in the first position of the stub entry is limited to supplementary indexing rows or to programs where a punch (PCH), punch and clear (PCC), transfer (TRA), end or stop action operation is used.

The words "GO TO" are used as the stub entry where it is desired to transfer control of program to another table. The words "GO TO" in the stub entry must be followed by a "TABLE XXXX" in the table row (Figure 5, example 8).

The word "PERFORM" is used as the stub entry where it is desired to only temporarily transfer control of the program to another table or series of tables. This also must always be followed by a "TABLE XXXX" in the table row. The end of the logic being "PERFORMED" is indicated by the operation code "END". Upon encountering an "END", the program will transfer control to the next result action in the same row of the table that contained the last executed "PERFORM". If no result actions exist, the program then executes the universal transfer. Examples of possible stub entries are shown in figures 4 and 5.







	1	2	3	4	5	6	7	8	9	10	11	12
(1)	6 Digit Parameter Name						Blank	3 Digit Operation Code			Blank	
(2)	6 Digit Parameter Name						Blank	2 Digit Operation Code		Blank		
(3)	6 Digit Parameter Name						Blank					
(4)	~	(Blank For Indexing and Punching)										
(5)	P	E	R	F	O	R	M	Blank				
(6)	G	O	T O		Blank							
	1	2	3	4	5	6	7	8	9	10	11	12

Figure 4. Examples of Stub Entries.

	1	2	3	4	5	6	7	8	9	10	11	12
(1)	T	E	S	T		1		N	G	R	[REDACTED]	☐
(2)	T	E	S	T		1		G	R			☐
(3)	T	E	S	T		1						☐
(4)	~											☐
(5)	P	E	R	F	O	R	M					☐
(6)	G	O		T	O							☐
	1	2	3	4	5	6	7	8	9	10	11	12

Figure 5. Corresponding Examples of the Conditions in Figure 4

B. Table Row Entries

As shown in figure 1, the operation code may exist in one of two positions: either in the stub row or the table row.

- If the operation code is identical for all of the table rows, then it may be written in the stub row operation code location instead of the table row location. By so doing, the amount of table writing required is minimized.
- If the operation code is included on the stub row, that operation code (whether result or action) will be applied to all succeeding table rows and hence there should be no operation code included in the table rows. An operation code in a table row will supersede any operation codes in the stub row.
- If, however, the operation codes are different for various rows in the table, and hence cannot be written in the stub row they must be written in the table row operation code location for each row even if the same code is repeated in several rows. As long as there is any exception to the operation code, it must be written in each table row.

In either location a 2-digit operation code must be left justified in the 3 position field -- leaving the blank digit on the right.

The permissible operation codes and an explanation of their function is contained in Section V, Operation Code.

The operand can have one of three forms:

1. alpha-numeric constants (BCD)
2. numeric constants (binary)
3. parameter names

Alpha-numeric constants, as stated previously, are alpha, numeric, or a combination alpha-numeric data that will not be used in arithmetic calculations and can therefore be stored in the decimal mode. Alpha-numeric constants must be identified by quotation marks around the operand in the shaded-format columns. Literal data need not have leading zeroes filled in; unused columns may be left blank. Literal data is stored in the BCD mode.

Numeric data to be converted to the binary mode does not require quotation marks around the operand but must have unused columns filled with zeroes.

Parameter names (1) are not enclosed in quotation marks, (2) must contain at least 1 alphabetic character, and (3) need not have leading zeroes filled in.

It is very important to describe and distinguish between numeric constants, alpha-numeric constants, and parameter names. Any errors will definitely be indicated either through the editing features of the LJED-225 TABCON program or through invalid data during a LJED-225 TABSOL run.

Examples of possible table row entries are shown in figures 6 and 7.

The table rows or secondary rows may be as follows:

	1	2	3	4	5	6	7	8	9	10	11	12	
(1)	3 Digit Operation Code			"	Alpha-numeric literal						"	☐	
(2)	3 Digit Operation Code			Blank	Numeric literals or parameter names						Blank	☐	
(3)	2 Digit Operation Code		Blank	"	Alpha-numeric literal						"	☐	
(4)	2 Digit Operation Code		Blank		Numeric literals or parameter names						Blank	☐	
(5)				"	Alpha-numeric literal						"	☐	
(6)					Numeric literals or parameter names						Blank	☐	
(7)	☐			(Blank (For No Operation))									
(8)	T	A	B	L	E	Blank	X	X	X	X	Blank	☐	
	1	2	3	4	5	6	7	8	9	10	11	12	
	4-Digit Table Number												

Figure 6. Examples of Table Row Entries

	1	2	3	4	5	6	7	8	9	10	11	12
(1)	N	E	Q	"				A	B	C	"	☐
(2)	N	E	Q		W	I	D	T	H	S		☐
(2)	N	E	Q		0	0	1	2	5	0		☐
(3)	E	Q		"				A	B	C	"	☐
(4)	E	Q			W	I	D	T	H	S		☐
(4)	E	Q			0	0	1	2	5	0		☐
(5)				"				A	B	C	"	☐
(6)					W	I	D	T	H	S		☐
(6)					0	0	1	2	5	0		☐
(7)	~											☐
(8)	T	A	B	L	E		2	0	0	5		☐
	1	2	3	4	5	6	7	8	9	10	11	12

-14-

Figure 7. Corresponding Examples of the Condition in Figure 6.

IV STRUCTURE TABLE FORM

Once decision logic has been defined through establishment of parameters or constants and their relationships along with the associated results or actions, this logic can be documented in a structure table and/or on the structure table form within the limits of the table entries as described previously. All structure tables, sooner or later, must be written on the Structure Table Form for processing. A sample of the form is shown in figure 8.

The table form provides for program identification, name, and date at the top of the form. This information is not key-punched, it is only for use of the table writer. Directly beneath this is a column count representing the 80 columns on a punched card. The form provides for a maximum of six table columns and a maximum of fourteen table rows plus a single stub row per page. The six 11-character column widths are separated by a special character (\sphericalangle) representing a 12-4-8 punch in the cards.

The table header or descriptor row includes table number, conditions, actions, rows, universal transfer (UTRA) and the error transfer (ETRA). This information is required only once per table and must be entered on the first page of the table.

The table number is a 4 digit, alpha-numeric value in positions 16-19 of the descriptor row. If an all-numeric value is used as the table number, the leading zeroes must be filled in; if the table number is alphabetic or alpha-numeric, unused positions need not be filled in, but the table number must be used consistently within the 4-digit field.

Examples:

Table 0050
Table 2000
Table A500
Table SQRT
Table PAY (Must always be right justified)
Table ABC (Must always be left justified)

Table numbers cannot contain special characters (# , (! ©)

STRUCTURE TABLE FORM - A														PROGRAM	NAME	DATE																																																																																																			
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	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	73	74	75	76	77	78	79	80																																				
Table No.																																CONDITIONS														ACTIONS														ROWS														NOTE														UTRA														E T R A													
Row No.																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			
0																																																																																																																			

9/11/61



GENERAL ELECTRIC

Figure 8A. Sample Structure Table Form

With the allotted four positions for table numbers and figuring for alpha A-Z (except I, Ø, and Z) and numeric 0-9, there are over one million unique combinations for table numbers,

The number of conditions in a table is defined as the number of table columns performing tests, therefore having test operation codes. In the usual quadrant representation of a structure table, the number of conditions is the number of columns to the left of the vertical double line. The number of conditions in the table is written in positions 22 and 23 of the table descriptor row -- preceding the word CONDITIONS. Tables containing no conditions (unconditional tables) must have CONDITIONS entered as 00 and not left blank.

The number of actions is the number of result or action columns in the table (columns to the right of the vertical double line) and is written in positions 36 and 37 of the table descriptor preceding the word ACTIONS. For any table the sum of CONDITIONS plus ACTIONS is limited to 54 (nine pages at six columns per page).

The number of table rows is defined as the number of horizontal rows in the table proper not counting the stub row, the table descriptor row, or any rows containing notes. The number of rows is written in positions 47 and 48 ahead of the word ROWS, and is limited to two digits, and therefore a maximum of 99.

The universal transfer (UTRA) is the next table to go to if the table exit is identical for each solution of the table. If each table row has its own GO TO, then the UTRA must be written as 0000.

The error transfer (ETRA) is the next table to go to if there is "no solution" to the table: "no-solution" indicating failure to pass all of the tests in any one table row. If the "no-solution" possibility does not exist, zeroes may be written in this field. Do not leave it blank. The rules pertaining to table numbering apply to the UTRA and ETRA. When completed, the table descriptor row might read as follows:

TABLE 0001. 02 CONDITIONS 02 ACTIONS 02 ROWS. NOTE
UTRA 0000 ETRA STØP.

Following the descriptor row of the table are seven additional rows identified only by a "0" in position 7. These seven rows are left for the table writer to include notes, expressions, statements or any other information desired to include with the table but not in the table. The information written in these rows will be ignored by the computer during processing. There is no limit on any of the data included in this area.

The rows in which this information is written are known as "noise-rows"; the cards that it is punched into "noise-cards". More than the allotted seven noise cards preceding the table body may be included but must be done so on another table form. Five "noise rows" are also included after the table body for notes or other information. Again, additional noise cards after the table may also be included on additional forms.

The table number must be repeated in positions 1-4 of each row including the header row and each used noise or table row.

The row number is assigned in positions 5 and 6 of the table form. The row numbering must be in ascending sequence from the top of the form (header row) through the noise rows, stub row, table rows, and finally any trailing noise rows. Row number 00 may not be used. The rows do not have to be numbered consecutively; in fact, it is suggested that they are not numbered consecutively to allow for additions or changes. Unused rows, whether in noise rows or table rows, need not be numbered.

The page number is entered in position 7 of each used row of the table body. The page numbering must begin with 1 and must be consecutive. The page number is increased only when an additional form is used for more table columns (sheets added to the right). Sheets added to allow for more table rows (added at the bottom) have the same page number as the sheet they follow.

If additional pages are added to allow more columns, the rows on the additional pages must be given the same row number as the corresponding row on the first page. For instance, if the stub row was numbered 15 and the first table row 20 on page 1, then on all succeeding pages the stub row must also be numbered 15 and the first table row 20.

If additional pages are added to allow more rows (added at the bottom), the row numbering must continue in ascending order from the first page to succeeding pages, each page itself in ascending sequence from top to bottom. Any one table is limited to a maximum of 9 pages, and therefore to the 54 columns as indicated previously.

V OPERATION CODES

A. Test Operation Codes

The test operation codes available in the LJED-225 TABSOL program are listed and defined below. All of the test operations are compatible with GECOM.

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
EQ	Equal	The contents of the stub parameter are EQUAL to the contents (constant or variable) of the operand field in a secondary row entry.
GR	Greater Than	The contents of the stub parameter are GREATER THAN the contents (constant or variable) of the operand field in a secondary row entry.
LS	Less Than	The contents of the stub parameter are LESS THAN the contents (constant or variable) of the operand field in a secondary row entry.
NEQ	Not Equal	The contents of the stub parameter are NOT EQUAL to the contents (constant or variable) of the operand field in a secondary row entry.
NGR	Not Greater Than	The contents of the stub parameter are NOT GREATER THAN the contents (constant or variable) of the operand field in a secondary row entry.
NLS	Not Less Than	The contents of the stub parameter are NOT LESS THAN the contents (constant or variable) of the operand field in a secondary row entry.

B. Result (or Action) Operation Codes

The available result or action operation codes for the LJED-225 TABSOL program are listed below:

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
(blank)	-	Put the constant or contents of the variable field into the stub parameter.
	No Operation	No action or test performed.

1. Arithmetic Operation Codes: (These codes are valid only for binary mode arithmetic operations.)

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
ADD	Add	Add the constant or the contents of the variable named in the data field to the contents of the stub parameter.
SUB	Subtract	Subtract the constant or the contents of the variable named in the data field from the contents of the stub parameter.
MPY	Multiply	Multiply the contents of the stub parameter by the constant or by the contents of the variable named in the data field.
DIV	Divide	Divide the contents of the stub parameter by the constant or by the contents of the variable named in the data field.
EXP	Exponentiate	Using the notation of A^X , the exponentiation routine requires A to be represented as XXXXXX and X.

As previously stated, the LJED-225 TABSOL program operates in fixed point arithmetic. It is extremely important to note that use of the five result or operation codes listed above require the table writer to control or position (housekeep) all constants and variables so that the "decimal points" are in the proper order for the arithmetic operations. For instance, two numbers when being added or subtracted must have the same number of positions to the

right of the decimal (decimal notation). To aid in maintaining the correct decimal notation for various arithmetic operations the following rules apply:

Add

In executing an ADD, both the augend and the addend must have the same decimal notation. The sum will also have the same notation.

Augend	1.234
<u>Addend</u>	<u>1.234</u>
Sum	2.468

Subtract

In subtracting, both the minuend and the subtrahend must have the same decimal notation. The remainder will also have this notation.

Minuend	2.468
<u>Subtrahend</u>	<u>1.234</u>
Remainder	1.234

Multiply

When multiplying, the product will have a decimal notation equal to the sum of the decimal notation of the multiplicand and the multiplier.

Multiplicand	1.250	(decimal notation)
<u>Multiplier</u>	<u>123.4</u>	3
Product	156.2500	+ 1
		4

Divide

In dividing, the quotient will have a decimal notation equal to the decimal notation of the dividend less the notation of the divisor.

	Quotient (Remainder)	40.0 (1)
Divisor	Dividend	1,200 ₍₃₎ 48.0000(4) (4-3=1)
		(decimal notation in parenthesis)

2. Shifts

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
SRN	Shift Right Numeric	Shift the numeric contents of the stub parameter right k positions. ($k \leq 10$)
SLN	Shift Left Numeric	Shift the numeric contents of the stub parameter left k positions. ($k \leq 10$)
SRD	Shift Right and Round	Shift the numeric contents of the stub parameter right (k-1) positions, add 5 to the low order (units) position then shift right one position, ($k \leq 10$)
SRA	Shift Right Alphabetic	Shift the alphabetic contents of the stub parameter right k positions. ($k \leq 6$)
SLA	Shift Left Alphabetic	Shift the alphabetic contents of the stub parameter left k positions. ($k \leq 6$)
SRV	Shift Right Variable	Shift right the numeric contents of the stub parameter k positions where k is the numeric content of the parameter named in the operand. ($k \leq 10$)
SLV	Shift Left Variable	Shift left the numeric contents of the stub parameter k positions where k is the numeric content of the parameter named in the operand. ($k \leq 10$)

3. Indexing

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
PXV	<u>Put an Indexed</u> <u>Variable</u>	Index the data field parameter number by the contents of the Indexer and store the contents of the indexed parameter in the contents of the stub parameter.

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
PVX	<u>Put a Variable</u> in an <u>Indexed</u> Stub	Put the contents of the parameter named in the data field into the parameter derived by indexing the parameter number of the stub parameter by the contents of Indexer.

In addition, two additional operation codes, AXV and AVX, are available that perform the same basic indexing function except they add the contents of a data field parameter to a stub parameter instead of putting (reset to zero and add) the contents. The AXV and AVX operation codes may be used effectively for totaling or summarizing.

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
AXV	<u>Add an Indexed</u> <u>Variable</u>	Index the data field parameter number by the contents of the indexer and add the contents of the indexed parameter to the contents of the stub parameter.
AVX	<u>Add a Variable</u> to an <u>Indexed</u> Stub	Add the contents of the parameter named in the data field to the parameter derived by indexing the parameter number of the stub parameter by the contents of Indexer.

In all cases the parameter numbers as defined in the dictionary are used as a tag to reference one parameter with relationship to another. As mentioned in the dictionary formulation, it is suggested that parameters employing indexing be grouped for purposes of clarity and simplicity. It is in this respect that the assignment of parameter numbers is no longer arbitrary.

Indexing allows the use of generalized tables containing specific values that vary with time. For instance, it might be desired to calculate the areas for one to ten circles, represented by DIAMETER 1 (DIA-1), DIAMETER 2 (DIA-2), DIAMETER 10 (DIA-10). Separate tables could be written for the calculation of each area expressed in terms of DIA-1, DIA-2, DIA-10, but this could become lengthy and redundant. Instead, if the variables DIA-1, DIA-2, etc. were entered in the dictionary as a list (see below) and if indexing is used, one generalized table can be written to test for the presence of DIA-1 DIA-10. The area can then calculate for all existing diameters.

Dictionary

<u>Parameter Number</u>	<u>Parameter Name</u>
011	DIA-1
012	DIA-2
013	DIA-3
014	DIA-4
015	DIA-5
016	DIA-6
017	DIA-7
018	DIA-8
019	DIA-9
020	DIA-10
111	AREA 1
112	AREA 2
113	AREA 3
114	AREA 4
115	AREA 5
116	AREA 6
117	AREA 7
118	AREA 8
119	AREA 9
120	AREA 10

The indexing routine requires three types of tables: an initializing table, an indexing table, and a finalizing table. The indexing operation codes are special in that their format requires two table columns to represent the complete operation as shown in figure 9.

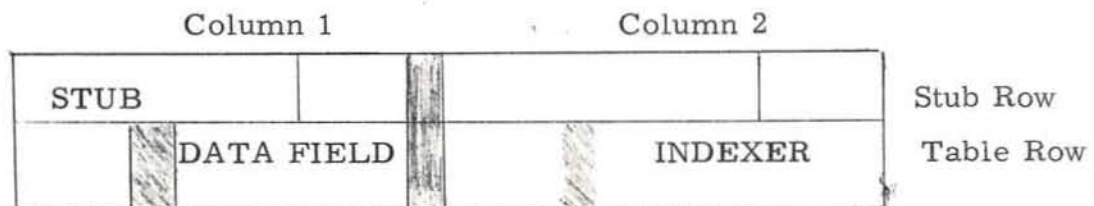


Figure 9. Example of indexing operation code

The initializing table is used to establish a value for the indexer.

TABLE 50	UTRA 60
-	INDEXER
-	Put 0

(Initializing Table)

The indexing table does the actual indexing as a function of the particular indexing operation code used. For purposes of the example the PXV and PVX codes will be used.

TABLE 60	UTRA 70				
-	DIAM	PERFORM	AREA-1	INDEXER	
-	PXV DIA-1	INDEXER	TABLE 100	PVX AREA	INDEXER
					ADD 1

(Indexing Table)

TABLE 100			
DIAM	AREA	GO TO	
≠ 0	$\frac{3.14 \text{ DIAM}^2}{4}$	END	
= 0	-	STOP	

TABLE 70		
INDEXER	GO TO	
< 10	TABLE 60	
= 10	STOP	

(Finalizing Table)

Using the dictionary and the four tables above, the indexing operations would work as follows:

- a. Table 50 is the initializing table and is always executed at the beginning of the routine to establish the correct value in INDEXER. Table 50 universally transfers to Table 60.
- b. The PXV operation in Table 60 does the following:
 - (1) Adds the previously established contents of Indexer (second table column) to the parameter number of the data field parameter (first table column) as defined in the dictionary.
 - (2) Puts the contents of indexed parameter in the contents of the stub parameter.

(The first time through Table 60 with INDEXER = 0, zero is added to 011, the parameter number of DIA-1 as defined in the parameter dictionary. Then the contents of the indexed parameter DIA-1 (011 \neq 0 = 011) are put in DIAM.)

- c. Next, Table 100 is "PERFORMED" and, providing a diameter exists (DIAM \neq 0), the area of the circle represented by DIA-1 is calculated and stored in AREA. If no diameter exists (DIAM = 0) the area calculation is not performed and the indexing stops.
- d. The fourth and fifth table columns of Table 60 store the contents of area in AREA-1 as follows:
 - (1) Adds the contents of Indexer to the parameter number of the stub parameter as defined in the dictionary.
 - (2) Puts the contents of the data field parameter into the contents of the indexed stub parameter.

(The first time through Table 60 with INDEXER = 0, zero is added to 111 (0 \neq 111 = 111), the parameter number of AREA-1 as defined in the dictionary. Then the contents of the data field are stored in the indexed stub parameter.)

- e. One is added to the contents of Indexer ($0 \neq 1 = 1$) and control is transferred to Table 70.
- f. Table 70 tests the contents of INDEXER, finds it less than 10 and goes back to Table 60; this time going through Tables 60 and 100 with INDEXER = 1 calculating AREA from DIA-2 and finally storing the result in AREA-2.

This process continues through each succeeding diameter storing the result in the corresponding AREA. When the contents of INDEXER becomes equal to 10, the indexing loop is completed.

4. The AND Instruction

The AND operation code is used to add either an alpha-numeric constant or a BCD variable to the contents of the stub parameter. The AND operation code should only be used to add BCD information, usually for output purposes.

5. The Convert Instruction (CON)

The convert operation code is used to convert a variable in binary mode to an equivalent 10-digit value in the BCD mode. The four high order (left) digits of the variable will be stored in the parameter named in the stub, the six low order digits in the parameter whose parameter number is the next sequential value in the dictionary.

6. Transfer Instructions

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
PERFORM	Perform and Return	Go to the table or series of tables beginning with the table number in the data field and perform their tests and actions. When an END operation code is encountered, return to the next action in the table row containing the last PERFORM.

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
END	Return	Return control to the next action in the table row containing the last executed PERFORM instruction. The END operation code is only used in conjunction with the PERFORM instruction.

NOTE

The PERFORM instruction is very powerful and can be used very effectively. For instance, one table may PERFORM another table or series of tables linked by GO TO's, each containing any necessary tests or results. The end of the series is identified by an END instruction. Or one table may PERFORM another, the second table also containing a PERFORM, and so on, to a limit of 15 PERFORMS -- one cascading to another. For each PERFORM there must also exist a corresponding END.

GO TO	Transfer Control To	Go to the first test or action if no tests exist of the table identified in the data field,
-------	---------------------	---

NOTE

Both the PERFORM and GO TO instructions must have TABLE XXXX in the data field.

TRA	Transfer	GO TO the first test or action, if no tests exist, of the table identified in the data field, (Accomplishes the same function as GO TO.)
-----	----------	--

7. Output Instructions

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
PCH	Punch	Punch the contents of the 14 variables #486-499 as output.
PCC	Punch and Clear	Punch the contents of the 14 variables #486-499 as output and reset variables 486-497 to zero after punching.

NOTE

Parameters 498 and 499 normally contain the problem number obtained directly from the input data cards. If the table writer has entered other data in parameters 498 and 499, the entered data will be punched.

The operand of the punch or punch and clear instruction may also include a GO TO address; i. e., the number of the table to go to next. If so, the punch instruction must be the last entry in the row. The table number is written as the last four digits of the operand. If the transfer is not used, the operand may be left blank.

8. The STOP Instruction

<u>Symbol</u>	<u>Meaning</u>	<u>Effect</u>
STOP	STOP	Cease processing the existing problem, execute the finalizing routine and start processing the next set of data. If no additional data exists, halt the computer. The operand of a STOP instruction is left blank.

VI PROBLEM INPUT DATA FORMAT

The LJED-225 TABSOL program utilizes a punched card input. There are two input formats available: one for numeric input to be converted to the binary mode for arithmetic operations; the other for data that can remain the BCD mode.

The first seven columns of the input card contain the problem number. Any unused columns must be filled with zeroes. The eighth card column contains---

A "0" punch if the card contains numeric data to be converted to the binary mode for processing.

A "1" if the card contains alpha-numeric data to remain in the BCD mode.

The remaining seventy-two card columns are divided into eight 6-digit data fields and eight 3-digit parameter number fields for data input.

<u>6 Digit Data Fields</u>	<u>Corresponding 3 Digit Parameter Number Fields</u>
9-14	15-17
18-23	24-26
27-32	33-35
36-41	42-44
45-50	51-53
54-59	60-62
63-68	69-71
72-77	78-80

Numeric input data cards (identified by "0" in column 8) require all unused data field positions and all unused parameter number fields to be filled with zeroes.

Unused data field positions of alpha-numeric input cards (identified by "1" in column 8) may be left blank, but all unused 3-digit parameter number fields must be filled with zeroes.

The parameters on input data cards may be arranged in any order, and the input cards themselves may be submitted in any order. Specialized input sheets can be made for particular problems as desired, each within the limits and specifications described above.

VII OUTPUT FORMAT

The LJED-225 TABSOL program may be obtained either on punched cards or through an "on-line" printer. TABSOL running instructions, Section XIV indicate the procedures for obtaining on-line printer output.

The output format normally consists of twelve 6-digit data fields (parameters #486-497) in columns 1-72, followed by an error identification column (73), and then a seven digit problem number field, columns 74-80. The error identification and problem number are contained in parameters 498 and 499.

The twelve 6-digit data fields may be mixed alpha-numeric as desired. The error identification column will contain an 8 punch if the card is a problem error card as explained in Error Identification. The seven digit problem number field contains the problem number from the data input card.

It is possible for the table writer to use parameters 498 and 499 for output data -- realizing that the problem number will be destroyed if not stored elsewhere and will not appear on problem error or output cards. When using parameters 498 and 499 for data output, the six digits of parameters 498 will be output in columns 73 through 78. Only the 2 high orders (left) positions of a parameter 499 can be obtained as output: they will appear in columns 79&80.

As described previously, the conversion (CØN) instruction must be used to output a variable in the binary mode, occupying two parameters to obtain all ten characters.

The AND instruction will be used to pack or add BCD variables for output.

Negative output values will be distinguished by a 11-X over the units' position of the data field. Leading zeroes are not punched in the card output nor printed in the printer output.

It is possible by data manipulation to use output from one program or core load as input to a subsequent program with no additional processing. This requires output data manipulation by the table writer.

VIII ERROR IDENTIFICATION

If any table solved during the processing of one problem has an "error" in solution, a "512" will be typed on the typewriter along the table number in which the error occurred. All subsequent output will have an 8 in column 73. As defined previously, an error is a failure to pass all the tests in any single table row. All error messages are described in the TABCON and TABSOL Error Identification Sections.

IX SETTING UP A TABSOL SYSTEM

The following information is from TIS Report No. R60MS312, 650 TABSON-TABSOL, by T. F. Kavanagh, J. H. Kelly, and P. G. Margaritis.

A. Design the System

Develop the required output, input, and the logic required to convert the input data to the appropriate output. If the system is too large to be handled by a single group of tables, plan the division into logical, self-sufficient segments, each to be expressed in a group.

B. Write The Tables

Write the structure tables, and their associated parameter dictionary in the tabular systems oriented language (TABSOL).

If the system contains several groups of tables, it is probably wisest to start with a single group and carry it through to completion before starting detail work on the remaining groups, since the experience gained from actually writing and using tables will prove more helpful in understanding how to use TABSOL than any amount of reading.

C. The parameter dictionary and the tables must then be keypunched. Key-punching instructions are given in Section X.

D. Check the parameter dictionary cards to see that there is no unintentional duplication of parameter numbers or names.

E. Check that all the variables used in the tables are listed in the dictionary.

F. The dictionary cards are not required to be in any sequence for processing.

G. The table description cards and table cards should be checked for any obvious errors that can be corrected before processing.

H. The table descriptor cards must be extracted from the table cards. The descriptor cards should then be sorted to ascending sequence on table number, card columns 1-4, or 16-19.

I. Sort the remaining table cards to ascending sequence on table numbers, row number and page number; i. e., columns 1-7.

J. If the cards are then re-assembled dictionary cards first, header cards next, and then table cards, the deck is ready for processing.

X KEYPUNCHING

A. Dictionary

The parameter dictionary is keypunched directly from the dictionary form -- row by row.

Card Columns

1- 8	Blank
9-11	Parameter Number
12	Blank
13-18	Parameter Name
19-79	Blank or any desired notes or comments
80	"D"

B. Table Forms

The structure tables are also keypunched directly from the table forms -- row by row.

1. Header (Descriptor) Card

Card Columns

1- 4	Table Number
5- 6	Row Number
7	"0"
8- 9	Blank
10-14	"TABLE"

Card Columns

15	Blank
16-19	Table Number
20	(period)
21	Blank
22-23	Number of Conditions
24	Blank
25-34	"CONDITIONS"
35	Blank
36-37	Number of Actions
38	Blank
39-45	"ACTIONS"
46	Blank
47-48	Number of Rows
49	Blank
50-53	"ROWS"
54	(period)
55	Blank
56-59	"NOTE"
60	Blank
61-64	"UTRA"
65	Blank
66-69	Universal Transfer Table Number
70	Blank
71-74	"ETRA"
75	Blank
76-79	Error Transfer Table Number
80	(period)

2. Noise Cards

Card Columns

1-4	Table Number
5-6	Row Number
7	Page Number
8-80	These columns are left for any statements, expressions, or other information the table writer may desire to include but not have processed.

3. Table Cards

Card Columns

1-4
5-6
7
8-80

Table Number

Row Number

Page Number

Columns 8-80 are punched directly as written on the table form. Columns 8, 20, 32, 44, 56, 68, and 80 must have a 12-4-8 punch.

C. Problem Input

Card Columns

1-7
8
9-80

Problem Number

"0" for numeric data

"1" for alpha-numeric data

Problem input data and parameter numbers

Data Fields	Corresponding Parameter Numbers
9-14	15-17
18-23	24-26
27-32	33-35
36-41	42-44
45-50	51-53
54-59	60-62
63-68	69-71
72-77	78-80

XI MACHINE REQUIREMENTS

The minimum machine configuration for the LJED-225 TABCON and TABSOL programs is a 8000 word card in - card out GE225 with a BCD package. If other hardware is available, (tapes, printers, etc.) it may be used to increase the effectiveness and efficiency of the programs.

XII TABCON TABLE TRANSLATOR PROGRAM

A. The Three Phases

The first phase of the LJED-225 TABCON program processes and edits the Parameter Dictionary cards. A parameter list is created of all parameter names with their corresponding symbolic locations.

The second phase processes the table descriptor cards; a list is established of all the information in the descriptor cards (conditions, actions, rows, UTRA, ETRA). At the end of this phase, the list is edited and diagnostic comments are typed.

If any errors were found in either the Dictionary or Descriptor cards, (phase 1 and 2), the computer stops at this point so the errors can be corrected and then the cards rerun, repeating phase 1 and 2 until all errors are corrected.

The third phase processes the table cards. These cards are processed entry and edits are made on the blocks themselves and against information in Descriptor table from the second phase. As the table blocks are processed, General Assembly Program (GAP) cards are punched (or printed on line. Two types of errors can occur in the table entries. One type is noted, the punching or printing of GAP cards stopped, but processing is continued with diagnostic comments being typed; the other requires that all processing be stopped until the error is corrected.

The third phase is terminated by an END card ("END" in columns 1-3) causing the program to revert back to the first phase and start processing parameter dictionary cards for the next set of tables. If, however, the card following the "END" card is an "END SENTINAL" card, processing stops and "END" is typed.

B. LJED-225 TABCON Error Print-outs

<u>Error Code</u>	<u>Errors in Main Program</u>	<u>Action</u>
A01	UTRA in Descriptor card not in table number list. Table number and UTRA are printed.	Continue

<u>Error Code</u>	<u>Errors in Main Program</u>	<u>Action</u>
A02	ETRA in Descriptor card not in table number list. Table number and ETRA are printed.	Continue
A03	At least one A01, A02 and/or B01 Error was found in descriptor table edit.	Restart
A04	Invalid card in deck.	Restart
<u>Error in Dictionary Card Processor</u>		
B01	Parameter number greater than 499 Parameter number printed.	Continue
<u>Error in Description Card Processor</u>		
C01	Descriptor card out of required ascending sequence.	Restart
<u>Errors in Table Card Pre-Processor</u>		
D01	Table out of ascending order Table number printed.	Restart
D02	More rows in table than indicated in Descriptor. Table number printed.	Restart
D03	Second table entry for indexing command not furnished. Table number printed.	Restart
D04	No GO TO intable descriptor or UTRA in table row Table number printed.	Restart
<u>Errors in Processing Table Blocks</u>		
E01	Stub missing from table column Table number printed.	Restart
E02	Parameter name not in parameter dictionary. Table number and missing parameter printed.	Continue
E03	Character other than quotation marks or blank in column used to denote LITERALS, and column is not a GO TO or PERFORM. Table number and bad character printed	Continue

E04	Invalid operation code Table number and operation code printed.	Continue
E05	Invalid table number in table operand Table number and invalid number printed.	Continue
E06	Table entry is TABLE XXXX but stub is neither "PERFORM", nor "GO TO", Table number and operand are printed.	Continue
E07	More columns in row than indicated in Descriptor. Table number printed.	Restart

XIII TABSOL TABLE SOLVING PROGRAM

A. TABSOL Executive Program

The first phase of the TABSOL program sets specified parameters to zero and then begins reading problem input cards. The information contained in the input cards is stored into the specified parameters. During this processing, the following edits are made:

1. Make sure that there is either a "0" or "1" in Columns 8 of the input card to signify numeric or alphanumeric information to be stored.
2. Parameter number specified is not greater than 499.
3. Problem numbers are in ascending sequence.

The second phase is initiated when a change in problem number is encountered. At this point, input card reading is ceased and control is transferred to the table solving area to solve the problem.

After completion, of a problem, specified parameters are again set to zero and the next set of problem input cards are read as described in Phase I. This process is continued until an end sentinel card is encountered, at which time the last problem is processed and code "505" will be typed to signify end of job

B. TABSOL Subroutines

All the subroutines for the arithmetic and logical table operations are included in TABSOL. These subroutines are assembled in fixed areas in lower memory. The GAP program produced by TABCON calls on these subroutines and on the parameter area symbolically. Therefore it is necessary, when assembling this program, to provide "LINKAGE CARDS" which gives the absolute locations of these subroutines and of the parameter area by the use of the pseudo-operation "EQO". The above subroutines maintain a ten decimal digit register in all operations.

C. TABSOL Error Printouts

<u>Error Code</u>	<u>Meaning</u>	<u>Action</u>
501	Input card out of sequence by problem number	Stop
502	Character other than 1 or 0 in Col. 8 of input card	Stop
503	Error on BCD to binary conversion (machine error)	Stop
504	Parameter number greater than 499 specified on input card	Stop
510	Arithmetic table error, Table number printed and program continues at ETRA	Continue
512	A transfer was made to ETRA Table number printed and program continues at ETRA.	Continue

XIV TABCON AND TABSOL RUNNING INSTRUCTIONS

A. Deck Set-Up for TABCON

1. TABCON binary deck.
2. Card with "TABCON" in Col. 1-6.
3. Dictionary cards (any sequence).
4. Descriptor cards (ascending sequence).
5. Table cards (must be in same order as descriptor cards).
6. Card with "END" in Col. 1-3 and for last set of tables being processed.
7. End sentinal card (all punches in Col. 1-3 and "END" in Col. 4-6).

B. Running TABCON

1. Load deck from above in card reader.
2. Before any processing, the typewriter will print, "SW19 to PRINT".
3. To start at this point -
 For punched cards - toggle sign switch
 For printed output - depress SW 19 and toggle sign switch
4. Program will proceed to produce GAP output and will type "END" at end of job.
5. When all the table cards have been processed through step 4 with no errors, place the TABSOL "LINKAGE CARDS" in front of output deck from step 4 and assemble using GAP.

C. Deck Set-Up for TABSOL

1. TABSOL binary deck.
2. TABCON generated binary deck from Step B, 5 above.
3. Transfer card.
4. Card with "TABSOL INPUT" in Col. 1-12.
5. Problem input cards.
6. End sentinal card (all punches in Col. 1-3 and "END" in Col. 4-6).

D. Running TABSOL

1. Load deck on card reader.
2. Before any output is produced, the typewriter will print "SW 19 to PRINT".
3. To start from this point --
 For punched cards - Toggle sign switch
 For printed output - depress SW 19 and toggle sign switch
4. Program will continue in the mode selected and will type "505" at end of job.

E. TABCON and TABSOL Messages

The following typed messages from card input subroutine are used in both TABCON and TABSOL.

<u>Typed Codes</u>	<u>Meaning</u>
CARD (Input Label)	O. K. - Go On
Card (Input Label) No	Compare card label to program label. If not acceptable, reload proper deck in card reader and set switch 0. If label is acceptable, set switch 19 and then 0
Load (Input Label) FD	Non-matching label has been forced
Bad Card Read	Remove deck from reader. Place last card read in front of deck and replace cards in reader. Depress switch 0
CRD	Hopper empty, but no END SENTINAL card has been read. Reload hopper with remaining cards and depress switch 0