CoA REPORT AERO No. 203

THE COLLEGE OF AERONAUTICS
CRANFIELD

DESCRIPTION OF THE ALGOL VERSION OF THE "TURBOCODE" SCHEME FOR THE PROGRAMMING OF THERMODYNAMIC CYCLE CALCULATIONS ON AN ELECTRONIC DIGITAL COMPUTER

by

J. R. Palmer and Sq. Ldr. K. P. Annand

CoA Report Aero No. 203
March, 1968

THE COLLEGE OF AERONAUTICS
CRANFIELD

Description of the Algol Version of the "Turbocode" Scheme for the Programming of Thermodynamic Cycle Calculations on an Electronic Digital Computer

by

J. R. Palmer, M. A., C. Eng., F. R. Ae. S. and
Sq. Ldr. K. P. Annand, B. A., D. C. Ae., R. A. F.

SUMMARY

The "Turbocode" Scheme for programming thermodynamic cycle calculations on an electronic digital computer was described in detail in an earlier Report (ref. 1) in terms of the Ferranti "Pegasus" version. The present Report describes a version, coded in Algol 60, which has been implemented on an I. C. T. 1905 computer. In view of the widespread understanding and use of Algol, the Scheme in its present form may be of considerably wider applicability than the original, subject only to variations in hardware representations, and a full listing of the Scheme is therefore presented, together with flow diagrams and additional descriptive and explanatory material.

CONTENTS

CONTENTS continued

1. Introduction

An earlier Report (ref. 1) described the Turbocode Scheme developed in the Propulsion Department of the College of Aeronautics for the programming of thermodynamic cycle calculations on an electronic digital computer. The original scheme described therein was programmed for the Ferranti "Pegasus", but it was stated that a follow-up Report on an Algol version, suitable for most of the larger present-day computers, would be issued as soon as possible. The present Report describes this version, as programmed for an I. C. T. 1900 Series Computer (specifically a 1905 with 32K core storage), and a full listing of the Scheme in the hardware representation of that machine is given, together with flow diagrams of the major elements of the Scheme, and descriptions of points of difference between the "Pegasus" and 1900 versions.

The descriptions and specifications of the various Functions, Subroutines and Bricks are not repeated here, and the reader should refer to ref. 1 for this information.

2. Brief Résumé of Basic Concepts

Although it is intended that the reader should refer to ref. 1 for a fuller description of the detailed action of the Scheme, it is desirable that the present report be self-contained as regards its description of the basic scheme of operation of Turbocode. To this end, the following brief notes are given to define the principal concepts, even though they appear also in ref. 1.

Each major portion of the Scheme, concerned with such operations as Data Input, Results Output and calculation of conditions at exit from a particular type of component, is known as a Brick. Most Bricks deal with a single component or process, and it is convenient to think of a Brick as a kind of operator which, operating on the given gas state at inlet to the process, calculates the gas state at outlet. By standardising the layout of gas state information in the various Bricks, a "common interface" approach becomes possible, and the thermodynamic Bricks can be thought of as being "plugged in" to one another in the sequence dictated by the Master Program.

The array of quantities defining the gas state at any station within the cycle is known as the Station Vector of that station, and consists of the following eight quantities:-

fuel-air ratio	ALPHA	(dimensionless)
mass flow rate	W	(lb/s)
static pressure	PSTATIC	($lbf/in^2 abs$)
total pressure	PTOTAL	($lbf/in^2 abs$)
static temperature	TSTATIC	(^oK)
total temperature	TTOTAL	(^oK)
flow velocity	VELOCITY	(ft/s)
flow area	AREA	(ft^2)

This is in fact a redundant set, but it is convenient to have rather more than the minimal set of five items (ALPHA, one of PSTATIC and PTOTAL, one of TSTATIC and TTOTAL, and any two of W, VELOCITY and AREA), to give maximum flexibility.

The Station Vectors alone seldom completely define the action of a Brick, however: normally additional information is needed, such as efficiencies, loss factors, pressure ratios, etc. and these items constitute the Brick Data. Each Brick has its own particular requirements as to Brick Data, described in its specification, and it is the Turbocode programmer's responsibility to ensure that the items of Brick Data are listed and made available as required by his own program.

Similarly, certain results are produced which are different in kind from the Station Vectors, e.g. thrust and specific fuel consumption: these are known collectively as the Engine Vector, and again its composition is controlled by the Turbocode programmer, using the relevant Brick Specifications. Certain Bricks use, as data, items generated in the Engine Vector by previous Bricks.

Provision is therefore made for the input of (partial) Station Vectors (Brick 15) and of Brick Data (Brick 16), and for the output of Station Vectors (Brick 31) and of the Engine Vector (Brick 32). The latter Brick also provides for resetting the Station Vectors, after they have been printed, to the values prevailing at the start of the current calculation, thereby avoiding the necessity of providing new data except where this differs from that of the preceding calculation. A further Brick (Brick 22) provides for arithmetic manipulation of Engine Vector Items (e.g. adding of thrusts and fuel flows of different components), and for the transfer of items in either direction between the Engine Vector, the Station Vectors and the Brick Data.

Since the actual operations required for a cycle calculation are fully programmed in the Turbocode Scheme itself, the Turbocode Master Program written by the user consists merely of instructions to link the requisite Bricks in the desired order, together with cross references to the Station Vector, Brick Data and Engine Vector items required. To make this as simple as possible, each Turbocode instruction takes the form of a Codeword, having a maximum of seven items consisting of unsigned integers, separated by commas, and each written on a separate line. Few Bricks require all

seven items, in which case arbitrary integers (usually zeros) can be used to complete the Codeword, though it is preferable to omit any right-hand zeros since the Codeword Input sequence can automatically fill the gaps with zeros. The meanings of the seven items are normally as follows:-

"Pegasus" notation	1900 Notation while in store	1900 Notation while being obeyed	Meaning
n	CW [K, 0]	NEXTBRICK	No. of brick required
a	CW [K, 1]	A	Inlet Station Vector No.
b	CW [K, 2]	B	Outlet Station Vector No.
c	CW [K, 3]	C	First Brick Data Item No.
d	CW [K, 4]	D	First Engine Vector Results Item No.
e	CW [K, 5]	E	First Engine Vector Data Item No.
f	CW [K, 6]	F	Jump Codeword No.

Each Codeword in the Master Program is numbered from zero, though this number does not appear in the Codeword as written. In the above table, K is the number of the Codeword currently being obeyed, and F is the number of the Codeword to be obeyed next, if this is not the next in Sequence. A typical Codeword is

4, 9, 10, 31, 4, 2, 30

which means "use Brick 4 (Single Turbine) with Inlet Station Vector 9, Outlet Station Vector 10, First Brick Data Item No. 31, First Engine Vector Result Item No. 4, First Engine Vector Data Item No. 2 and Jump Codeword Number 30". When using this particular Brick, a Jump occurs only if the given exit area is too small for the given inlet conditions.

3. Alterations Necessitated by Re-Coding in Algol

3.1 Preliminary Considerations

The form of the "Pegasus" version of the Scheme was in part dictated by certain features peculiar to that computer viz:-

(a) The storage capacity was limited to 7K words, which was insufficient

to accommodate all Bricks simultaneously. It was therefore necessary to employ selective assembly of Bricks from a library tape in accordance with the requirements of each particular Master Program: since many of the Bricks are mutually exclusive, and since no cycle could be envisaged which would require more than about half the total brick storage space, this introduced no operational limitations, but it did involve a rather time-consuming process of program input.

(b) Owing to lack of programming personnel, it was never possible to realise the original intention of coding the entire scheme in "Pegasus" machine code: in fact, only the Codeword Input, Assembly, Codeword Obey and Error Tracing facilities were so coded, together with the four fundamental routines for calculating specific heat, enthalpy, temperature-dependent entropy and combustion fuel-air ratio. All the Bricks, and the Subroutines employed by them, were coded in "Pegasus" Autocode.

The interpretive nature of this Autocode, together with the necessity for its performing floating-point arithmetic on a fixed-point computer by software, imposed severe restrictions (by a factor of the order of 20) on the already modest operating speed of the "Pegasus", and on the available storage space, since the Autocode Scheme itself occupied 890 words.

(c) The above shortcomings inherent in the computer system employed were partially offset by the ease with which programs, whether in machine code or Autocode, could be segmented and subsequently assembled.

In considering the adaptation of the Turbocode Scheme to a more modern computer, of far greater speed and storage capacity (in fact an I. C. T. 1905), three considerations were paramount:

(1) From the user's point of view, the method of using the Bricks, and writing the Master Program and Data, should be identical with that of the "Pegasus" version.

(2) The coding should be in one of the widely used "universal" languages: this limited the choice to Algol or to Fortran, since Extended Mercury Autocode was regarded as insufficiently powerful or general, while the properties and availability of PL/1 were not known at the time the work was started.

Since it has always been a fundamental requirement that the Scheme could be augmented by additional Bricks whenever necessary, it was desirable that this should be accomplished using a source language widely used within the College, and since a policy decision had already been taken to make instruction in Algol available to all College personnel, this language seemed the obvious choice.

(3) In view of the greatly increased storage capacity available (initially 32K words of core store, subsequently to be augmented by magnetic tape and disc storage), it would be possible to store the entire Scheme, compiled in machine code form, within the computer: this would greatly simplify and speed up the program input phase even

while paper tape input remained necessary, while eventually the whole Scheme would be permanently stored on magnetic tape or discs. In any case, the block structure of the Algol 60 Reference Language does not allow for segmentation of programs, though particular hardware representations (of which the 1900 Series version is one) may permit it.

3.2 The Scheme Structure Adopted

The development of the Algol version of the Scheme was carried out in advance of the installation of the I. C. T. 1905 computer, and consequently a variety of other machines was used (specifically the Ferranti "Pegasus", the Elliott 803 and two Elliott 4100 Series machines). In most cases, program segmentation was not possible, so as the Scheme was built up Brick by Brick the storage requirements during the compilation and/or running phases outgrew the capacities of the smaller machines employed, and compilation became excessively slow. The first definitive Algol version was therefore developed as a single segment, albeit of considerable length, and is described in this form in this Report: prospective users might need to segment it in accordance with the capacities and conventions of their own installations. It is intended in future to make use of this apparent restriction to collect the Bricks into a library, but as this facility is not available to all compilers, and has not been fully studied as to its feasibility in the present context, it is not proposed to describe this aspect, either now or in subsequent work. Should this reversion to the pattern of the "Pegasus" Scheme be implemented, it would avoid the need to recompile all or part of the Scheme whenever a new Brick was added: furthermore, the 1900 Series compilers for various languages are designed to be compatible in that they produce semi-compiled output in a standard form, making it possible to consolidate a program such as this Scheme from segments written in a variety of source languages. Future additions to the Scheme are not therefore constrained to be written in Algol.

The decision to write the Scheme as a single program (whether segmented or not) entailed one basic change in program structure.

In the "Pegasus" Scheme, the mode of operation was:-

(a) Master Program read in and stored by Codeword Input, which listed the Bricks required.

(b) Bricks selected from Library by Assembly; and

(c) Codewords decoded and obeyed by Codeword Obey (employing the appropriate Bricks as subroutines, which in turn called up the Subroutines proper and Functions as required).

In the Algol Scheme, regarded as the "driver" program, the Master Program becomes a species of Data. The Codewords of the Master Program are read in by Codeword Input and stored as elements of a two-dimensional array CW, each "row" corresponding to one codeword, and each "column" to one of the seven elements of the standard form of codeword. Codeword Obey then uses these array elements as parameters for the Bricks which it

calls up, which are written as distinct blocks. It must be emphasized that the Bricks are not written as Algol procedures, since their inputs and outputs (Station Vectors, Engine Vector and Brick Data) are global in nature. Nevertheless, in effect, each Brick with its associated codeword behaves like a procedure, of which the codeword elements are the actual parameters in the strict Algol sense.

It will be seen from the appended listing and flow diagrams that Codeword Obey utilises a switch designator called BRICK to call up the individual bricks, its elements being the labels B1, B2, etc., corresponding to Brick 1, Brick 2, etc. The subscript of this switch designator is called NEXTBRICK, which is set equal to the first element (i.e. the required Brick Number) of the codeword currently being obeyed. Labels corresponding to non-existent Bricks cause output of an appropriate diagnostic message, followed by a search for another Master Program.

The Subroutines are written as procedures, with identifiers of the form Sn, where n is the Subroutine number, while the Functions are written as function procedures with the same identifiers (SPHT, ENTH and PRES) as were used in the "Pegasus" version. (The omission of the FUEL function is explained in Appendix 4). Those Standard 1900 Series procedures which are used are described in Appendix 1.

In listing the Scheme, sufficient comment sequences have been incorporated to make much of the Scheme self-explanatory. Further details of the method of specifying and using each Brick, and of their mode of operation, are given in ref. 1, and the present Report mentions (in Appendices 2 and 3) only points of difference between the two versions of the Scheme. Appendix 1 describes features peculiar to the 1900 Series implementation of Algol.

4. Proposed Future Developments

The Turbocode Scheme described in ref. 1 and in the present Report is intended primarily for design-point calculations, although Bricks 25 (Off-Design Convergent-Divergent Nozzle), 34 (Determination of Bypass Ratio), 35 (Determination of Off-Design Turbine Inlet Temperature) and 36 (Determination of Off-Design Intake Mass Flow) permit limited off-design investigations, assuming constant component efficiencies, and choking of all turbine nozzle guide vanes and propelling nozzles.

It is obviously desirable to develop a much more comprehensive scheme for off-design calculations which can deal with the full characteristics of the various components, and which can cope as efficiently yet generally as possible with the large-scale iterative processes involved. Such a scheme - temporarily entitled "Characteristic Turbocode" - is in process of development, and will be reported on in due course. At present this experimental scheme is not compatible with the Turbocode Scheme described here in respect of the format of the Master Program, and it may well be that the present scheme will require extensive modification if such compatibility is found to be desirable. Whether or not experience confirms this, much of the material

incorporated in the present Scheme will naturally be incorporated in the new one.

As has been pointed out earlier, one of the fundamental concepts of Turbocode is that Bricks should be added to it as and when the need arises. It is not proposed to report on such additions, but copies of them can be made available to prospective users.

5. Conclusions

The Algol version of the Turbocode Scheme, originally developed for the "Pegasus" computer, is now available, and is fully described by the present Report, supplemented by ref. 1. Further information, and copies of the current version of the Scheme and of future additions to it (in I. C. T. 1900 Series hardware representation on punched cards or 8-channel punched paper tape) may be obtained from the Department of Aircraft Propulsion of the College of Aeronautics.

Acknowledgement

The co-operation of International Computers and Tabulators Limited, in permitting use of the material on which Appendix 1 is based, is gratefully acknowledged.

References

1. Palmer, J. R. — The "Turbocode" Scheme for the Programming of Thermodynamic Cycle Calculations on an Electronic Digital Computer - College of Aeronautics Report Aero 198 - July 1967.

2. Naur, P. (ed.) — Revised report on the algorithmic language ALGOL 60 - International Federation for Information Processing - 1962.

3. — 1900 Series Algol Manual - International Computers and Tabulators Limited - 1967.

Appendix 1 - Notes on the I. C. T. 1900 Series Implementation of Algol 60

(The following notes are based, with permission, on ref. 3.)

A. 1. 1 Hardware Representation

Two modes of representation are available - the "Full" Mode incorporating lower case letters, and available only on paper tape, and the "Normal" Mode, which is available on both paper tape and punched cards. Only the Normal Mode will be described and used here. The basic symbols which have different representations in the Algol 60 Reference Language and in the Normal Mode are:-

Algol 60 Reference Language	Normal Mode
Upper Case Letters	(not represented)
Lower Case Letters	Corresponding Upper Case Letters
Underlined, or Heavy Type, Delimiters (e.g. true, begin, for, etc.)	Corresponding words enclosed by single apostrophes (e.g. 'TRUE', 'BEGIN', 'FOR', etc.)
× (multiplication sign)	* (asterisk)
÷	'/'
⩾	'GE'
⩽	'LE'
≠	# or 'NE'
¬	'NOT'
∧	'AND'
∨	'OR'
⊃	'IMPL'
≡	'EQUIV'
10	& or '10'
␣	% or '-'
(	'('
)	')'

In addition, certain further Underlined (Heavy Type) Delimiters are used, of which only external ('EXTERNAL') is employed here to represent the body of a procedure which is in the 1900 Algol Library, but which requires explicit declaration.

A. 1. 2 Library Procedures

The 1900 Algol Library contains a large number of standard procedures, most of which require no declaration (Implicitly Declared Procedures), while others (Explicitly Declared Procedures) must be declared using a particular procedure heading in standard form, with the symbol external as procedure body. The following brief notes describe only those procedures actually employed in the Turbocode Scheme, and are intended merely to indicate their main features as an aid to understanding the listing. Fuller details are given in ref. 3.

A. 1. 2. 1 Implicitly Declared Procedures

In addition to the 9 function procedures defined in the Algol Report (ref. 2) - viz.: SIN, COS, ARCTAN, SQRT, LN, EXP, ABS, SIGN and ENTIER - many other procedures of this type are provided, mainly for input and output purposes. Those employed here are:-

(a) real procedure READ

Reads next number from input. Acceptable terminators are: double space, comma, semicolon and newline (or new card).

(b) procedure PRINT (QUANTITY, M, N)

Outputs the real variable QUANTITY in a format determined by the integer variables M and N.

If $M = 0$ and $N \neq 0$, floating point output is used, with an argument d of (N + 1) significant digits in the range $1 \leqslant d < 10$, and a two-digit exponent.

If $M \neq 0$ and $N \neq 0$, fixed point output is used, with M digits before the decimal point and N digits after it.

If $M \neq 0$ and $N = 0$, integer fixed point output is used, with M significant digits.

(c) procedure SPACE (N)

Outputs N spaces

(d) procedure NEWLINE (N)

Outputs N newlines

(e) procedure PAPERTHROW

Outputs a paperthrow to the head of a new page on the line printer

(f) procedure WRITETEXT (⌜STRING⌝)

Outputs the given string STRING. Layout characters, enclosed between ineer string quotes, may be employed thus:-

⌜nS⌝ output n spaces

⌜nC⌝ output n newlines

⌜nP⌝ output n paperthrows

(g) integer procedure READCH

Reads a single character and gives it an internal integer code value.

(h) integer procedure CODE (⌜CH⌝)

Generates the internal integer code value of the string CH. CH is normally a single character, but the following symbols are also used:-

EL end-of-line code (i.e. newline or end-of-card)
SS space symbol code (i.e. string space␣ or %)

(i) procedure PRINTCH (I)

Outputs a single character whose internal integer code value has previously been assigned to I by a READCH or CODE call.

(j) procedure SELECTINPUT (N)

Selects input channel N

(k) procedure SELECTOUTPUT (⌜N⌝)

Selects output channel N

(l) procedure COPYTEXT (N)

Copies characters from input to output until the string N is encountered: N itself is not copied.

(m) procedure PAUSE (N)

Halts the program and outputs the message HALTED:- N to the console typewriter, where N is an integer. The operator can restart the program if desired.

A. 1. 2. 2 Explicitly Declared Procedures

(a) Boolean procedure TEST (X); value X; integer X; external;

Assigns the value true to TEST if sense switch number X is on,

or false if it is off. This switch is set by the operator (though it is also possible to program this). The Switch is off when program execution starts.

(b) procedure TIMENOW; external;

Outputs time in form HH(hours)/ MM(minutes)/SS(seconds)

(c) procedure DATENOW; external;

Outputs date in form DD(day)/MM(month)/ YY(year)

Appendix 2 - General Observations on the Details of the Scheme

A. 2. 1 Layout of the Scheme

As the listing shows in more detail, the Scheme is laid out in the following order:-

(1) Global Type, Array and Switch Declarations

(2) Procedure Declarations for Explicitly-Declared Library Procedures DATENOW, TIMENOW and TEST

(3) Function Procedure Declarations for the Functions SPHT, ENTH and PRES

(4) Procedure Declarations for the Subroutines S2, S5, S9, S10, S16 and S111217.

(5) Start of the Program Proper, concerned with initialising certain variables, and arrays, selecting and setting up the peripherals and printing the Scheme title and the date.

(6) Codeword Input (starting at the label INPUT)

(7) Codeword Obey

(8) Error sequence used if a codeword is improperly formed, or if there are too many codewords (starting at the label NEXTPROGRAMME).

This sequence also bears those labels which have no associated Bricks.

(9) The Bricks

A. 2. 2 The Global Variables

The real variables TWOGCJ, GC and R are self-explanatory. CONST is used for various temporary constants, LCV for the Lower Calorific Value of the Fuel, and QS for its stoichiometric fuel-air ratio. All of these quantities are constants, and were given identifiers merely for ease of writing the program.

The integer variables are used as follows:- I is 1 greater than the serial number of the last codeword in the program (0 being the first) and is therefore equal to the total number of codewords in the program, K is the serial number of the codeword currently being obeyed, and NEXTBRICK, A, B, C, D, E, F are the seven items of this codeword. AA, BB, CC, DD and EE are copies of the second to sixth codeword items of Brick 26, which is used in conjunction with Brick 27 or Brick 34 to extend the number of codeword items available. (see ref. 1).

The real arrays are as follows:- ALPHA, W, PSTATIC, PTOTAL, TSTATIC, TTOTAL, VELOCITY and AREA are the station vector elements, while XALPHA, XW, XPSTATIC, XPTOTAL, XTSTATIC, XTTOTAL,

XVELOCITY and XAREA are copies of the initial values of these elements, used by Brick 32 for resetting at the end of each calculation. BRICKDATA and ENGINEVECTOR are self-explanatory, while A1 and A2 are polynomial coefficients for SPHT, D1 and D2 for ENTH, and C1 and C2 for PRES.

The integer arrays are CW, the codeword elements, and CD, the internal integer codes for the characters 0 to 9, end-of-line symbol, comma and right parenthesis, which are the only characters that are meaningful to Codeword Input.

The Boolean Variables are :- FIRST, which is true before execution of the Master Program has started, and false thereafter; CHECK, which is set true (by sense switch 1) if error tracing printout is to be used; and SENSE, which is set true by sense switch 2 if diagnostic printout of the Station Vectors and Engine Vector is required after a program failure.

A. 2. 3 Restrictions on Master Programs

Because of the limited sizes of the relevant arrays (which could, however, be altered very readily), Master Programs are subject to the following restrictions:-

(1) Number of Codewords must not exceed 51 (serial nos. 0-50 inclusive).

(2) Number of Station Vectors must not exceed 21 (serial nos. 0-20 inclusive).

(3) Number of Brick Data Items must not exceed 101 (serial nos. 0-100 inclusive).

(4) Number of Engine Vector Items must not exceed 51 (serial nos. 0-50 inclusive).
Also in the Scheme as listed here,

(5) Brick Numbers outside the limits 1 to 40 inclusive, and also Brick Numbers 7, 11, 13, 14, 17 to 21 inclusive and 28 are not allowed (since Bricks bearing these numbers do not at present exist). The addition of further Bricks may well modify this limitation in the future, but in no case may a Brick Number less than 1 be used (otherwise a zero or negative subscript to the switch designator BRICK would be generated). The label B40, which appears in the Codeword Obey sequence, is the remnant of Brick 40, which was originally used to set the value of the Boolean variable CHECK alternately true and false: this function has now been taken over by the use of Sense Switch 1, but the label B40 has been retained to permit use of Master Programs written with B40 incorporated.

Appendix 3 - Notes on Codeword Input, Codeword Obey and Optional Printing

A. 3. 1 Codeword Input, with Error Sequence (See Flow Diagram in Fig. 6)

This sequence is normally entered at the start of a run, as soon as initialisation of the global variables and selection of the peripherals has been completed, but is also entered if the Master Program has too many codewords. or contains a codeword having more than seven elements, or if a non-existent Brick is called for. In these latter cases, a new Master Program can be input if available.

On entry, FIRST is set to true , denoting that execution of the Master Program has not yet started. The program title is then read and copied to output, being terminated by the String £$, and the codeword serial number I is set to zero. For each codeword in turn, NEW is set to true (denoting the start of a new codeword) and the item number J is set to zero. For each item, the item sum K is set to zero.

As each character is read in (to the variable S), its internal integer code value is compared with those already stored in the array CD (corresponding to the digits 0 to 9, end-of-line, comma and right parenthesis respectively). If the character is not one of this set, it is ignored and the next character is read in.

On reading any of the digits 0 to 9, NEW is set to false (denoting that significant information has been read), and the new digit is added to ten times the previous partial item sum K to form a new partial item sum, and the next character is read.

On reading "newline" or "end of card" it is ignored if NEW is true (thereby permitting extra newlines between codewords), otherwise the item sum K is assigned to element CW [I, J] of the codeword array, the remaining elements (if any) of the codeword are put equal to zero, the codeword serial number I is increased by 1 and the next codeword is sought. This makes it unnecessary to incorporate extra zeros at the right-hand end of a codeword merely to make the number of items up to seven.

On reading "comma", K is assigned to CW [I, J], the item number J is increased by 1, and the next item is sought.

On reading a right parenthesis (indicating that the preceding integer is the entry codeword number K), control is transferred to Codeword Obey.

If the number of codewords exceeds 51 or if the number of elements in a codeword exceeds 7 (NOTE: this will occur if the seventh item is followed by a comma), an appropriate diagnostic message is printed, followed by the messages LOAD MORE CODEWORDS IF AVAILABLE on the output and HALTED:- 40 on the console typewriter. The computer then halts, permitting the operator to load and execute a further Turbocode Master Program if desired.

This latter sequence is also entered, after printing the message

BRICK n NOT AVAILABLE, if any non-existent brick (no. n) is called for.

A. 3. 2 Codeword Obey (See Flow Diagram in Fig. 7)

If this routine is entered from Codeword Input (at label OBEY), CHECK is set to true or false according as Sense Switch 1 is on or off. The elements of the codeword are then assigned to NEXT BRICK, A, B, C, D, E, F respectively, and provided that the Brick Number NEXT BRICK is in the range 1 to 40 inclusive, control is transferred by the switch designator BRICK [NEXT BRICK] to the required Brick. Otherwise, the error sequence for non-existent Bricks described in A. 3. 1 above is entered. However, Codeword Obey is normally entered after the previous Codeword has been obeyed. At the entry print NORMAL (used when the codeword to be obeyed follows immediately after that just obeyed), the current codeword serial number K is increased by 1. At the entry point JUMP (used when item F of the codeword just obeyed specifies a jump to a codeword other than the one next in sequence), F is assigned to K. In either case, if the new value of K is not in the range $0 \leqslant K < I$, where (I - 1) is the serial number of the last codeword, the illegal-program sequence is entered (see A. 3. 1 above): otherwise, Codeword Obey is continued as on initial entry from Codeword Input.

A. 3. 3 Error Tracing Facilities

A. 3. 3. 1 Trace Printing during Normal Running

At first entry to the Scheme at the start of a new Master Program, and thereafter at every entry to Codeword Obey, the Boolean variable CHECK is set to true or false according as Sense Switch 1 is currently ON or OFF - the operator can change this setting at any time. While CHECK is true, at every entry to each Brick, an abbreviated title of that Brick will be printed out. This facility is very useful in tracing the course of a calculation, especially when a program error is suspected, since the diagnostic facilities built into the Algol compiler relate to the Algol source program - whose details are not normally known to the user - rather than to the Turbocode Scheme as such, and are therefore difficult to interpret unless the user is thoroughly familiar with the details of the Scheme. However, such printing naturally slows up the calculation very markedly, and so should be used sparingly.

A. 3. 3. 2 Printing of Partial Results in the Event of Program Failure

If a program stops unaccountably, or if it appears to have entered a loop, the operator can halt the program, put Sense Switch 2 ON and re-enter the Scheme from the beginning. This will cause the Boolean variable SENSE to be set to true, resulting in entry to the penultimate codeword of the Master program, which is assumed to correspond to Brick 31 (Station Vector Output). After obeying this codeword, the final codeword is obeyed - assumed to correspond to Brick 32 (Engine Vector Output and Station Vector Reset) - after which the program is halted with the message HALTED:- 32 output to the console typewriter. At this point the operator

can suspend the program, the programmer now having a full printout of all results calculated up to the point of failure, which helps to locate the error: alternatively, the calculation of the next data point can be attempted.

A. 3. 4 Suppression of Station Vector Output

The output of the full Station Vectors is relatively time-consuming, and frequently only the flow areas within the cycle are of interest. Provision is therefore made within Brick 31 to suppress Station Vector Output, by use of Sense Switch 3. The Boolean variable SUPPRESS is set to true or false according as this sense switch is on or off: suppression occurs when SUPPRESS is true , when Station Vector printing is restricted to the Station Vector numbers and areas of these stations whose areas have been computed. This facility can be switched on or off by the operator at any time.

Appendix 4 - Improvements in Certain Subroutines and Bricks

Since the original "Pegasus" version of the Scheme was developed over a considerable period, and since a number of Subroutines and Bricks were produced at short notice for particular problems, a considerable number of storage- and time-consuming redundancies had crept into the coding. In the course of re-coding the Scheme, the opportunity has been taken to eliminate these redendancies as far as possible. This has been done in three main ways:-

A. 4. 1 Elimination of Superfluous Subroutines

In the transition from a low-level to a high level language, certain of the Subroutines degenerated into single Algol statements, which no longer justified writing them as separate procedures. The calls of these Subroutine procedures have therefore been replaced by their single-statement procedure bodies: the Subroutines thus eliminated are:-

S1 (calculates Δh from α_{in}, t_{in} and t_{out})

S4 (calculates p_{out}/p_{in} (isentropic) from α_{in}, t_{in} and t_{out})

S7 (calculates V from W, p, t and A)

S14 (calculates A from W, p, t and V)

S15 (calculates W from p, t, V and A)

A. 4. 2 Rearrangement of Other Subroutines

In the earlier Scheme, Subroutines S11 (calculates critical p, t, V and A from α, W, P and T), S12 (calculates supersonic p, t and V from α, W, P, T and A) and S17 (as S12, but subsonic) formed a single sequence of instructions with alternative entry points: S11 was used by both S12 and S17 to check that the given area is not less than the critical area, in which case no solution is possible and the given mass flow requires scaling down.

The block structure of Algol does not permit alternative entry points to a given procedure body, but this difficulty has been circumvented by using the single procedure S11 1217 (A, B, TYPE) in which the parameter A specifies the relevant Station Vector, B specifies the highest numbered Station Vector whose mass flow value requires scaling if the given area is inadequate, and TYPE indicates whether the erstwhile S11 (TYPE = 0), S12 (TYPE = +1) or S17 (TYPE = -1) is required. In addition, Brick 25 used S12 and S17 in circumstances in which the given area is known to exceed the critical value, so that the S11 area check and associated mass flow scaling is unnecessary. This is achieved by setting TYPE equal to +2 for S12 and to -2 for S17.

The other alteration in this category is the introduction of a new Subroutine called S10 (A, B, C), which solves the frictionless parallel-flow momentum equation for two inlet streams (Station Vectors A and B) and a

single outlet stream (Station Vector C). If there is only a single inlet stream, B is given a negative value (usually -1). The introduction of this Subroutine materially shortens Bricks 5 (Mixing), 6 (Constant Pressure Heating with Fundamental Pressure Loss) and 25 (Off-Design Convergent-Divergent Nozzle).

A. 4. 3 Grouping of Bricks with Common Features

A number of Bricks perform closely similar tasks, and consequently have a substantial part of their coding in common. Advantage was taken of this in the original Scheme to use a common sequence of instructions for Bricks 10 (Compression, given Temperature Rise) and 12 (Compression, given Work Input). This process has now been carried further by including Brick 2 (Compression, given Pressure Ratio) in the same group, and also the rather unnecessary Brick 3 (Calculate Work Done). This reorganisation has been carried out in such a manner that programs calling specifically for Brick 3 can still be used, provided that the codeword for Brick 3 immediately follows that for the associated Brick 2.

Pursuing this policy further, Bricks 8 (Optimum Convergent Nozzle) and 23 (Optimum Convergent-Divergent Nozzle) have been coalesced. Bricks 6 (Constant Pressure Heating with Fundamental Loss), 29 (Constant Pressure Heating without Fundamental Loss) and 39 (Constant Volume Heating) have also been combined, and since Bricks 6 and 29 were the only ones using the Function FUEL, its statements have been incorporated into the new combined Brick, and it no longer exists as a separate Function.

Appendix 5 - Requirements for New Bricks

The structure of the Turbocode Scheme is such that it is a relatively simple matter for anyone acquainted with Algol programming to write new Bricks. In so doing, it is important to bear the following points in mind:-

1. The Codeword system described in Section 2 of the main body of the Report must be rigidly adhered to, and as far as possible the standard meanings of the various Codeword elements should be retained. Where the Brick Data and Engine Vector requirements cannot be fitted into a single Codeword, Brick 26 (Auxiliary Codeword Brick) can be employed to provide extra Codeword elements, as is already done in the cases of Bricks 27 and 34.

2. The available Global Variables, Functions and Subroutines are described in Appendix 2.

3. A Brick should be written as a Block, bearing a label of the type Bn: (where n must obviously be chosen from among the Brick Numbers not already in use). This same label number must also be deleted from those at the head of the Error Sequence.

4. If all available spare Brick Numbers have been used up, further numbers beyond 40 may be employed. In this case, the requisite extra elements must be added to the switch BRICK at the start of the Scheme and the statement

```
'IF' NEXT BRICK 'LE' 40 'AND' NEXT BRICK > 0 'THEN'
'GO TO' BRICK [NEXT BRICK ] 'ELSE' 'GO TO' B7
```

   in Codeword Obey must be amended accordingly.

5. To preserve the checking facility, the first statement of the Brick should be

```
'IF' CHECK 'THEN' WRITE TEXT ('(' 2C') 'Bn')').
```

6. Normally, exit from a Brick is to the label NORMAL, but a jump exit (to a Codeword specified by element f of the Codeword of the current Brick) may be employed, exiting to the label JUMP.

Appendix 6 - Notes on the Flow Diagrams

The previous Report (ref. 1) did not give flow diagrams of the various elements of the Scheme. This omission has now been remedied in Figures 1 - 24.

The following points should be noted:-

1. The structure of certain parts of the Scheme is so simple that the provision of flow diagrams for them is unnecessary. This applies to the Functions SPHT, ENTH and PRES, the Subroutine S9 and the Bricks 1, 5, 9, 24, 26, 33 and 40.

2. In the interests of brevity, the counting and testing operations associated with for statements (except those incorporating while elements) are not given in full: instead the box enclosing the controlled statement(s) shows a path looping back to its entry point, annotated with the name and range of values of the controlled variable in the form I = 1 (1) 50. In most cases, a single exit path is shown from the box, which is followed when the for list is exhausted, but where a go to statement is included among the controlled statements, two exit paths are given, annotated "go to" and "exh" respectively.

3. As an aid to cross-references between the listing and the flow diagrams, all labels occurring are written alongside the box to which they refer.

4. Since the Bricks all use similar sequences, controlled by the Boolean variable CHECK, to activate trace printing of the name of the Brick concerned, this sequence is omitted from the flow diagrams.

5. Similarly, many Bricks end by employing Subroutine S16 to calculate static conditions and area from the total conditions, if the velocity is given, or Subroutine S11 1217 to calculate static conditions and (subsonic) velocity from the total conditions, if the area is given. This sequence is also omitted from most of the flow diagrams.

6. The following abbreviated notation is employed:-

Notation in Listing	Notation in Flow Diagrams
ALPHA [A], XALPHA [A]	α_A, $X\alpha_A$
W [A], XW [A]	W_A, XW_A
PSTATIC [A], XPSTATIC [A]	p_A, Xp_A
PTOTAL [A], XPTOTAL [A]	P_A, XP_A
TSTATIC [A], XTSTATIC [A]	t_A, Xt_A
TTOTAL [A], XTTOTAL [A]	T_A, XT_A

Notation in Listing contd.	Notation in Flow Diagrams contd.
VELOCITY [A], XVELOCITY [A]	V_A, XV_A
AREA [A], XAREA [A]	A_A, XA_A
GAMMA	γ
TWO GCJ	$2g_cJ$
GC	g_c
SPHT (ALPHA, T)	$C_p(\alpha, t)$
ENTH (ALPHA, T)	$h(\alpha, t)$
PRES (ALPHA, T)	$\pi(\alpha, t)$
SQRT (X)	$\sqrt{X}$
LN (X)	$\log_e X$
EXP (X)	exp X
ABS (X)	$\lvert X \rvert$
X ↑ Y	X^Y
X * Y	XxY
1400.69	J
4633.056	144xgc
0.0685582	R/J
0.666808	R/144
2.54671	J/550
BRICK DATA [C]	BD_C
ENGINE VECTOR [D]	EV_D

Appendix 7 - Compilation and Execution Procedure for I. C. T. 1900 Series Computers

A. 7. 1 Compilation

In the I. C. T. 1900 Series implementation of Algol, of which # XALP is the paper tape version and # XALM the magnetic tape version, compilation occurs in two stages. In the first stage a complete program, or segments of it, is read from punched cards (entry point 20) or from 8-channel punched paper tape (entry point 21). If no errors are detected, a "semi-compiled" output tape is produced, followed by a "general purpose loader" tape. As compilation proceeds, a full or abbreviated listing of the Algol program may be output on a line printer or tape punch, in which each statement is allocated a number. If a syntax error is detected, an appropriate diagnostic message is output, immediately following the erroneous statement, while diagnostic messages relating to semantic errors are output at the end of the input phase. Once an error has been detected, compilation ceases, but the source program continues to be read until twenty errors have been found; if this should occur, the program is too much at fault to justify further compiler action and the attempted compilation is abandoned. The listing terminates with the symbol EC if compilation has been completed (preceded by a statement of the core storage requirement), or with ZZ if an error has been found.

On successful completion of the first stage of compilation, the second (consolidation) stage can take place. The general purpose loader is input, followed by the semi-compiled tape(s) of the program and/or segments, followed by the Algol library tape, from which those procedures which the program has called for are read. When this stage has been completed the program is "consolidated" into a viable object program, and it is possible to dump the entire object program on to paper tape or magnetics: it is obviously desirable to produce such a "binary dump" for any program which is to be used repeatedly, since much time and effort is thereby saved on program input for subsequent runs.

Compilation from other source languages, such as Fortran, Extended Mercury Autocode or PLAN, is carried out in a similar manner, and results in semi-compiled output which is of the same form in all cases: thus, subject to certain compatibility restrictions (see ref. 3), semi-compiled segments originating in different source languages can be consolidated into a single object program.

In the case of Turbocode, only the binary dump version is normally used for day-to-day running. A master source program is maintained on punched cards, however, and in the event of any alteration or addition to the Scheme cards are altered, added or removed and a new compilation is made. Where segmentation is employed, those segments which have not been altered can be retained in semi-compiled form, and incorporated with the new material during the consolidation stage.

A. 7. 2 - Execution

Once consolidation has taken place, or the binary dump version of a program has been loaded, the method of execution is the same regardless of the original source language: the operator types the message GO # (Program Name) (Number) to initiate program execution at the entry point designated by the Number. In Turbocode, the Program Name is R 001, and the normal entry point number is 20, causing the program to be obeyed from the start. If a run stops in the middle of the program for any reason, a similar message omitting the entry point number causes the program to continue from the point at which it stopped.

In interpreting the following operating instructions for Turbocode, it is necessary to remember that

a) the Master Program is viewed by the Scheme as a species of data;

b) the Master Program must have a title, terminated by the string £$;

c) the Data proper must follow the Master Program, and must have a title, terminated by £$;

d) every tape must terminate with the TC_4 (End of Tape) character, to cause input to stop. If cards are used, input will stop automatically when the card pack is exhausted.

In order to simplify the input process, and bearing in mind that a Master Program consists of 51 seven-element codewords at most, it has been found desirable to punch a copy of the Master Program, with its title, at the head of each Data tape.

A. 7. 3 - Operating Instructions for the Turbocode Scheme

A. Normal Operation, or if Computer Stops "HALTED:- EE" (Execution Error)

If several Master Program/Data Tapes are to be processed during the same run, the computer must always be restarted from the beginning of the Scheme.

B. If Computer Stops "HALTED:- 40" (due to excessive number of Codewords, excessive number of elements in a Codeword, or to a call for a Non-Existent Brick)

Load next Master Program/Data Tape and proceed as at A above.

C. If a Loop Stop or other Failure occurs, not leading to "HALTED:- EE" or "HALTED:- 40"

(1) Stop Computer

(2) Put Sense Switch 2 on to activate special diagnostic printing

(see Appendix 3, Section A. 3. 3. 2)

(3) Restart computer from the beginning of the Scheme without moving the Data Tape.

(4) When diagnostic printing ends with "HALTED:- 32", put Sense Switch 2 off.

(5) Load next Master Program/Data Tape and proceed as at A above.

D. To Pick up in the Middle of a Data Tape

(1) Restart computer from the beginning of the Scheme

(2) As soon as output starts, stop computer

(3) Move tape along to desired point

(4) Restart computer from the point of interruption of the Scheme

E. Other Sense Switch Settings

These are used only when specifically called for:-

(1) Sense Switch 1 causes special trace printing while on (see Appendix 3, Section A. 3. 3. 1).

(2) Sense Switch 3 suppresses most of the Station Vector output while on (see Appendix 3, Section A. 3. 4).

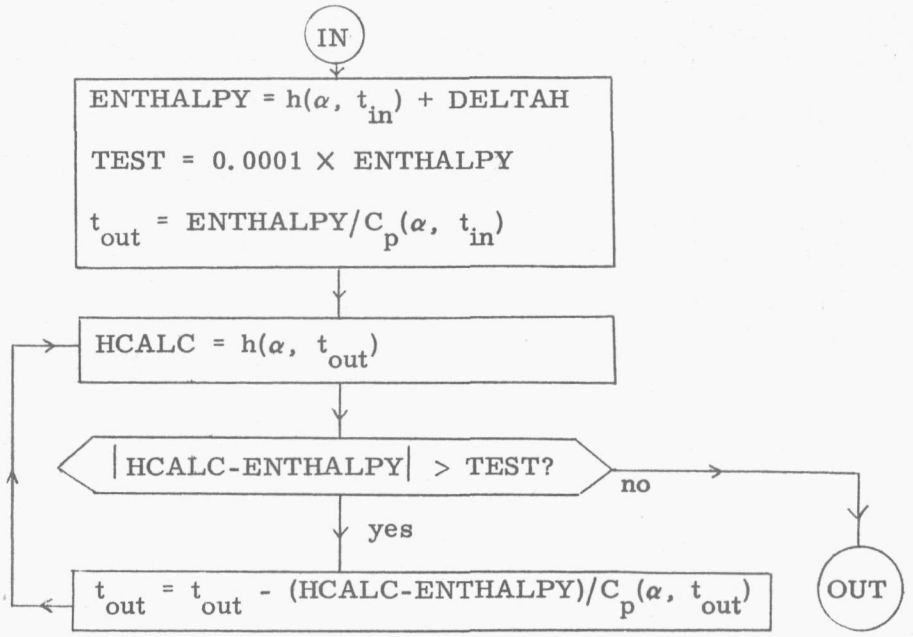

FIGURE 1 - Subroutine S2 (Find Temperature from Enthalpy Change)

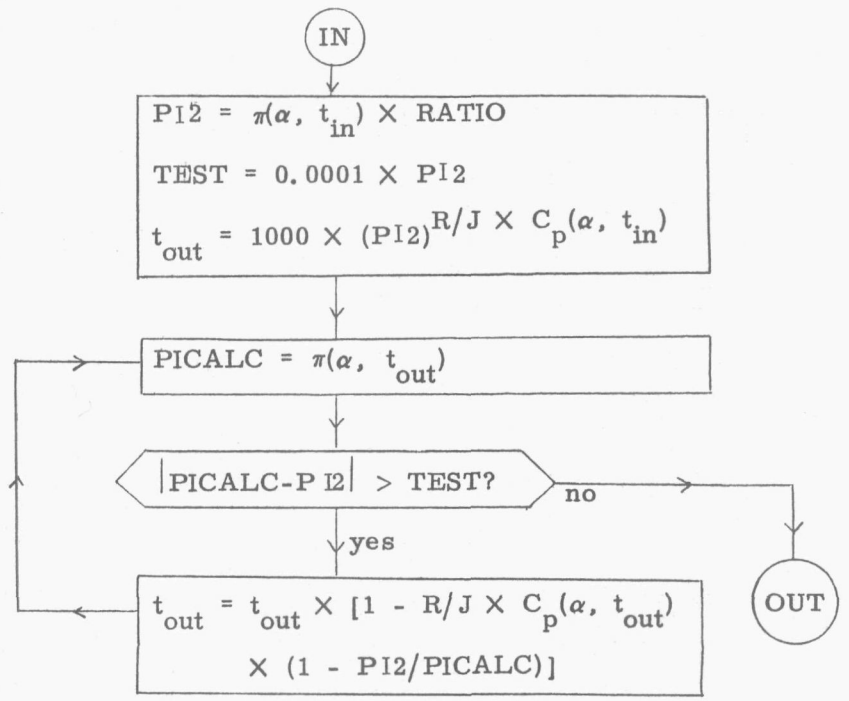

FIGURE 2 - Subroutine S5 (Find Temperature from Isentropic Pressure Ratio)

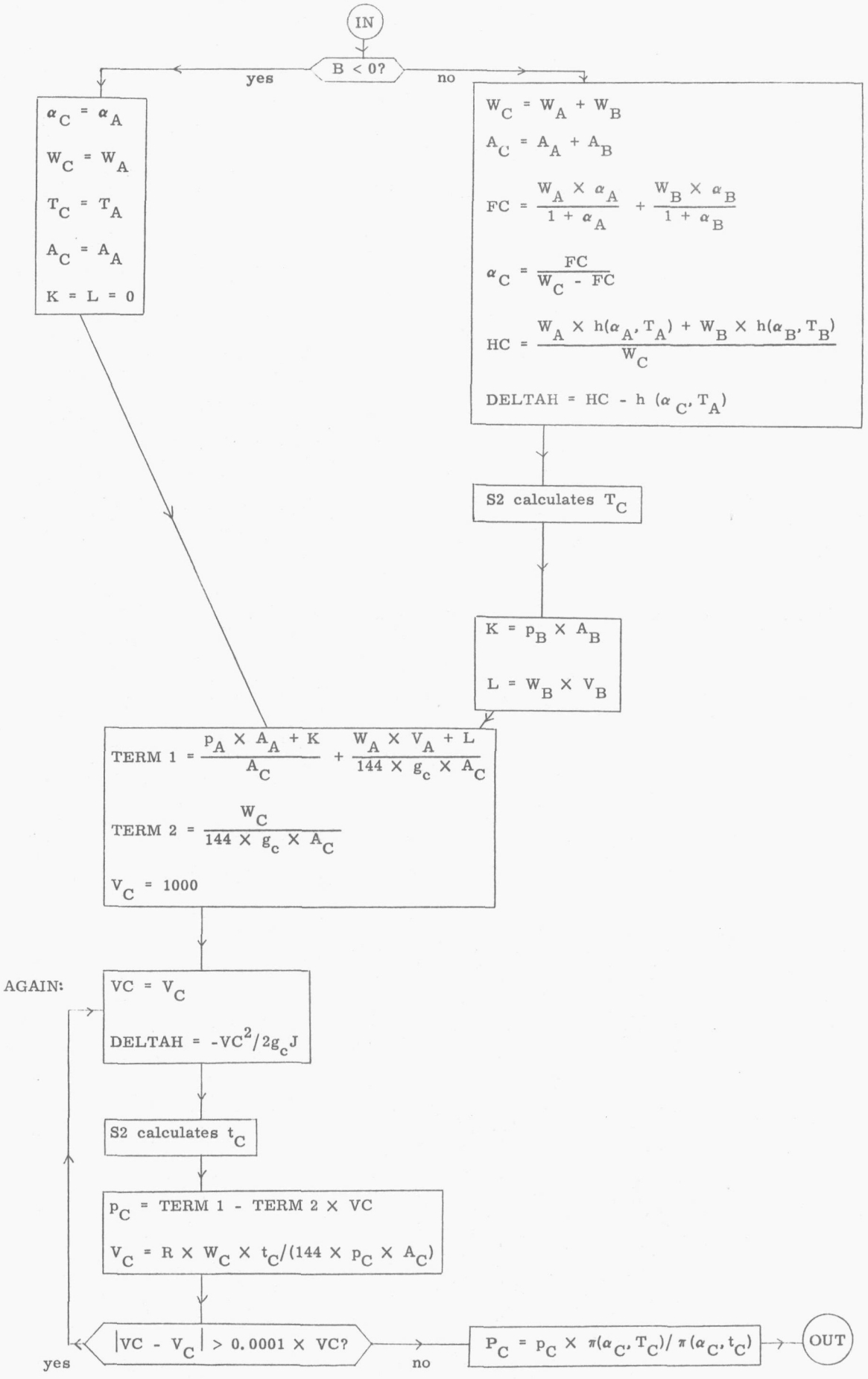

FIGURE 3 - Subroutine 10 (Solve Momentum Equation for Frictionless Parallel Flow)

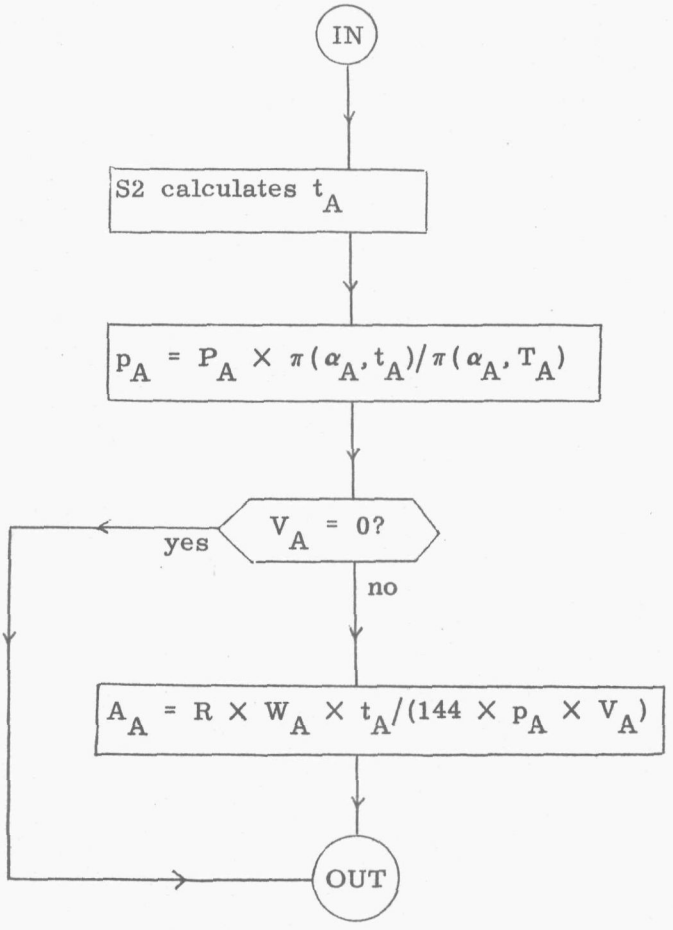

FIGURE 4 - Subroutine S16 (Find Static Conditions and Area, given Total Conditions and Velocity)

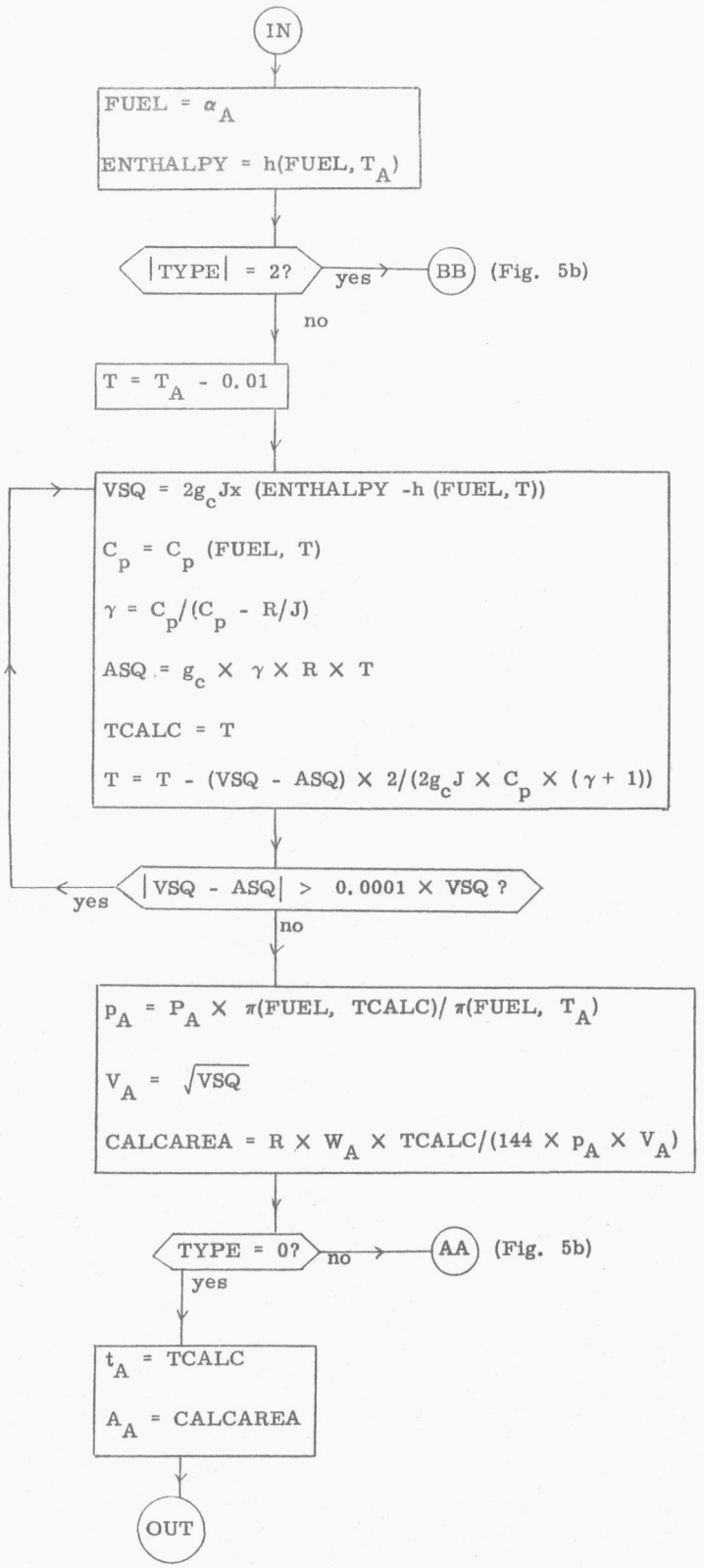

FIGURE 5a - Subroutine S111217 (Find Critical Conditions and/or p, t and V for given α, W, P, T and A)

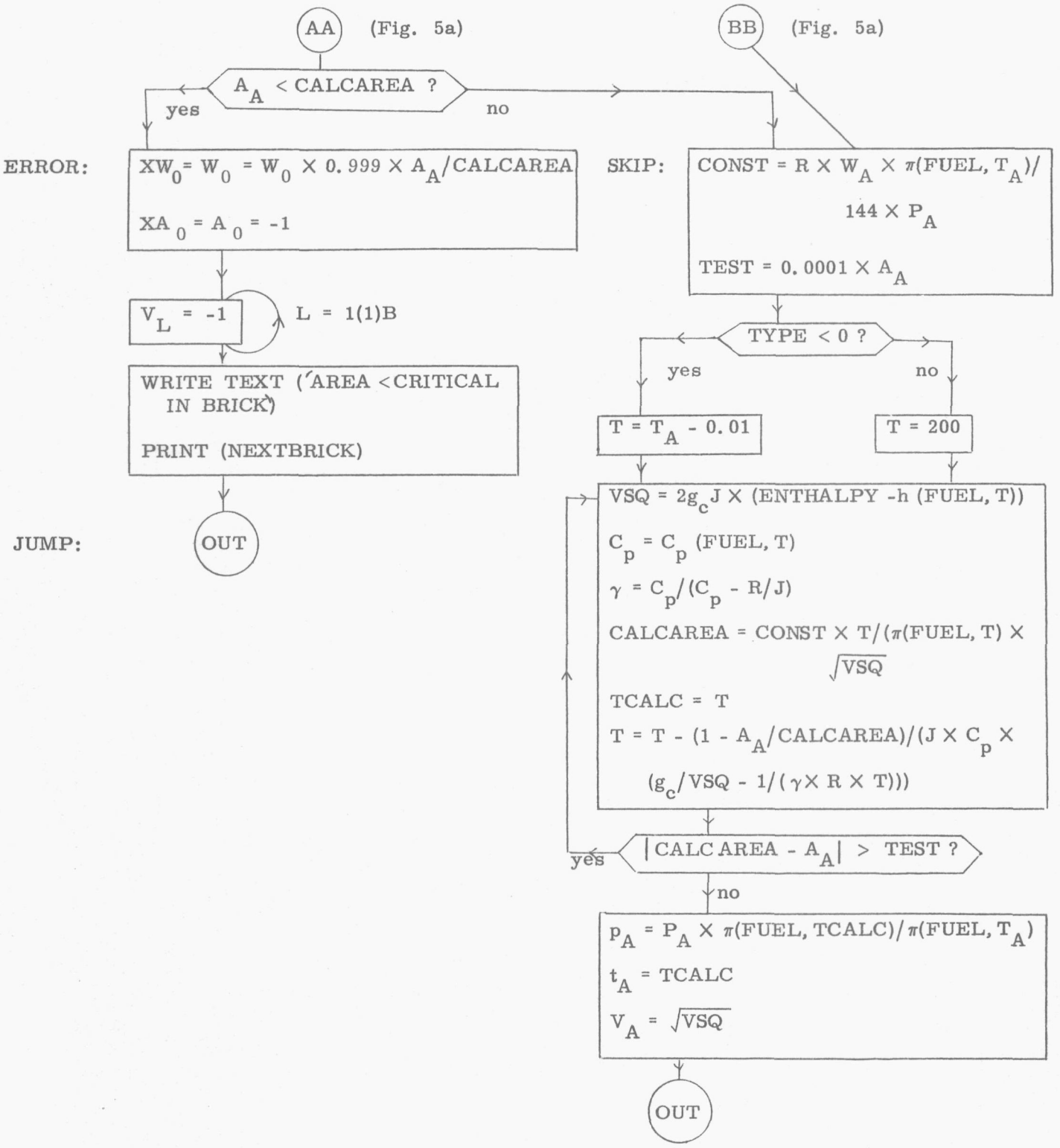

FIGURE 5b - Subroutine S111217 (contd.)

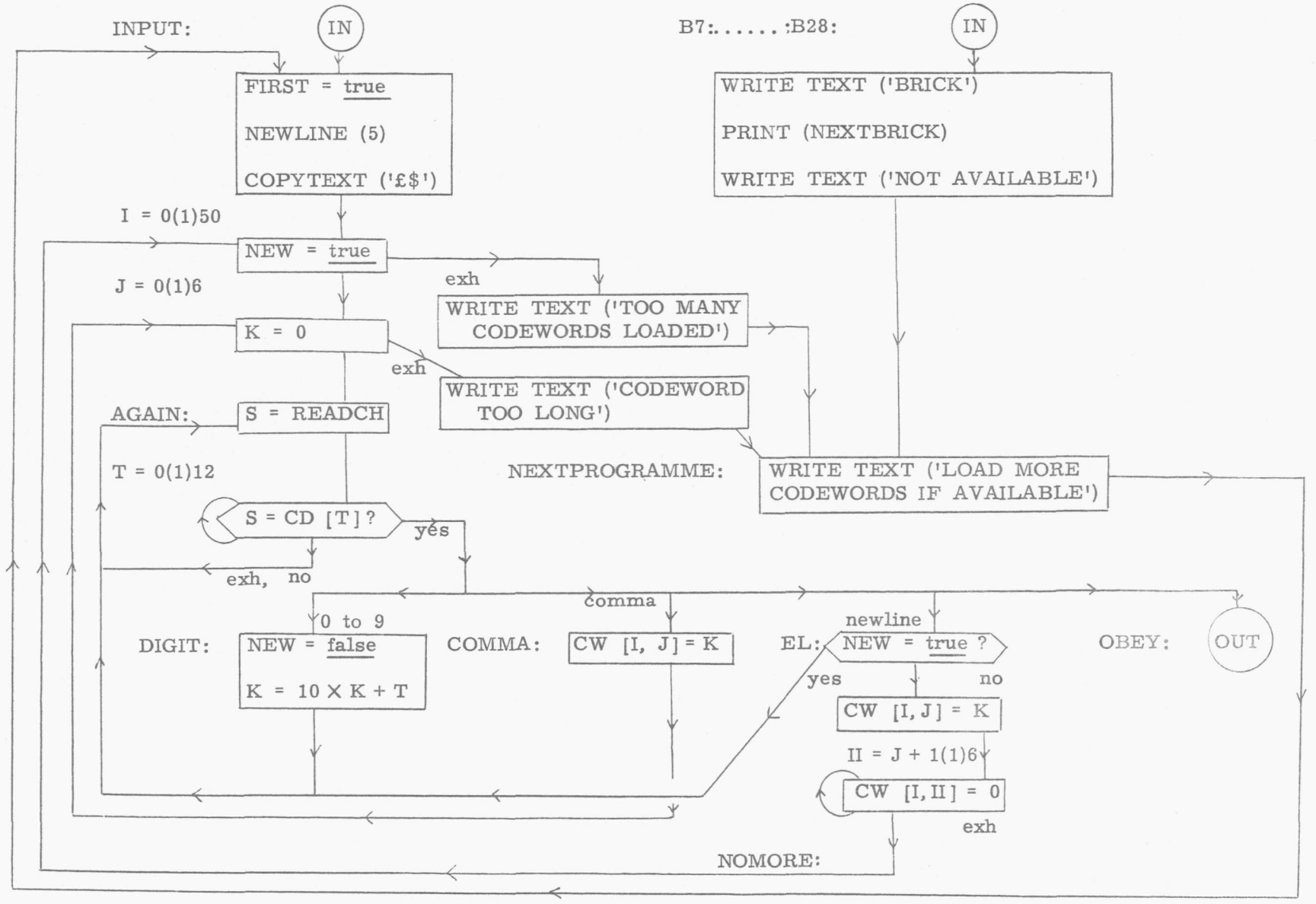

FIGURE 6 - Codeword Input, with Error Sequence

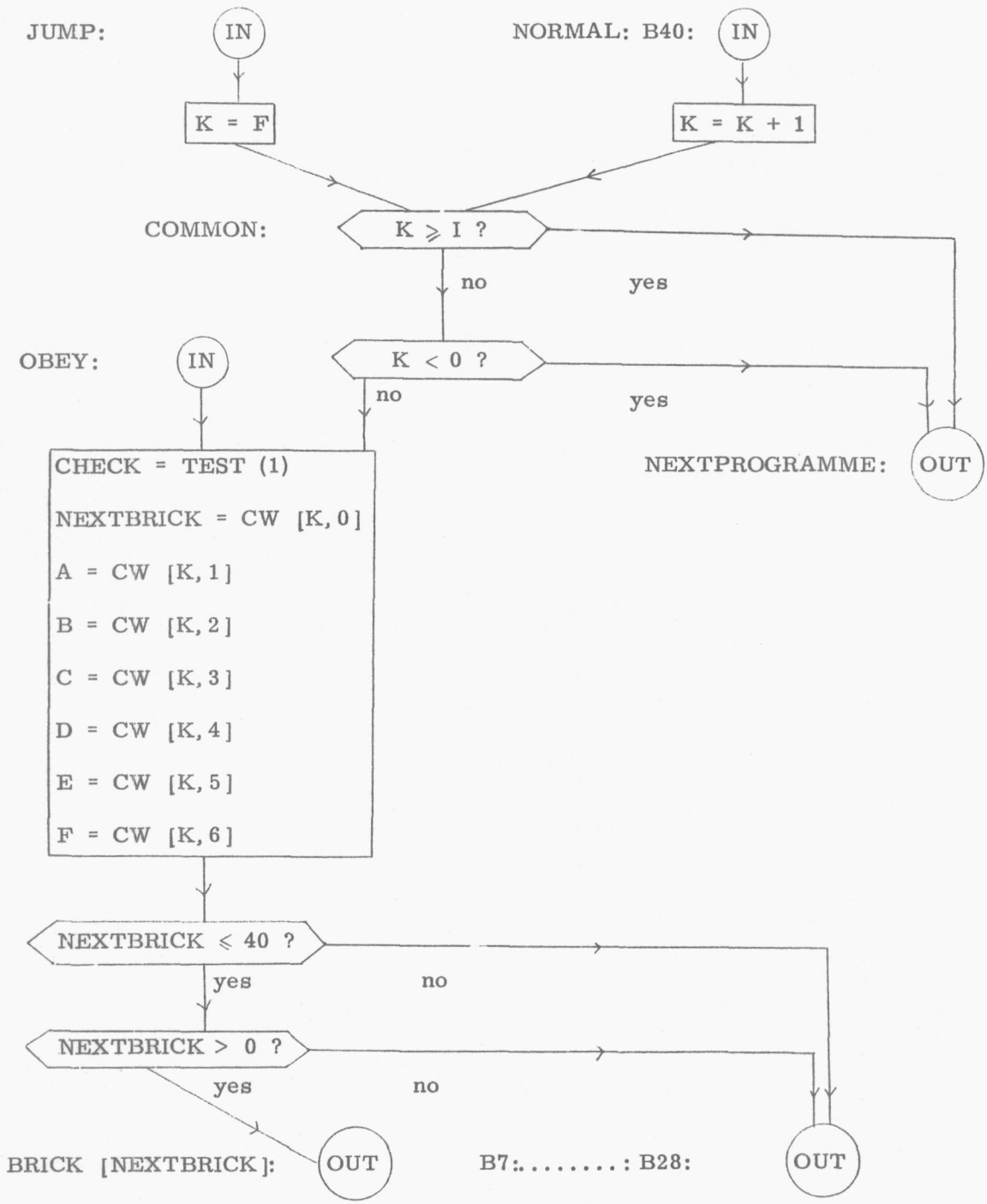

FIGURE 7 - Codeword Obey

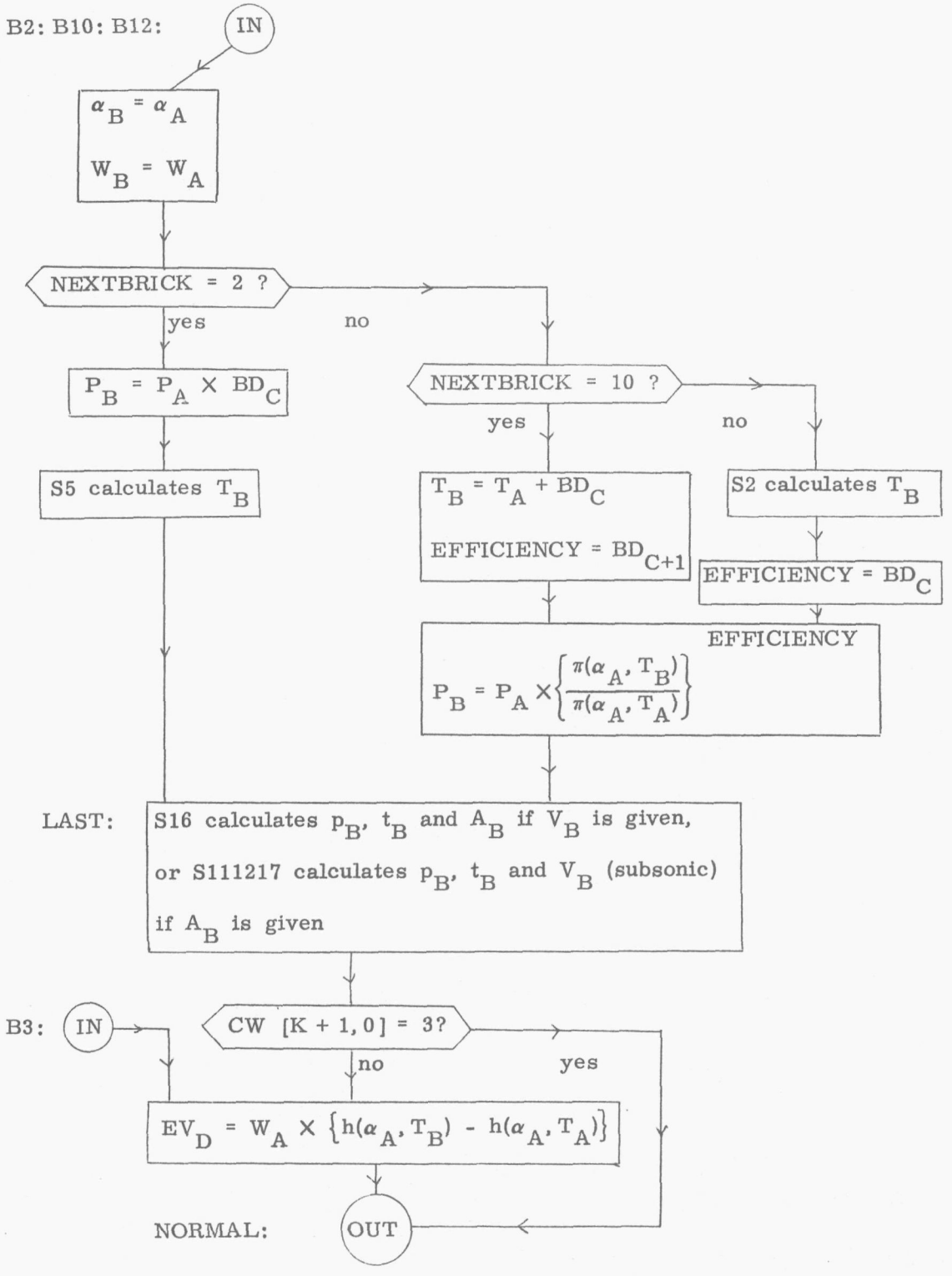

FIGURE 8 - Bricks 2, 10 and 12 (Compression) and 3 (Work Done)

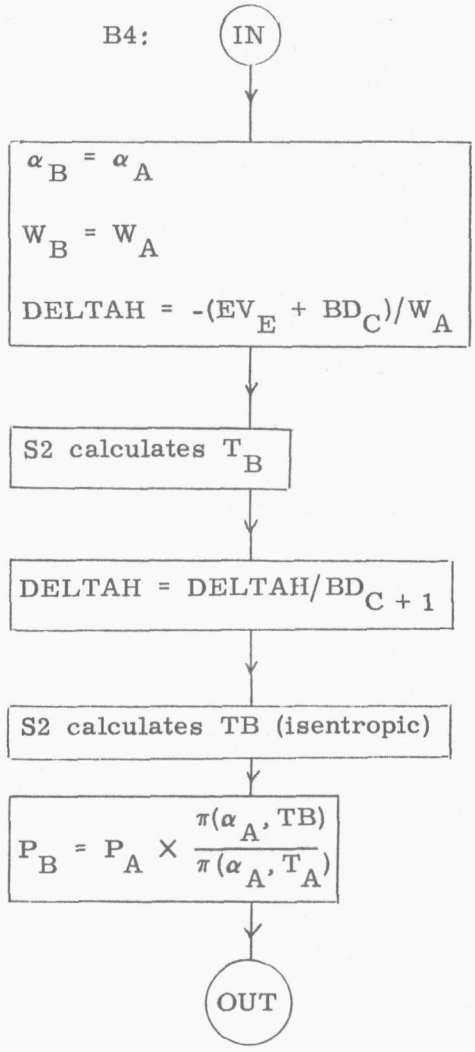

FIGURE 9 - Brick 4 (Single Turbine)

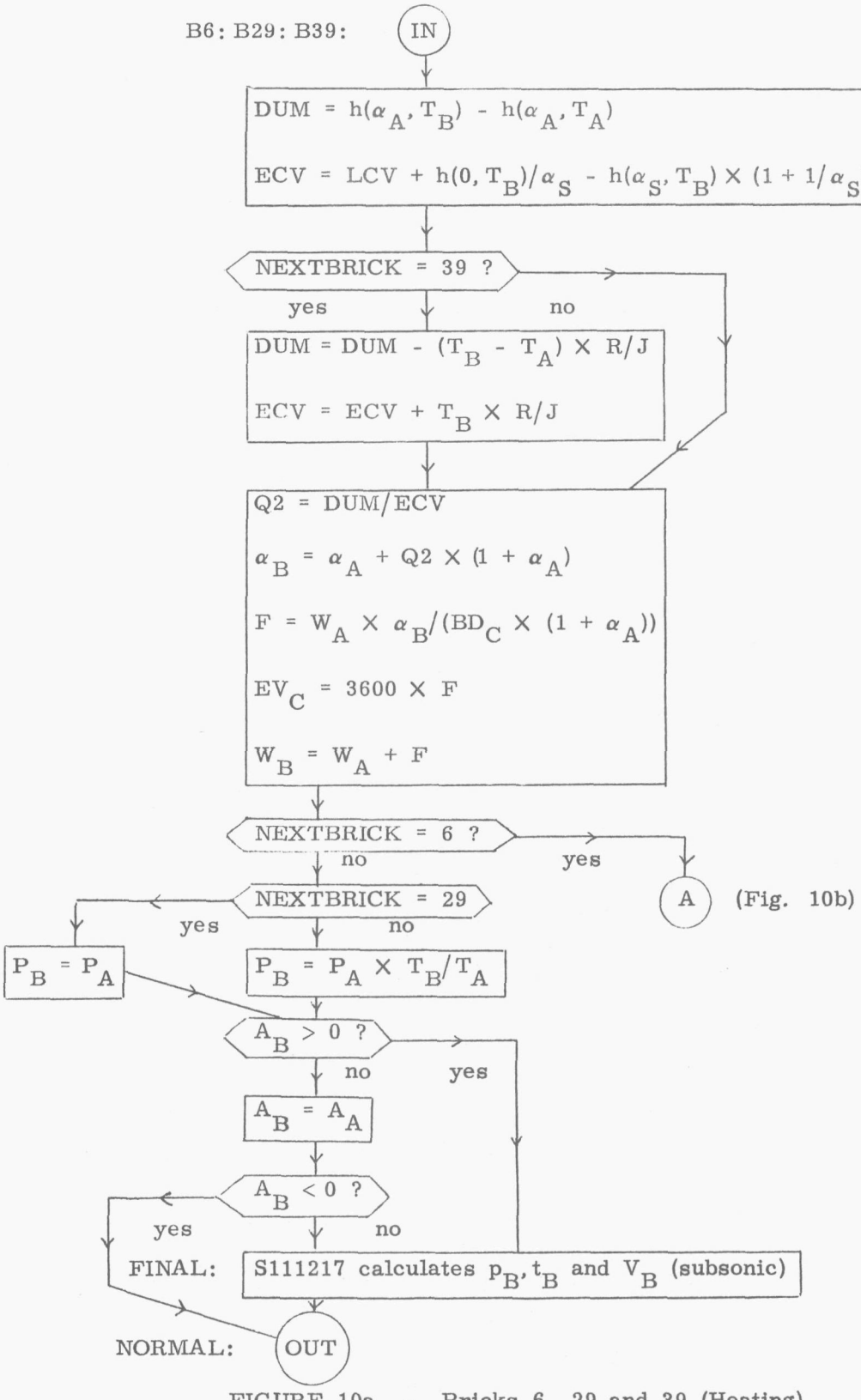

FIGURE 10a - Bricks 6, 29 and 39 (Heating)

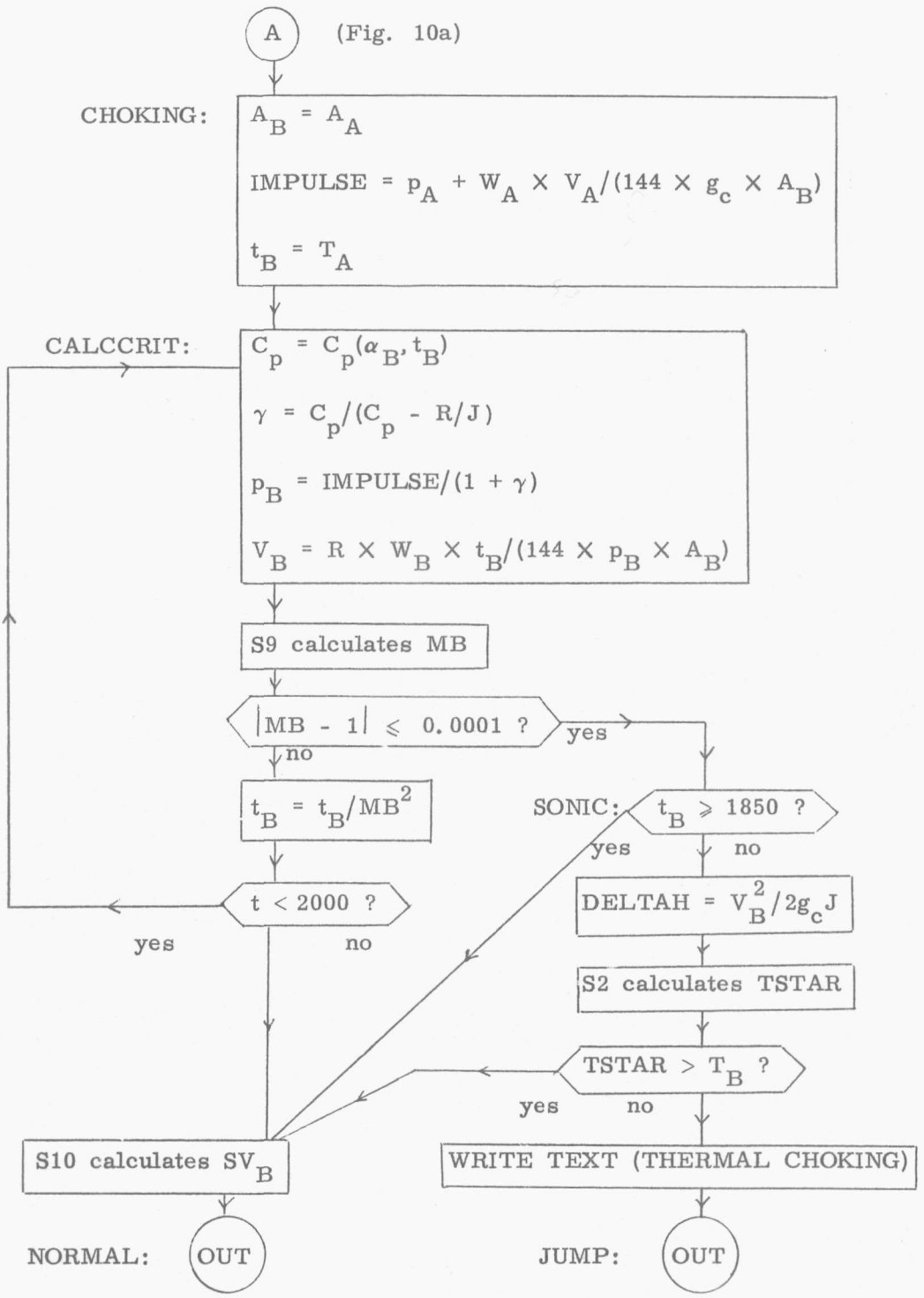

FIGURE 10b - Bricks 6, 29 and 39 (contd.)

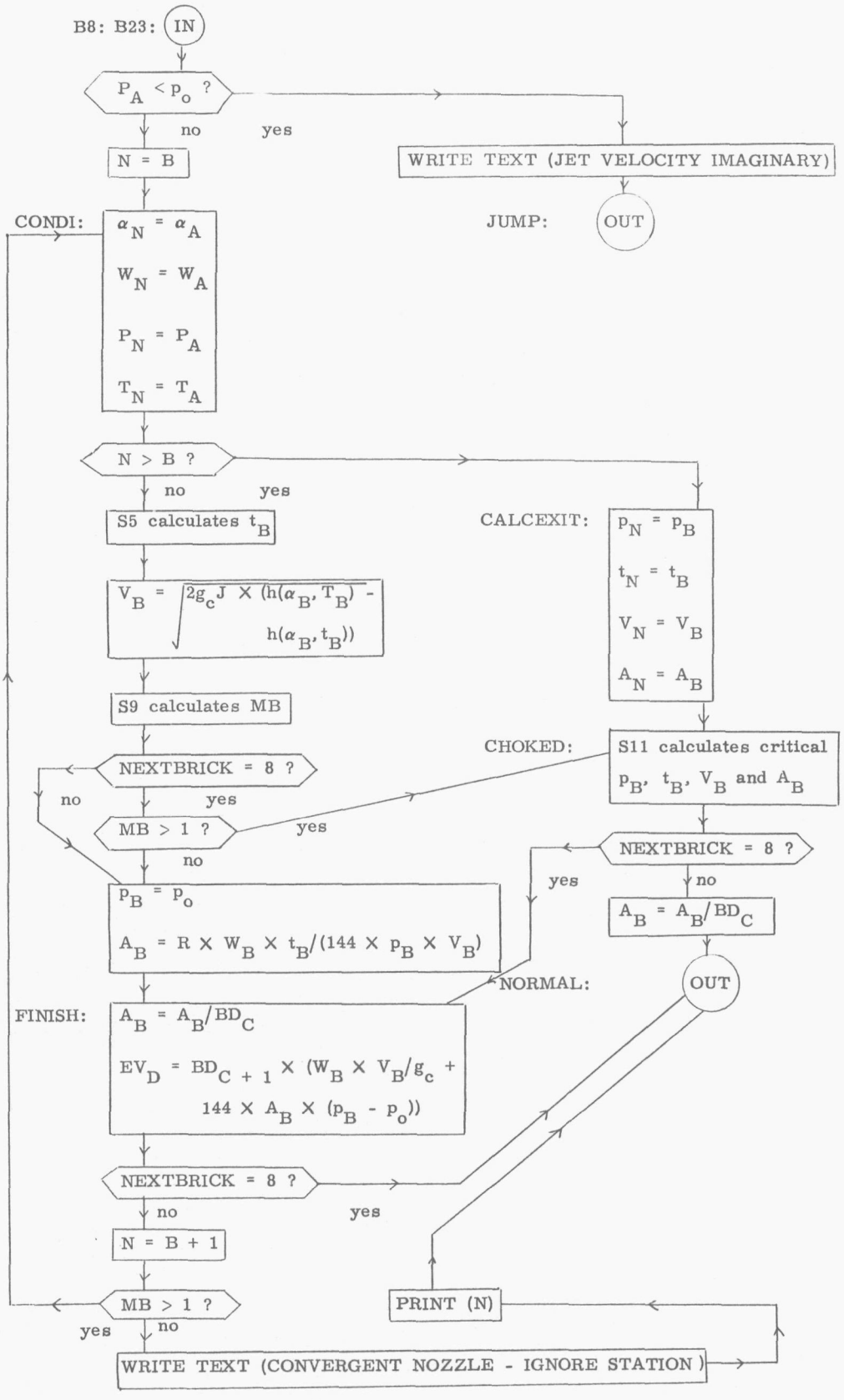

FIGURE 11 - Bricks 8 (Convergent Nozzle) and 23 Con-Di Nozzle)

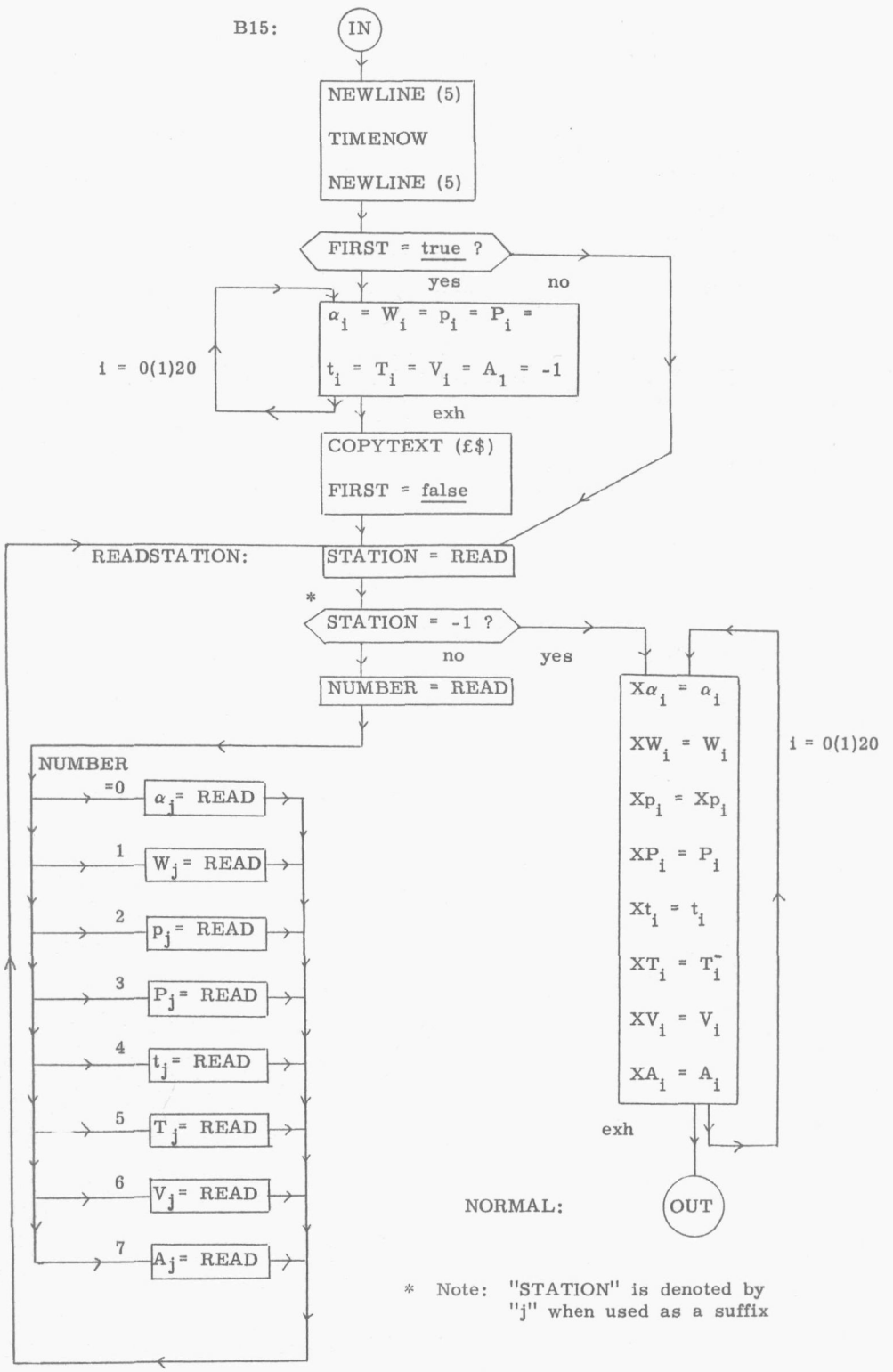

FIGURE 12 - Brick 15 (Station Vector Input)

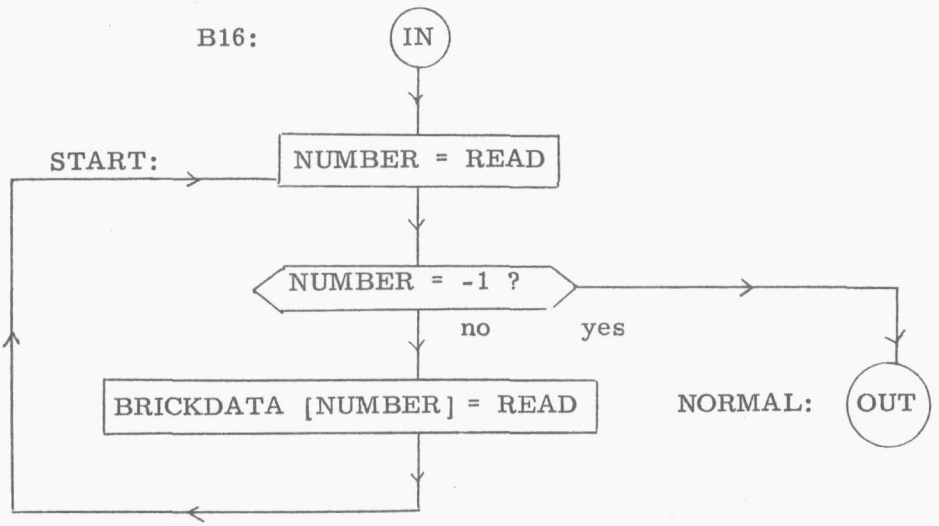

FIGURE 13 - Brick 16 (Engine Vector Input)

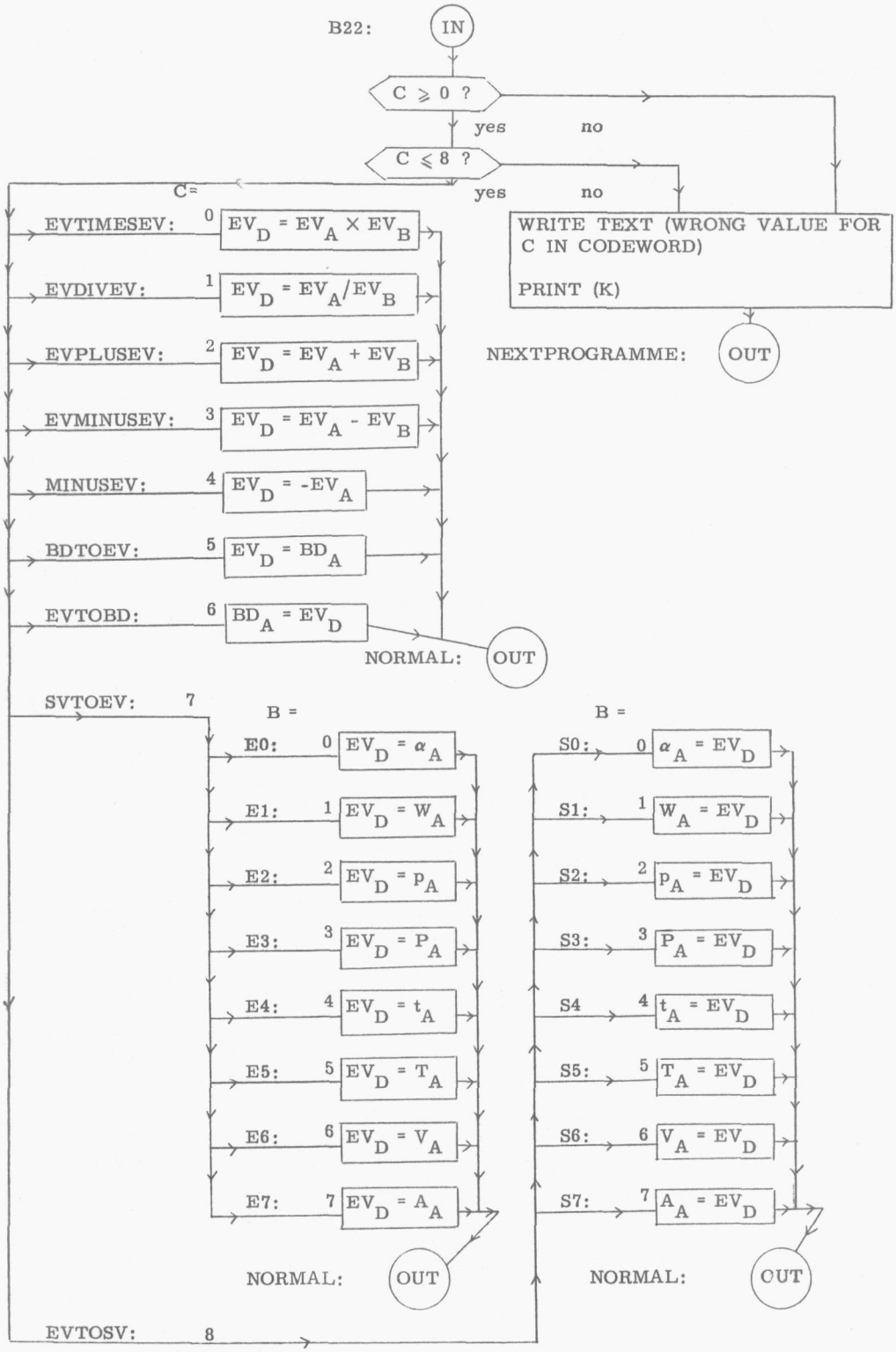

FIGURE 14 - Brick 22 (Arithmetic on Engine Vector, Station Vectors and Brick Data)

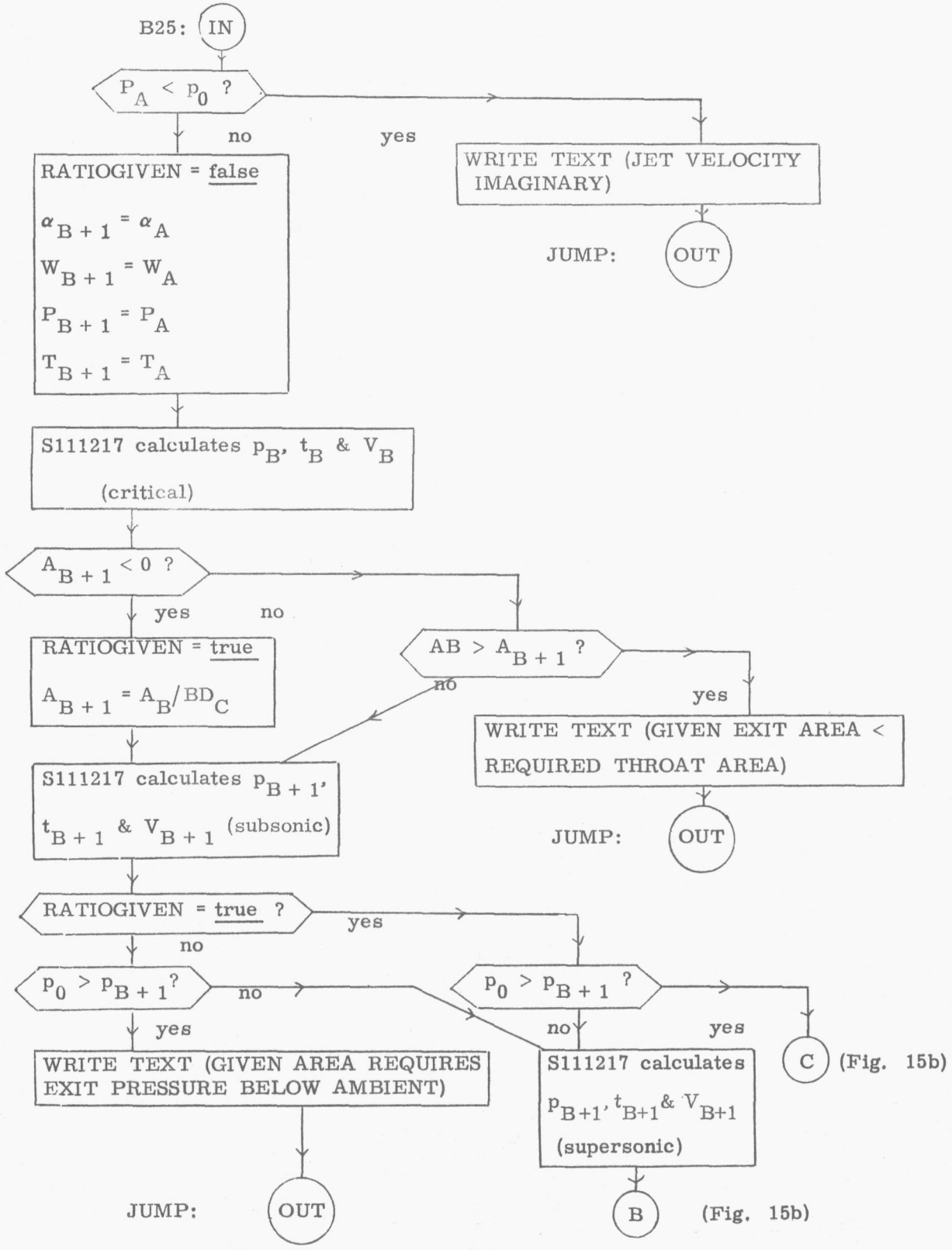

FIGURE 15a - Brick 25 (Off-Design Convergent-Divergent Nozzle)

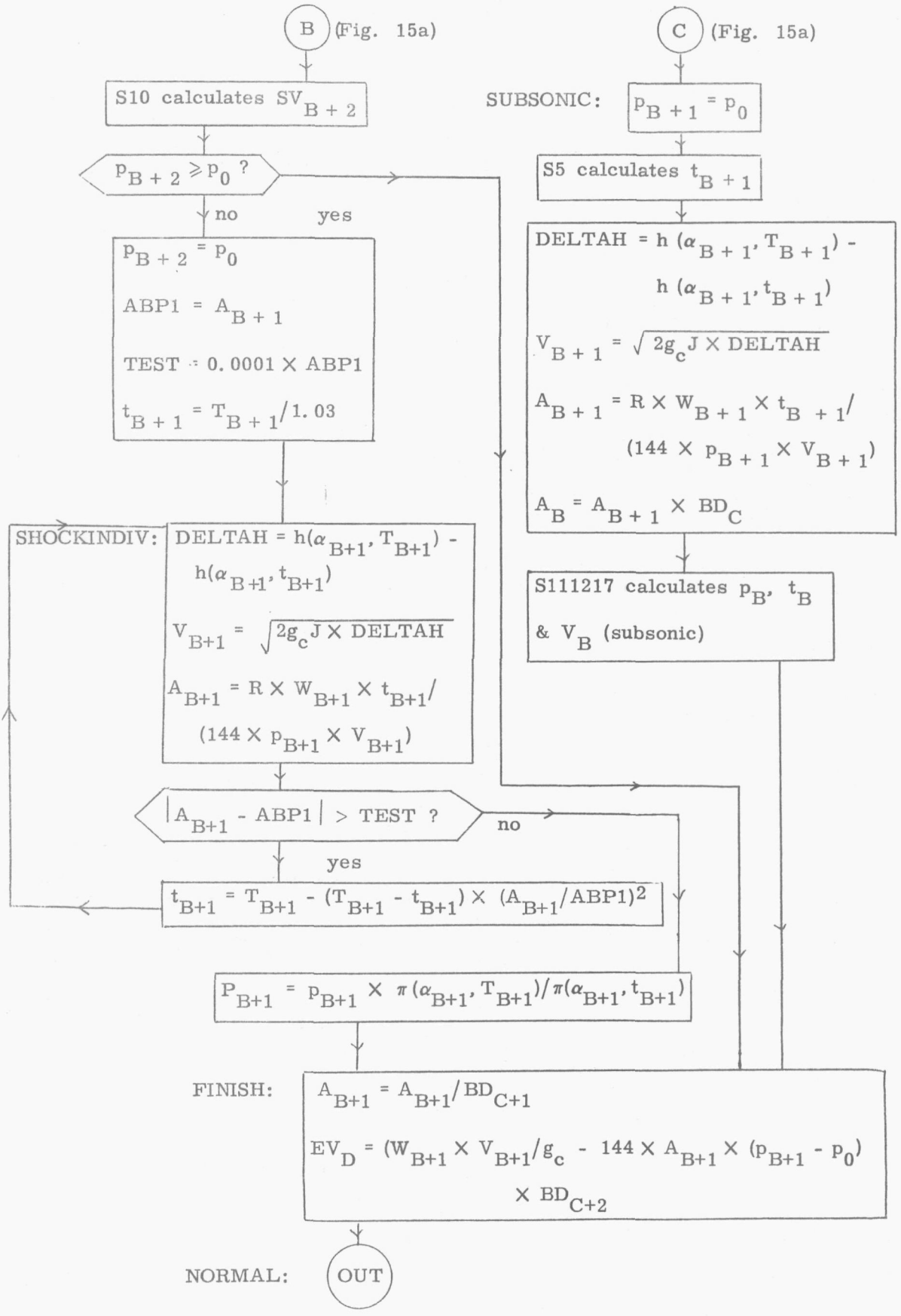

FIGURE 15b - Brick 25 (continued)

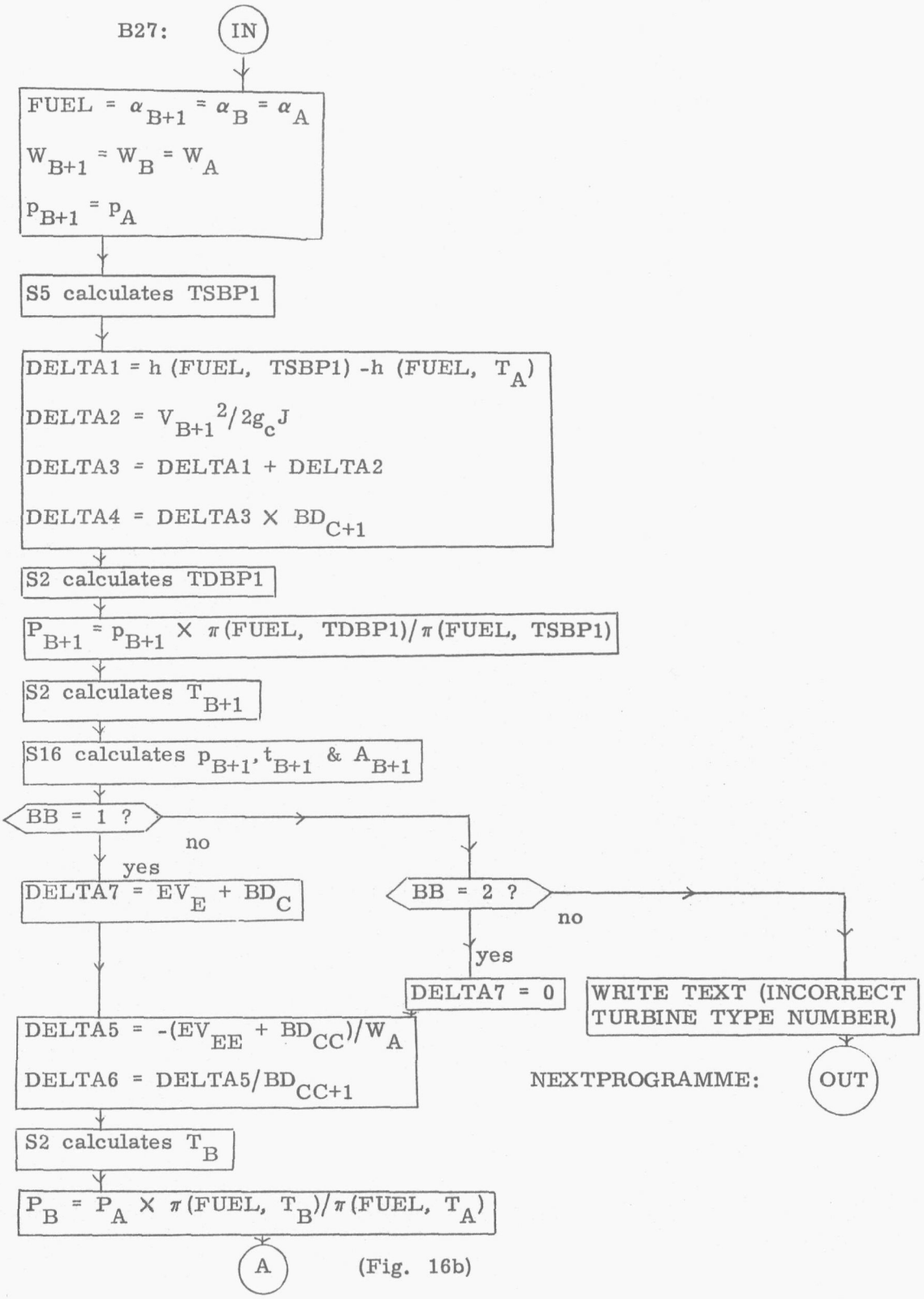

FIGURE 16a - Brick 27 (Two Turbines in Series)

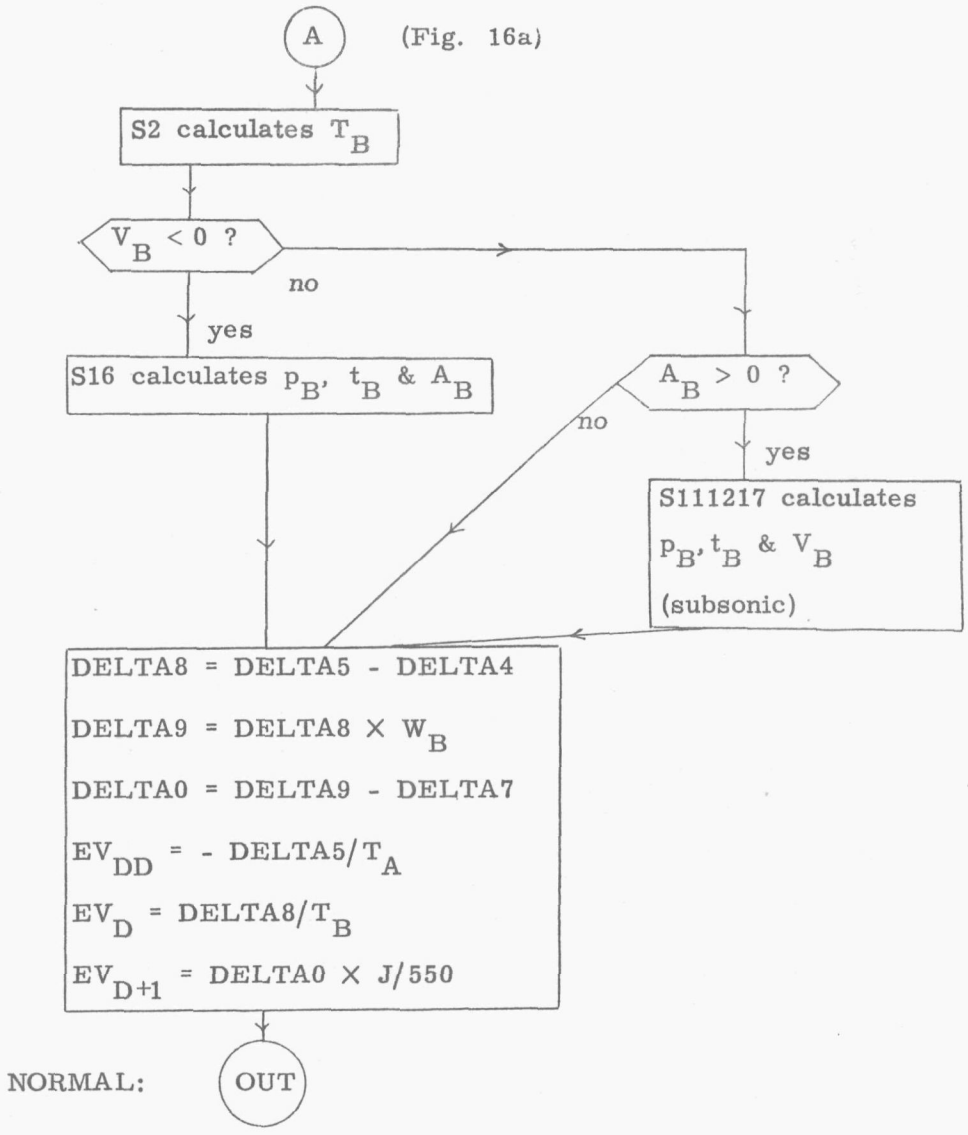

FIGURE 16b - Brick 27 (continued)

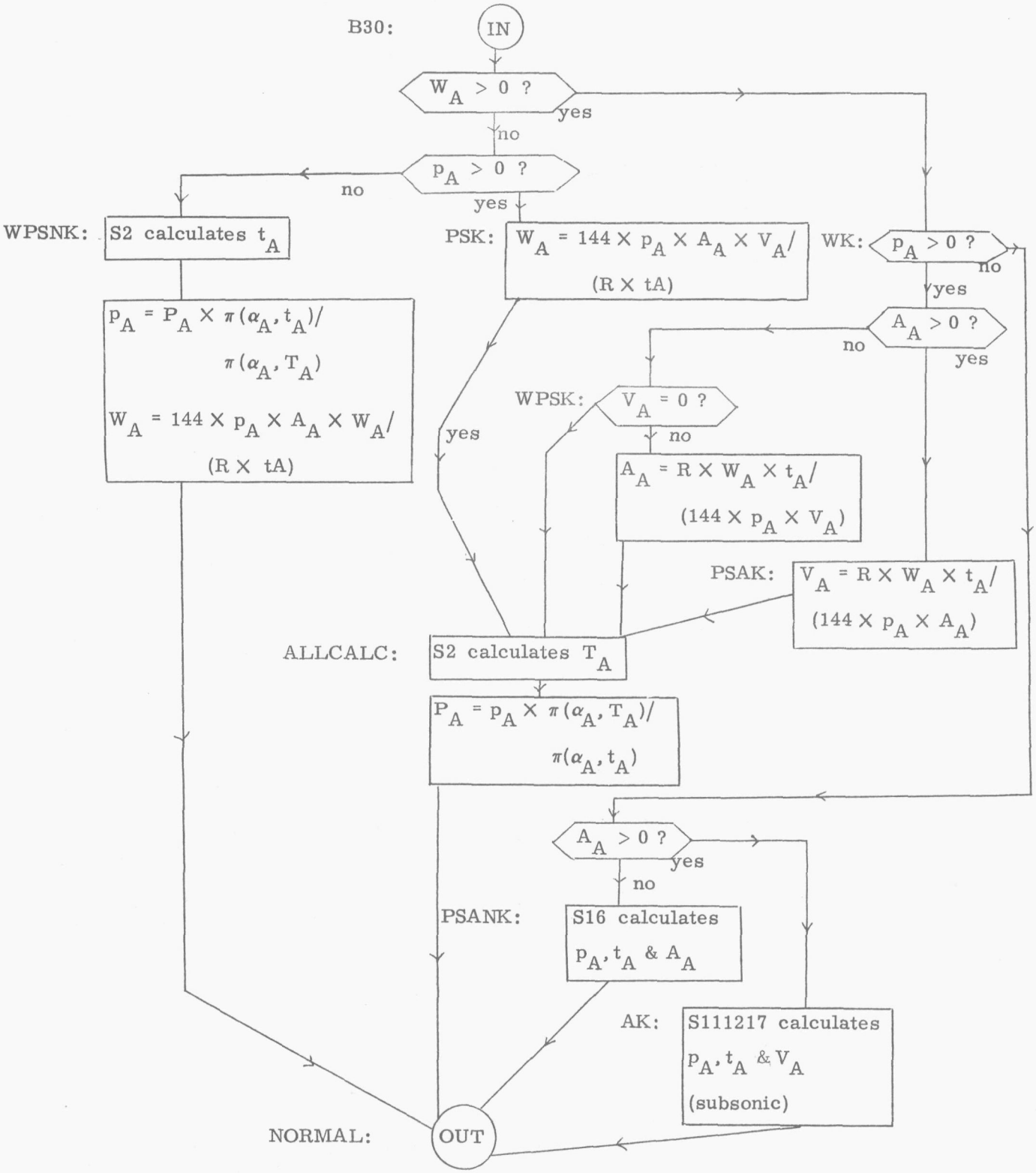

FIGURE 17 - Brick 30 (Filling-in Station Vector)

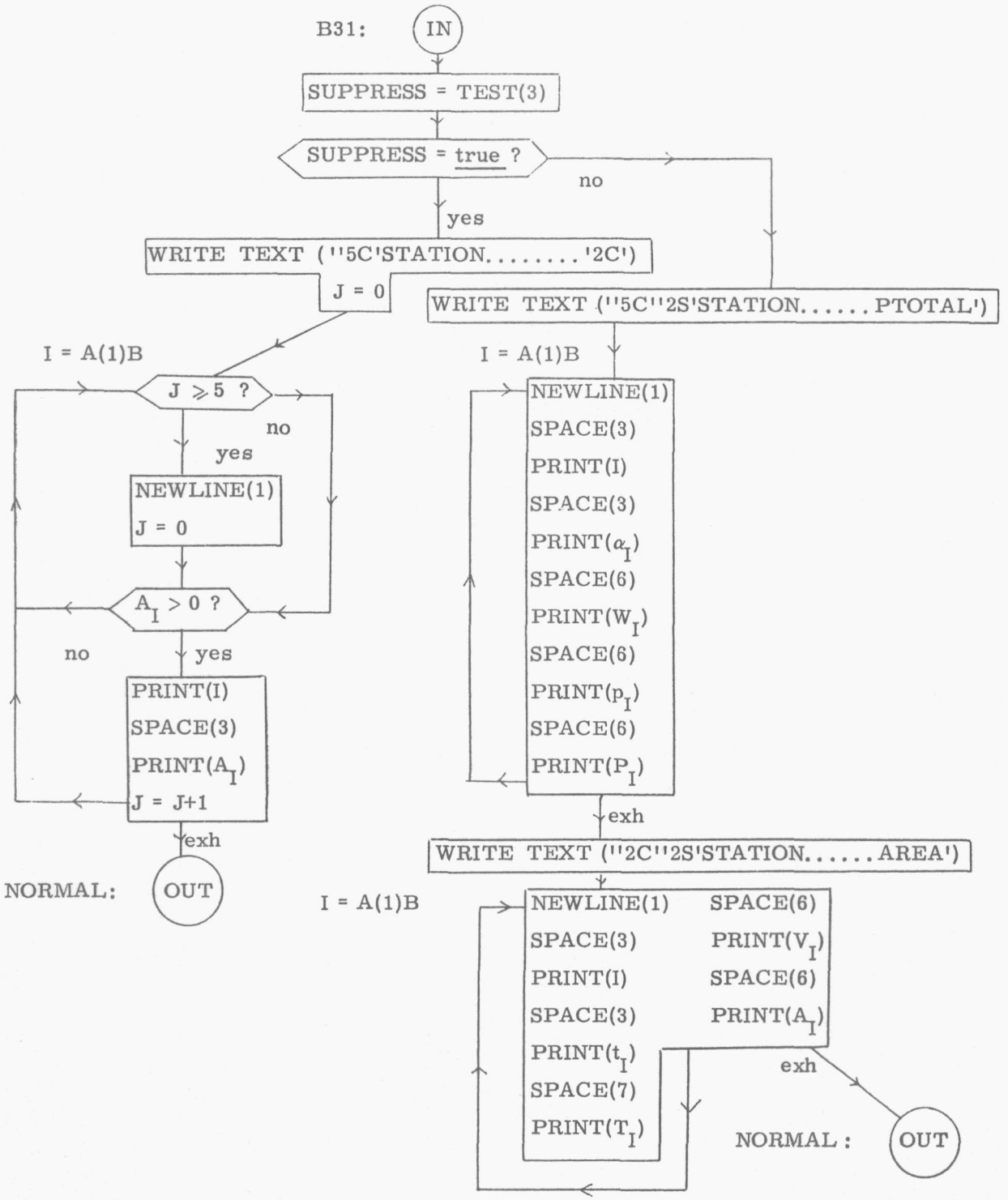

FIGURE 18 - Brick 31 (Station Vector Output)

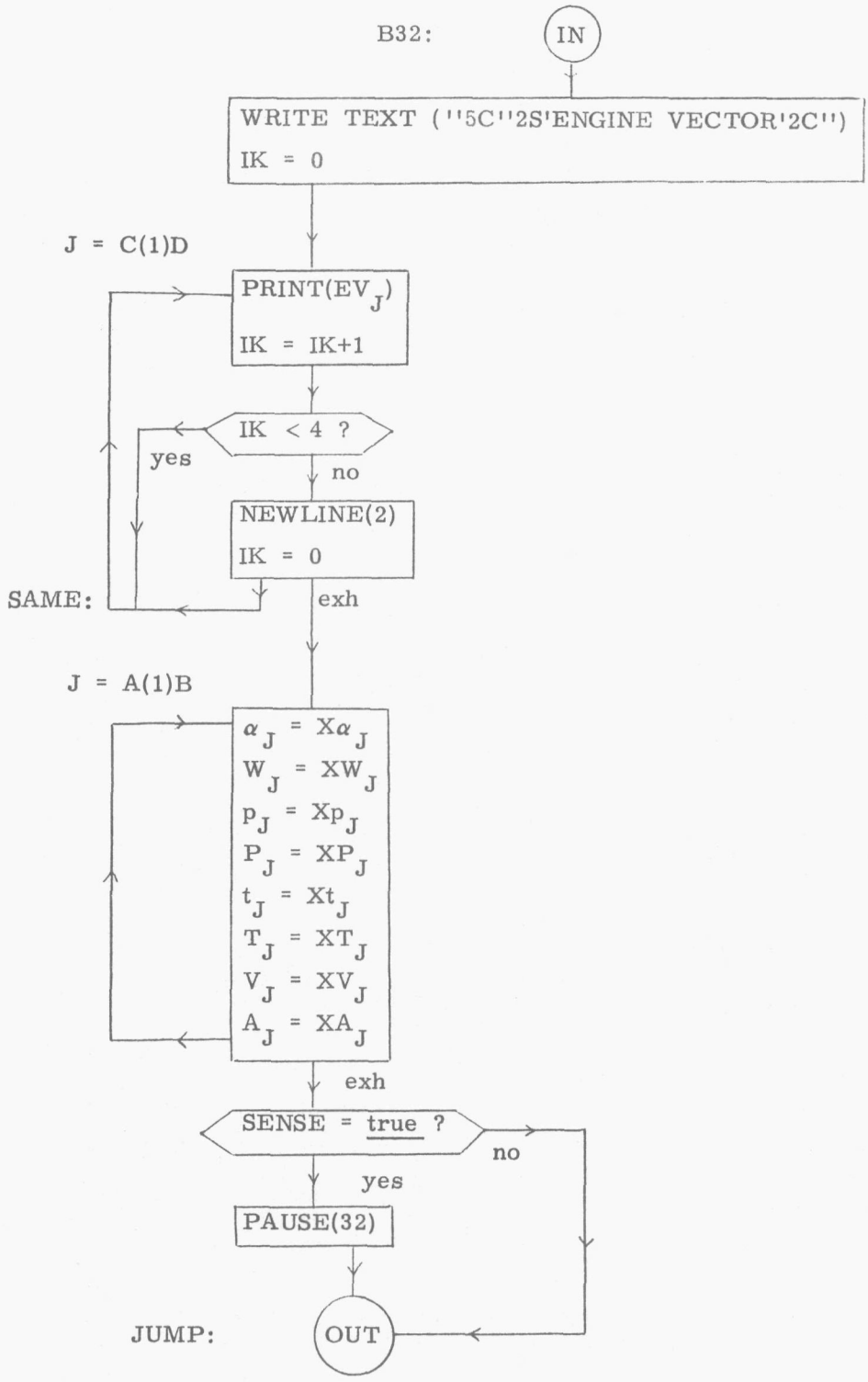

FIGURE 19 - Brick 32 (Engine Vector Output and Station Vector Reset

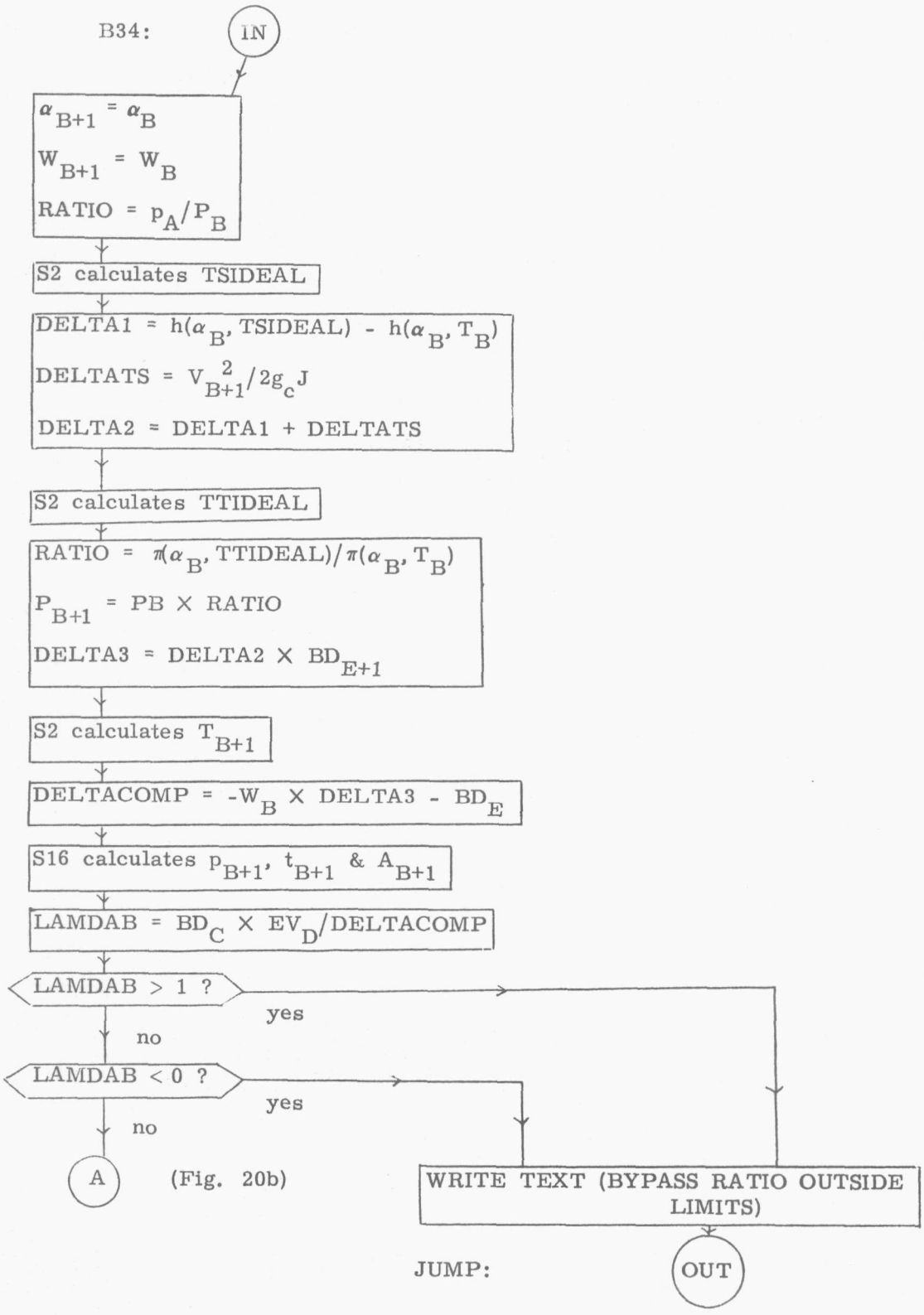

FIGURE 20a - Brick 34 (Find Bypass Ratio)

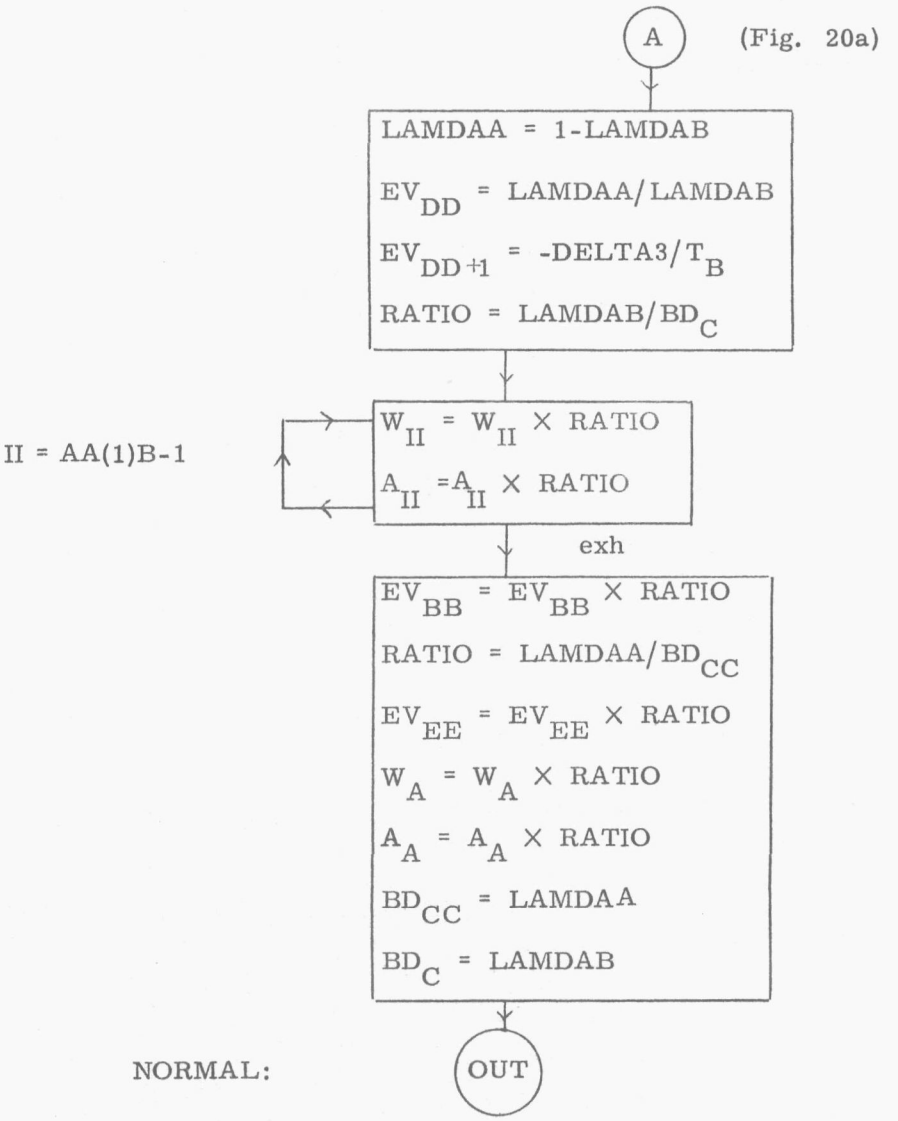

FIGURE 20b - Brick 34 (continued)

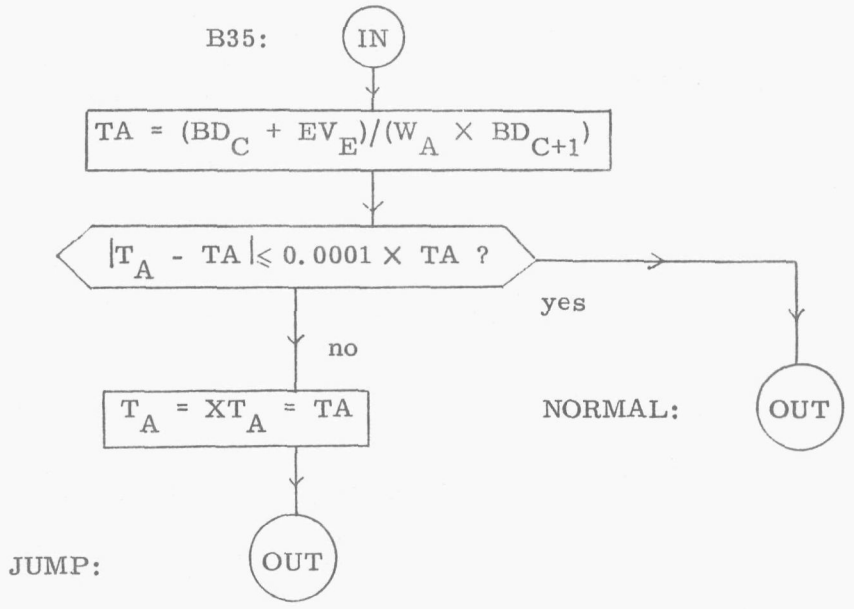

FIGURE 21 - Brick 35 (Find Turbojet Turbine Entry Temperature)

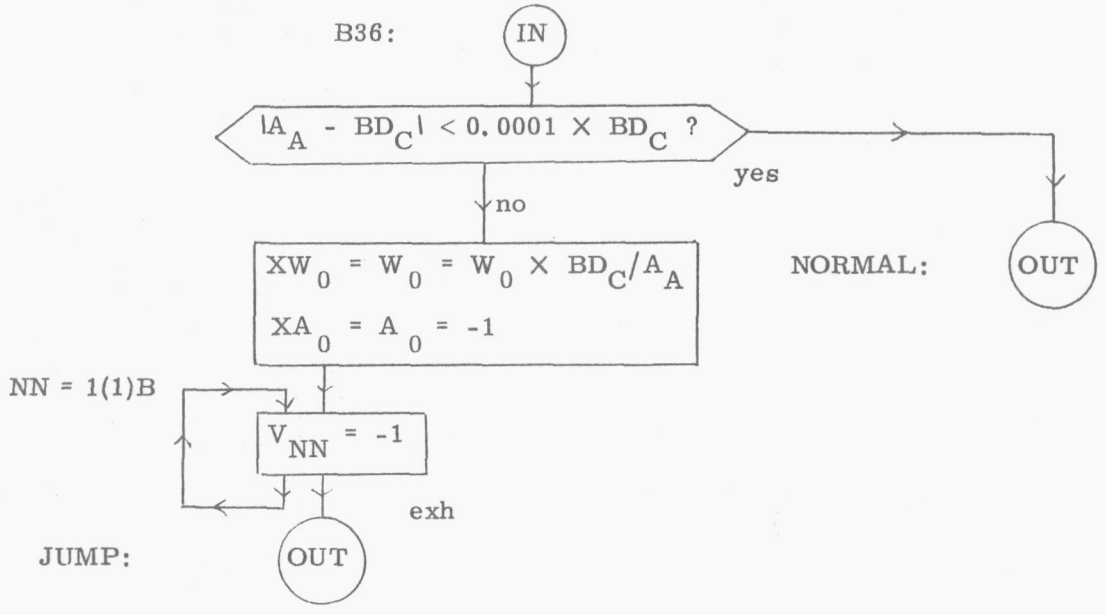

FIGURE 22 - Brick 36 (Find Turbojet Inlet Mass Flow)

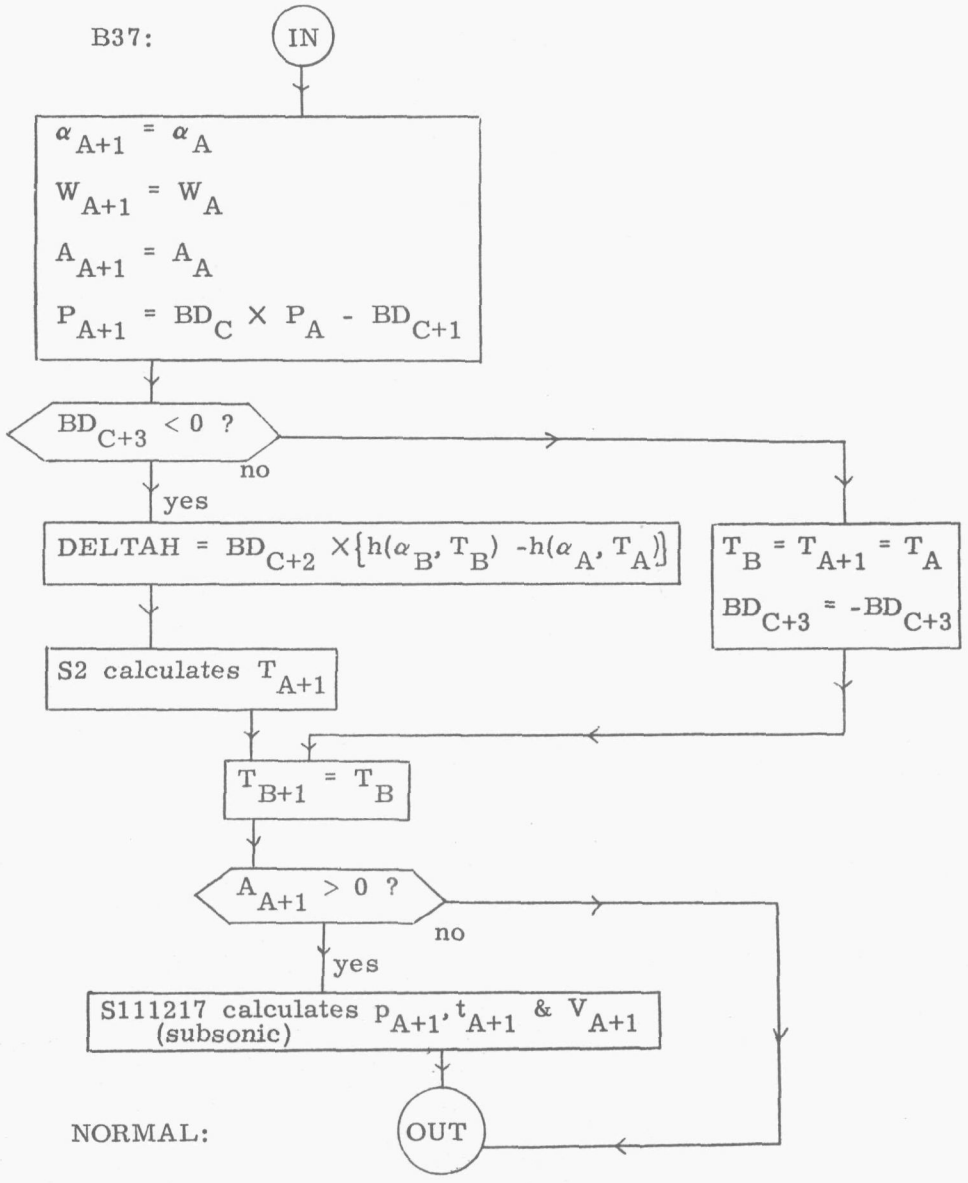

FIGURE 23 - Brick 37 (Heat Exchanger - Cold Side)

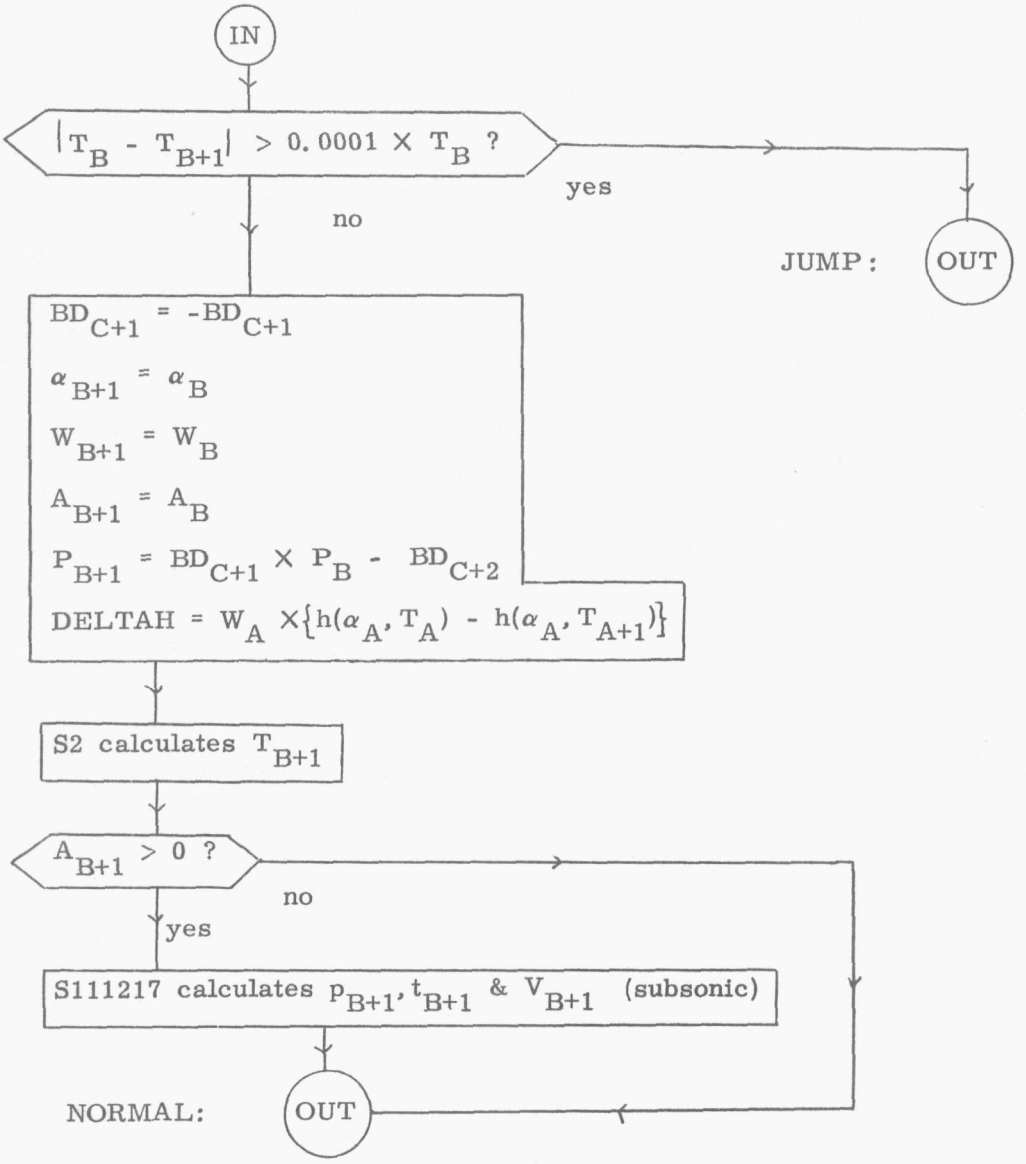

FIGURE 24 - Brick 38 (Heat Exchanger - Hot Side)

09/07/68 14/51/04

```
GO #XALP 21
'LIST' (LP)
'PROGRAM' (R001)
'INPUT' 1=TR0
'OUTPUT' 2=LP0


  'BEGIN' 'REAL' TWOGCJ,GC,R,CONST,LCV,QS;
  'INTEGER' I,K,NEXTBRICK,A,B,C,D,E,F,AA,BB,CC,DD,EE;
  'ARRAY' ALPHA,XALPHA,W,XW,PSTATIC,XPSTATIC,PTOTAL,XPTOTAL,TSTATIC,
  XTSTATIC,TTOTAL,XTTOTAL,VELOCITY,XVELOCITY,AREA,XAREA[0:20],
  BRICKDATA[0:100],A1,A2,C1,C2[2:5],D1,D2[2:6],ENGINEVECTOR[0:50];
  'INTEGER' 'ARRAY' CW[0:50,0:6], CD[0:12];
  'BOOLEAN' FIRST,CHECK,SENSE;
  'SWITCH' BRICK:=B1,B2,B3,B4,B5,B6,B7,B8,B9,B10,B11,B12,B13,B14,B15,
  B16,B17,B18,B19,B20,B21,B22,B23,B24,B25,B26,B27,B28,B29,B30,B31,
  B32,B33,B34,B35,B36,B37,B38,B39,B40;

  'PROCEDURE' DATENOW;
  'EXTERNAL';


  'PROCEDURE' TIMENOW;
  'EXTERNAL';


  'BOOLEAN' 'PROCEDURE' TEST(X);
  'VALUE' X;
  'INTEGER' X;
  'EXTERNAL';


  'COMMENT'******THE REAL PROCEDURES USE A FOURTH ORDER POLYNOMIAL
  APPROXIMATION FOR CP AS A FUNCTION OF T,AND DERIVATIONS FROM
  THIS FOR ENTHALPY AND TEMP. DEPENDENT ENTROPY.COEFFICIENTS
  FOR THE POLYNOMIAL AND ITS DERIVATIONS ARE STORED AS A1,A2,
  FOR CP,D1,D2 FOR H AND C1,C2 FOR JSO/R,WHERE 1 REFERS TO
  AIR AND 2 TO THE DIFFERENCE BETWEEN VALUES FOR AIR AND
  STOICHIOMETRIC PRODUCTS,WHICH ARE PREMULTIPLIED BY (1+QS)/QS
  WHERE QS=STOICHIOMETRIC FUEL/AIR RATIO******;

  'REAL' 'PROCEDURE' SPHT(Q,T);
  'VALUE' Q,T;
  'REAL' Q,T;
    'BEGIN' 'COMMENT'******SPHT(Q,T) CALCULATES AS SPHT THE CP OF A
    MIXTURE FUEL/AIR RATIO Q AT TEMPERATURE T******;
    'REAL' CPA,CPD,X;
    'INTEGER' I;
    X:=(T-1100.0)/900.0;
    CPA:=0.01455924;
    CPD:=-0.072127082;
    'FOR' I:=2 'STEP' 1 'UNTIL' 5 'DO'
      'BEGIN' CPA:=CPA*X+A1[I];
      CPD:=CPD*X+A2[I]
      'END' ;
    SPHT:=CPA+Q/(1.0+Q)*CPD
    'END' SPHT;

  'REAL' 'PROCEDURE' ENTH(Q,T);
  'VALUE' Q,T;
  'REAL' Q,T;
    'BEGIN' 'COMMENT'******ENTH(Q,T) CALCULATES AS ENTH THE ENTHALPY OF
    A MIXTURE FUEL/AIR RATIO Q AT TEMP. T******;
    'REAL' HA ,HD,X;
    'INTEGER' I;
    X:=(T-1100.0)/900.0;
    HA:=2.62066319;
    HD:=-12.9828749;
    'FOR' I:=2 'STEP' 1 'UNTIL' 6 'DO'
      'BEGIN' HA:=HA*X+D1[I];
      HD:=HD*X+D2[I]
      'END';
    ENTH:=HA+Q/(1.0+Q)*HD
    'END' ENTH;

  'REAL' 'PROCEDURE' PRES(Q,T);
  'VALUE' Q,T;
  'REAL' Q,T;
    'BEGIN' 'COMMENT'******PRES(Q,T) CALCULATES AS PRES THE PI FUNCTION
    (=EXP(JSO/R)) OF A MIXTURE FUEL/AIR RATIO Q AT TEMP. T****;
    'REAL' SOA,SOD,X;
    'INTEGER' I;
    X:=(T-1100.0)/900.0;
    SOA:=0.053096219;
    SOD:=-0.263040882;
    'FOR' I:=2 'STEP' 1 'UNTIL' 5 'DO'
      'BEGIN' SOA:=SOA*X+C1[I];
      SOD:=SOD*X+C2[I]
      'END';
    SOA:=SOA+3.56639099*LN(T);
    SOD:=SOD-0.785475254*LN(T);
    PRES:=EXP(SOA+Q/(1.0+Q)*SOD)
    'END' PRES;

  'PROCEDURE' S2(ALPHA,TIN,DELTAH,TOUT);
  'VALUE' ALPHA,TIN,DELTAH;
  'REAL' ALPHA,TIN,DELTAH,TOUT;
    'BEGIN' 'COMMENT'******S2,WHERE DELTAH IS ADDED BETWEEN TIN AND
    TOUT,CALCULATES TOUT,GIVEN ALPHA,TIN,DELTAH******;
    'REAL' ENTHALPY,HCALC,TEST;
    ENTHALPY:=ENTH(ALPHA,TIN)+DELTAH;
    TEST:=0.0001*ENTHALPY;
    TOUT:=ENTHALPY/SPHT(ALPHA,TIN);
    'FOR' HCALC:=ENTH(ALPHA,TOUT) 'WHILE' ABS(HCALC-ENTHALPY)>TEST 'DO'
    TOUT:=TOUT-(HCALC-ENTHALPY)/SPHT(ALPHA,TOUT)
```

```
  'END' S2;

'PROCEDURE' S5(ALPHA,RATIO,TOUT,TIN);
'VALUE' ALPHA,RATIO,TIN;
'REAL' ALPHA,RATIO,TOUT,TIN;
  'BEGIN' 'COMMENT'******S5.GIVEN ALPHA,RATIO(=P2IDEAL/P1) AND TIN
  CALCULATES TOUT FOR ISENTROPIC PROCESS******;
  'REAL' PI2,TEST,PICALC;
  PI2:=PRES(ALPHA,TIN)*RATIO;
  TEST:=0.0001*PI2;
  TOUT:=1000.0*PI2↑(0.0685582/SPHT(ALPHA,TIN));
  'FOR' PICALC:=PRES(ALPHA,TOUT) 'WHILE' ABS(PICALC-PI2)>TEST 'DO'
  TOUT:=TOUT*(1.0-(0.0685582/SPHT(ALPHA,TOUT))*(1.0-PI2/PICALC))
  'END' S5;

'PROCEDURE' S9(A,MA);
'VALUE' A;
'REAL' MA;
'INTEGER' A;
  'BEGIN' 'COMMENT'*****S9 GIVEN A IF ALPHA(A),TSTATIC(A),VELOCITY(A)
  AREKNOWN,CALCULATES THE MACH NUMBER (MA)*******;
  'REAL' CPT,GAMMA;
  CPT:=SPHT(ALPHA[A],TSTATIC[A]);
  GAMMA:=CPT/(CPT-0.0685582);
  MA:=VELOCITY[A]/SQRT(3089.36*GAMMA*TSTATIC[A])
  'END' S9;

'PROCEDURE' S10(A,B,C);
'VALUE' A,B,C;
'INTEGER' A,B,C;
  'BEGIN' 'COMMENT'******S10,GIVEN ALPHA,W,PSTATIC,TTOTAL,VELOCITY AND
  AAREA AT S.V.[A] AND [B],CALCULATES S.V.[C] FOR FRICTIONLESS PARALLEL
  FLOW. IF B IS NEGATIVE,THERE IS A SINGLE INLET FLOW DEFINED BY S.V.
  [A],WHILE IF B IS POSITIVE OR ZERO,MIXING OF THE TWO FLOWS DEFINED
  BY S.V.[A] AND S.V.[B] OCCURS******;
  'REAL' FC,HC,DELTAH,K,L,TERM1,TERM2,VC;
  'IF' B<0 'THEN'
    'BEGIN' ALPHA[C]:=ALPHA[A];
    W[C]:=W[A];
    TTOTAL[C]:=TTOTAL[A];
    AREA[C]:=AREA[A];
    K:=L:=0.0
    'END'
  'ELSE'
    'BEGIN' W[C]:=W[A]+W[B];
    AREA[C]:=AREA[A]+AREA[B];
    FC:=W[A]*ALPHA[A]/(1.0+ALPHA[A])+W[B]*ALPHA[B]/(1.0+ALPHA[B]);
    ALPHA[C]:=FC/(W[C]-FC);
    HC:=(W[A]*ENTH(ALPHA[A],TTOTAL[A])+W[B]*ENTH(ALPHA[B],TTOTAL[B]))
    /W[C];
    DELTAH:=HC-ENTH(ALPHA[C],TTOTAL[A]);
    S2(ALPHA[C],TTOTAL[A],DELTAH,TTOTAL[C]);
    K:=PSTATIC[B]*AREA[B];
    L:=W[B]*VELOCITY[B]
    'END';
  TERM1:=(PSTATIC[A]*AREA[A]+K)/AREA[C]+(W[A]*VELOCITY[A]+L)/
  (4633.056*AREA[C]);
  TERM2:=W[C]/(4633.056*AREA[C]);
  VELOCITY[C]:=1000.0;
  AGAIN:VC:=VELOCITY[C];
  DELTAH:=-VC↑2/TWOGCJ;
  S2(ALPHA[C],TTOTAL[C],DELTAH,TSTATIC[C]);
  PSTATIC[C]:=TERM1-TERM2*VC;
  VELOCITY[C]:=0.666808*W[C]*TSTATIC[C]/(PSTATIC[C]*AREA[C]);
  'IF' ABS(VC-VELOCITY[C])>0.0001*VC 'THEN' 'GO TO' AGAIN;
  PTOTAL[C]:=PSTATIC[C]*PRES(ALPHA[C],TTOTAL[C])/PRES(ALPHA[C],
  TSTATIC[C])
  'END' S10;

'PROCEDURE' S16(A);
'VALUE' A;
'INTEGER' A;
  'BEGIN' 'COMMENT'******S16,GIVEN A ,IF ALPHA(A),W(A),PTOTAL(A),
  TTOTAL(A),VELOCITY(A) ARE KNOWN,CALCULATES PSTATIC(A),
  TSTATIC(A) AND AREA(A)******;
  S2(ALPHA[A],TTOTAL[A],-VELOCITY[A]↑2/TWOGCJ,TSTATIC[A]);
  PSTATIC[A]:=PTOTAL[A]*PRES(ALPHA[A],TSTATIC[A])/PRES(ALPHA[A],
  TTOTAL[A]);
  'IF' VELOCITY[A]≠0 'THEN'
  AREA[A]:=0.666808*W[A]*TSTATIC[A]/(PSTATIC[A]*VELOCITY[A])
  'END' S16;

'PROCEDURE' S111217(A,B,TYPE);
'VALUE' A,B,TYPE;
'INTEGER' A,B,TYPE;
  'BEGIN' 'COMMENT'******S111217,GIVEN A,B AND TYPE, IF ALPHA[A],W[A],
  PTOTAL[A] AND TTOTAL[A] ARE KNOWN, CALCULATES CRITICAL VALUES OF
  PSTATIC[A],TSTATIC[A],VELOCITY[A] AND AREA[A], PROVIDED THAT TYPE≠-2
  OR 2. IF TYPE=0 THIS IS ALL. IF TYPE=-2 OR -1 AND AREA[A] IS GIVEN,
  IT CALCULATES PSTATIC[A],TSTATIC[A] AND VELOCITY[A](SUBSONIC), OR
  SUPERSONIC IF TYPE=1 OR 2. IF TYPE=-1 OR 1 AND GIVEN AREA IS LESS
  THAN THE CRITICAL VALUE,W[0] IS SCALED DOWN
  APPROPRIATELY, AND AREA[0] AND VELOCITIES[1] TO [B] INCLUSIVE ARE
  MARKED UNKNOWN.A JUMP THEN OCCURS TO THE START OF THE MASTER PROGRAM;
      'REAL' ENTHALPY,VSQ,ASQ,CP,GAMMA,T,TCALC,CALCAREA,FUEL,TEST,CONST;
  'INTEGER' L;
  FUEL:=ALPHA[A];
  ENTHALPY:=ENTH(FUEL,TTOTAL[A]);
  'IF' ABS(TYPE)=2 'THEN' 'GO TO' SKIP;
  'FOR' T:=TTOTAL[A]-0.01,T+(VSQ-ASQ)*2.0/(TWOGCJ*CP*(GAMMA+1.0))
  'WHILE' ABS(VSQ-ASQ)>0.0001*VSQ 'DO'
    'BEGIN' VSQ:=TWOGCJ*(ENTHALPY-ENTH(FUEL,T));
    CP:=SPHT(FUEL,T);
    GAMMA:=CP/(CP-0.0685582);
    ASQ:=GC*GAMMA*R*T;
    TCALC:=T
    'END' CRITICAL LOOP;
  PSTATIC[A]:=PTOTAL[A]*PRES(FUEL,TCALC)/PRES(FUEL,TTOTAL[A]);
  VELOCITY[A]:=SQRT(VSQ);
  CALCAREA:=0.666808*W[A]*TCALC/(PSTATIC[A]*VELOCITY[A]);
  'IF' TYPE=0 'THEN'
    'BEGIN' TSTATIC[A]:=TCALC;
    AREA[A]:=CALCAREA;
    'GO TO' FINISH
```

```
    'END';
  'IF' AREA[A]<CALCAREA 'THEN' 'GO TO' ERROR;
  SKIP:CONST:=0.666808*W[A]*PRES(FUEL,TTOTAL[A])/PTOTAL[A];
  TEST:=0.0001*AREA[A];
  'FOR' T:='IF' TYPE<0 'THEN' TTOTAL[A]-0.01 'ELSE' 200.0,
  T-(1.0-AREA[A]/CALCAREA)/(1400.69*CP*(GC/VSQ-1.0/(GAMMA*R*T)))
  'WHILE' ABS(CALCAREA-AREA[A])>TEST 'DO'
    'BEGIN' VSQ:=TWOGCJ*(ENTHALPY-ENTH(FUEL,T));
    CP:=SPHT(FUEL,T);
    GAMMA:=CP/(CP-0.0685582);
    CALCAREA:=CONST*T/(PRES(FUEL,T)*SQRT(VSQ));
    TCALC:=T
    'END' FINAL LOOP;
  PSTATIC[A]:=PTOTAL[A]*PRES(FUEL,TCALC)/PRES(FUEL,TTOTAL[A]);
  TSTATIC[A]:=TCALC;
  VELOCITY[A]:=SQRT(VSQ);
  'GO TO' FINISH;
  ERROR:XW[0]:=W[0]:=W[0]*0.999*AREA[A]/CALCAREA;
  XAREA[0]:=AREA[0]:=-1.0;
  'FOR' L:=1 'STEP' 1 'UNTIL' 8 'DO' VELOCITY[L]:=-1.0;
  WRITE TEXT('(''('2C')'AREA<CRITICAL%IN%BRICK')');
  PRINT(NEXTBRICK,2,0);
  'GO TO' JUMP;
  FINISH:
  'END' S111217;

'COMMENT'******HERE STARTS THE PROGRAMME PROPER******;


SENSE:=TEST(2);
'IF' SENSE 'THEN'
  'BEGIN' K:=I-2;
  'GO TO' COMMON
  'END';
CHECK:=TEST(1);
SELECT INPUT(1);
SELECT OUTPUT(2);
PAPERTHROW;
NEWLINE(2);
WRITE TEXT('('R001%TURBOCODE'('5C')'')');
DATENOW;
NEWLINE(5);
TWOGCJ:=90131.6;
GC:=32.174;
R:=96.0204;
LCV:=10400.0;
QS:=0.06823;
A1[2]:=-0.008914491;
A1[3]:=-0.02233222;
A1[4]:=0.03906451;
A1[5]:=0.2768212;
A2[2]:=0.049318373;
A2[3]:=-0.028102521;
A2[4]:=0.195281359;
A2[5]:=0.477810178;
D1[2]:=-2.00576047;
D1[3]:=-6.69966599;
D1[4]:=17.5790296;
D1[5]:=249.139080;
D1[6]:=277.32;
D2[2]:=11.0966339;
D2[3]:=-8.43075621;
D2[4]:=87.87661170;
D2[5]:=430.02916;
D2[6]:=323.766085;
C1[2]:=-0.129874284;
C1[3]:=0.07521572;
C1[4]:=0.38599815;
C1[5]:=-19.8581411;
C2[2]:=0.6684720140;
C2[3]:=-1.43050668;
C2[4]:=6.34548724;
C2[5]:=11.4669278;
CD[0]:=CODE('('0')');
CD[1]:=CODE('('1')');
CD[2]:=CODE('('2')');
CD[3]:=CODE('('3')');
CD[4]:=CODE('('4')');
CD[5]:=CODE('('5')');
CD[6]:=CODE('('6')');
CD[7]:=CODE('('7')');
CD[8]:=CODE('('8')');
CD[9]:=CODE('('9')');
CD[10]:=CODE('('EL')');
CD[11]:=CODE('(',')');
CD[12]:=CODE('(')')');

INPUT:FIRST:='TRUE';
  'BEGIN''COMMENT'******CODEWORDS ARE STORED AS A 2-DIMENSIONAL
  ARRAY CW[0:50,0:6].THE FIRST DIMENSION INDICATES THE SEQUENCE
  ORDER OF THE CODEWORD,THE SECOND REPRESENTS N,A,B,C,D,E,

  F. THUS EACH CODEWORD OCCUPIES ONE LINE IN THE ARRAY. K   IS

  THENUMBER OF THE FIRST CODEWORD TO BE OBEYED,I IS THE NUMBER
  OF THE LAST CODEWORD,PLUS ONE******;
  'INTEGER' J,II,S,T;
  'BOOLEAN' NEW;
  'SWITCH' CS:=DIGIT,DIGIT,DIGIT,DIGIT,DIGIT,DIGIT,DIGIT,
  DIGIT,DIGIT,DIGIT,EL,COMMA,OBEY;
  NEWLINE(5);
  COPYTEXT('('£$')');
  'FOR' I:=0 'STEP' 1 'UNTIL' 50 'DO'
    'BEGIN' NEW:='TRUE';
    'FOR' J:=0 'STEP' 1 'UNTIL' 6 'DO'
      'BEGIN' K:=0;
      AGAIN:S:=READ CH;
      'FOR' T:=0 'STEP' 1 'UNTIL' 12 'DO'
      'IF' S=CD[T]'THEN''GOTO' CS[T+1];
      'GOTO' AGAIN;
      DIGIT:NEW:='FALSE';
      K:=10*K+T;
      'GOTO' AGAIN;
      EL:'IF' NEW 'THEN''GOTO' AGAIN;
      CW[I,J]:=K;
      'FOR' II:=J+1 'STEP' 1 'UNTIL' 6 'DO'
```

```
      CW[I,II]:=0;
      'GOTO' NOMORE;
      COMMA:CW[I,J]:=K
      'END' LOOP CONTROLLED BY J;
    WRITE TEXT ('('''('2C')'CODEWORD%TOO%LONG')');
    'GOTO' NEXTPROGRAMME;
    NOMORE:
    'END' LOOP CONTROLLED BY I;
  WRITE TEXT ('('''('2C')'TOO%MANY%CODEWORDS%LOADED')');
  'GOTO' NEXTPROGRAMME
  'END' CODEWORD INPUT;

'COMMENT'******CODEWORD OBEY-CONTROL IS TRANSFERRED TO A BLOCK APPRO-
PRIATE TO THE BRICK SPECIFIED BY CW(K,0).THE BRICK WILL
NORMALLY RETURN TO AN INSTRUCTION INCREASING K BY 1,
BUT A JUMP EXIT WILL SET K IN THE BRICK.IF K EXCEEDS THE
TOTAL NUMBER OF CODEWORDS,THE OPERATOR IS REQUESTED TO LOAD
THE NEXT TURBOCODE PROGRAMME.OTHERWISE THE CYCLE IS REPEATED.***
THE FORMER BRICK 40 IS ALSO INCORPORATED, WHICH SETS OR CANCELS
THE CHECKING FACILITY. THE FIRST ENTRY WILL CAUSE A PRINT OF
THE ABBREVIATED BRICK OR PROCEDURE NAME FOR EACH SUBSEQUENT BRICK
OR PROCEDURE CALL. A SECOND CALL OF BRICK 40 WILL CANCEL THE
PRINTING, A THIRD WILL RESET IT. ETC., BUT IF SENSE SWITCH
1 IS OFF, NO CHECK PRINTING WILL OCCUR AT ALL******;
JUMP:K:=F;
'GO TO' COMMON;
B40:NORMAL:K:=K+1;
COMMON:'IF' K 'GE' I 'OR' K<0 'THEN'
'GO TO' NEXTPROGRAMME;
OBEY:CHECK:=TEST(1);
NEXTBRICK:=CW[K,0];
A:=CW[K,1];
B:=CW[K,2];
C:=CW[K,3];
D:=CW[K,4];
E:=CW[K,5];
F:=CW[K,6];
'IF' NEXTBRICK 'LE' 40 'AND' NEXTBRICK>0 'THEN' 'GO TO'
BRICK[NEXTBRICK];

B7:B11:B13:B14:B17:B18:B19:B20:B21:B28:
WRITE TEXT ('('''('2C')'''('2S')'BRICK')');
PRINT(NEXTBRICK,2,0);
WRITE TEXT ('('''('2S')'NOT%AVAILABLE')');

NEXTPROGRAMME:WRITE TEXT('('''('2C')'LOAD%MORE%CODEWORDS%IF%
AVAILABLE')');
PAUSE(40);
'GO TO' INPUT;

B1:
  'BEGIN' 'COMMENT'*******BRICK 1-TRANSFORMATION.LAMBDAW,DELTAW
  LAMBDAP AND DELTAP ARE STORED AS C-TH AND SUBSEQUENT
  ELEMENTS IN BRICKDATA******;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B1')');
  ALPHA[B]:=ALPHA[A];
  TTOTAL[B]:=TTOTAL[A];
  W[B]:=W[A]*BRICKDATA[C]-BRICKDATA[C+1];
  PTOTAL[B]:=PTOTAL[A]*BRICKDATA[C+2]-BRICKDATA[C+3];
  'IF' VELOCITY[B] > 0 'THEN' S16(B) 'ELSE' 'IF' AREA[B] > 0 'THEN'
  S111217(B,B,-1);
  'GO TO' NORMAL
  'END' BRICK 1;

B2:B10:B12:
  'BEGIN'
  'COMMENT'******BRICK-COMPRESSION GIVEN PRESSURE RATIO-REQUIRES PRESS

  -URE RATIO AND POLYTROPIC EFFICIENCY AS BRICKDATA[C] AND [C+1].
  BRICK 10-COMPRESSION GIVEN TEMPERATURE RISE-REQUIRES TEMPERATURE
  RISE AND POLYTROPIC EFFICIENCY AS BRICKDATA[C] AND [C+1]. BRICK 12-
  COMPRESSION GIVEN WORK INPUT-REQUIRES WORK INPUT AS ENGINEVECTOR[E]
  AND POLYTROPIC EFFICIENCY AS BRICKDATA[C]. ALL THREE BRICKS,
  GIVEN ALSO STATION VECTOR[A], CALCULATE STATION VECTOR[B] AND ALSO
  WORK INPUT AS ENGINEVECTOR[D]******;
  'REAL' EFFICIENCY;
  'IF' CHECK 'THEN'
    'BEGIN' WRITE TEXT('('''('2C')'B')');
    PRINT(NEXTBRICK,2,0)
    'END';
  ALPHA[B]:=ALPHA[A];
  W[B]:=W[A];
  'IF' NEXTBRICK=2 'THEN'
    'BEGIN' PTOTAL[B]:=PTOTAL[A]*BRICKDATA[C];
    S5(ALPHA[B],BRICKDATA[C]↑(1.0/BRICKDATA[C+1]),TTOTAL[B],TTOTAL[A]);
    'GO TO' LAST
    'END'
  'ELSE' 'IF' NEXTBRICK=10 'THEN'
    'BEGIN' TTOTAL[B]:=TTOTAL[A]+BRICKDATA[C];
    EFFICIENCY:=BRICKDATA[C+1]
    'END'
  'ELSE'
    'BEGIN' S2(ALPHA[A],TTOTAL[A],ENGINEVECTOR[E]/W[A],TTOTAL[B]);
    EFFICIENCY:=BRICKDATA[C]
    'END';
  PTOTAL[B]:=PTOTAL[A]*(PRES(ALPHA[A],TTOTAL[B])/
  PRES(ALPHA[A],TTOTAL[A]))↑EFFICIENCY;
  LAST:'IF' VELOCITY[B]>0 'THEN' S16(B) 'ELSE' 'IF' AREA[B]>0 'THEN'
  S111217(B,B,-1);
  'IF' CW[K+1,0]=3 'THEN' 'GO TO' NORMAL
  'END' BRICKS 2,10 AND 12;

B3:
  'BEGIN' 'COMMENT'******BRICK 3-WORK DONE .W.D.(=DIFFERENCE BETWEEN
  TOTAL ENTHALPIES AT A AND B I.E.W(A)*(HB-HA))IS
  STORED IN ENGINEVECTOR(D)******;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B3')');
  ENGINEVECTOR[D]:=W[A]*(ENTH(ALPHA[A],TTOTAL[B])-ENTH(ALPHA[A],
  TTOTAL[A]));
  'GO TO' NORMAL;
  'END' BRICK 3;

B4:
  'BEGIN' 'COMMENT'******BRICK 4-SINGLE TURBINE.
```

```
  REQUIRED WORK OUTPUT IS ENGINEVECTOR DATA,
  -DELTAH/TA IS ENGINEVECTOR RESULT,AUXILIARY WORK
  AND ADIABATIC EFFICIENCY ARE C-TH AND NEXT BRICKDATA******;
  'REAL' DELTAH,TB;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B4')');
  ALPHA[B]:=ALPHA[A];
  W[B]:=W[A];
  DELTAH:=-(ENGINEVECTOR[E]+BRICKDATA[C])/W[A];
  ENGINEVECTOR[D]:=-DELTAH/TTOTAL[A];
  S2(ALPHA[A],TTOTAL[A],DELTAH,TTOTAL[B]);
  DELTAH:=DELTAH/BRICKDATA[C+1];
  S2(ALPHA[A],TTOTAL[A],DELTAH,TB);
  PTOTAL[B]:=PTOTAL[A]*PRES(ALPHA[A],TB)/PRES(ALPHA[A],TTOTAL[A]);
  'IF' VELOCITY[B] > 0 'THEN' S16(B) 'ELSE' 'IF' AREA[B] > 0 'THEN'
  S111217(B,B,-1);
  'GO TO' NORMAL
  'END' BRICK 4;

B5:
  'BEGIN' 'COMMENT'******BRICK 5-FRICTIONLESS PARALLEL MIXING-
  GIVEN S.V.[A] AND [B] WHERE A AND B ARE THE TWO INLET PLANES
  CALCULATES S.V.[B+1], THE OUTLET PLANE, ASSUMING NO FRICTION
  AND NO OVERALL CHANGE IN AREA******;
  'IF' CHECK 'THEN' WRITE TEXT('('''('2C')'B5')');
  S10(A,B,B+1);
  'GO TO' NORMAL
  'END' BRICK 5;

B6:B29:B39:
  'BEGIN' 'COMMENT'******BRICK 6(FRICTIONLESS CONSTANT PRESSURE
  HEATING WITH FUNDAMENTAL PRESSURE LOSS), BRICK 29(DITTO WITHOUT LOSS)
  AND BRICK 39(CONSTANT VOLUME HEATING)-GIVEN S.V.[A] AND TTOTAL[B],
  ALSO COMBUSTION EFFICIENCY AS BRICKDATA[C], CALCULATES S.V.[B] AND
  FUEL FLOW AS ENGINEVECTOR[D], TAKING THE LIQUID FUEL ENTHALPY AS
  ZERO. FOR BRICK 6 ONLY, THE GIVEN VALUE OF TTOTAL[B] MAY EXCEED THAT
  REQUIRED FOR THERMAL CHOKING, IN WHICH CASE A MESSAGE IS PRINTED AND
  A JUMP EXIT OCCURS******;
  'REAL' DUM,ECV,Q2,F,IMPULSE,CP,GAMMA,MB,DELTAH,TSTAR;
  'IF' CHECK 'THEN'
    'BEGIN' WRITE TEXT('('''('2C')'B')');
    PRINT(NEXTBRICK,2,0)
    'END';
  DUM:=ENTH(ALPHA[A],TTOTAL[B])-ENTH(ALPHA[A],TTOTAL[A]);
  ECV:=LCV+ENTH(0,TTOTAL[B])/QS-ENTH(QS,TTOTAL[B])*(1.0+1.0/QS);
  'IF'NEXTBRICK=39'THEN'
    'BEGIN' DUM:=DUM-0.0685582*(TTOTAL[B]-TTOTAL[A]);
    ECV:=ECV+0.0685582*TTOTAL[B]
    'END';
  Q2:=DUM/ECV;
  ALPHA[B]:=ALPHA[A]+Q2*(1.0+ALPHA[A]);
  F:=W[A]*ALPHA[B]/(BRICKDATA[C]*(1.0+ALPHA[A]));
  ENGINEVECTOR[D]:=3600.0*F;
  W[B]:=W[A]+F;
  'IF' NEXTBRICK=6 'THEN' 'GO TO' CHOKING 'ELSE'
  'IF' NEXTBRICK=29 'THEN' PTOTAL[B]:=PTOTAL[A]
  'ELSE' PTOTAL[B]:=PTOTAL[A]*TTOTAL[B]/TTOTAL[A];
  'IF' AREA[B]>0 'THEN' 'GO TO' FINAL;
  AREA[B]:=AREA[A];
  'IF' AREA[B] 'LE' 0 'THEN' 'GO TO' NORMAL;
  FINAL:S111217(B,B,-1);
  'GO TO' NORMAL;
  CHOKING:AREA[B]:=AREA[A];
  IMPULSE:=PSTATIC[A]+W[A]*VELOCITY[A]/(4633.056*AREA[B]);
  TSTATIC[B]:=TTOTAL[A];
  CALCCRIT:CP:=SPHT(ALPHA[B],TSTATIC[B]);
  GAMMA:=CP/(CP-0.0685582);
  PSTATIC[B]:=IMPULSE/(1.0+GAMMA);
  VELOCITY[B]:=0.666808*W[B]*TSTATIC[B]/(PSTATIC[B]*AREA[B]);
  S9(B,MB);
  'IF' ABS(MB-1.0) 'LE' 0.0001 'THEN' 'GO TO' SONIC;
  TSTATIC[B]:=TSTATIC[B]/MB↑2;
  'IF' TSTATIC[B]<2000.0 'THEN' 'GO TO' CALCCRIT;
  CALCPRESS:S10(A,-1,B);
  'GO TO' NORMAL;
  SONIC:'IF' TSTATIC[B] 'GE' 1850.0 'THEN' 'GO TO' CALCPRESS;
  DELTAH:=VELOCITY[B]↑2/TWOGCJ;
  S2(ALPHA[B],TSTATIC[B],DELTAH,TSTAR);
  'IF' TSTAR 'GE' TTOTAL[B] 'THEN' 'GO TO' CALCPRESS;
  WRITE TEXT('('''('2C')'THERMAL%CHOKING')');
  'GO TO' JUMP
  'END' BRICKS 6,29 AND 39;

B8:B23:
  'BEGIN' 'COMMENT'******BRICK 8(OPTIMUM CONVERGENT NOZZLE) AND
  BRICK 23(OPTIMUM CONVERGENT-DIVERGENT NOZZLE)-GIVEN S.V.[A],PSTATIC
  [0] AND DISCHARGE AND THRUST COEFFICIENTS AS BRICKDATA[C] AND [C+1]
  CALCULATES S.V.[B](THROAT) AND S.V.[B+1](EXIT,CON-DI CASE ONLY), ALSO
  GROSS THRUST AS ENGINEVECTOR[D], ASSUMING ISENTROPIC EXPANSION. IN
  THE CON-DI CASE, IF PRESSURE RATIO IS LESS THAN CRITICAL, A
  CONVERGENT NOZZLE IS ASSUMED AND A MESSAGE IS PRINTED INDICATING
  THAT S.V.[B+1] IS TO BE IGNORED******;
  'INTEGER' N;
  'REAL' MB;
  'IF' CHECK 'THEN'
    'BEGIN' WRITE TEXT('('''('2C')'B')');
    PRINT(NEXTBRICK,2,0)
    'END';
  'IF' PTOTAL[A]<PSTATIC[0] 'THEN'
    'BEGIN' WRITE TEXT('('''('2C')'JET%VELOCITY%IMAGINARY')');
    'GO TO' JUMP
    'END';
  N:=B;
  CONDI:ALPHA[N]:=ALPHA[A];
  W[N]:=W[A];
  PTOTAL[N]:=PTOTAL[A];
  TTOTAL[N]:=TTOTAL[A];
  'IF' N>B 'THEN' 'GO TO' CALCEXIT;
  S5(ALPHA[B],PSTATIC[0]/PTOTAL[B],TSTATIC[B],TTOTAL[B]);
  VELOCITY[B]:=SQRT(TWOGCJ*(ENTH(ALPHA[B],TTOTAL[B])-
  ENTH(ALPHA[B],TSTATIC[B])));
  S9(B,MB);
  'IF' NEXTBRICK=8 'AND' MB>1 'THEN' 'GO TO' CHOKED;
  PSTATIC[B]:=PSTATIC[0];
  AREA[B]:=0.666808*W[B]*TSTATIC[B]/(PSTATIC[B]*VELOCITY[B]);
```

```
  FINISH:AREA[B]:=AREA[B]/BRICKDATA[C];
  ENGINEVECTOR[D]:=BRICKDATA[C+1]*(W[B]*VELOCITY[B]/GC+144.0*AREA[B]*
  (PSTATIC[B]-PSTATIC[0]));
  'IF' NEXTBRICK=8 'THEN' 'GO TO' NORMAL;
  N:=B+1;
  'IF' MB>1 'THEN' 'GO TO' CONDI;
  WRITE TEXT('('''('2C')'CONVERGENT%NOZZLE-IGNORE%STATION')');
  PRINT(N,2,0);
  'GO TO' NORMAL;
  CALCEXIT:PSTATIC[N]:=PSTATIC[B];
  TSTATIC[N]:=TSTATIC[B];
  VELOCITY[N]:=VELOCITY[B];
  AREA[N]:=AREA[B];
  CHOKED:S111217(B,B,0);
  'IF' NEXTBRICK=8 'THEN' 'GO TO' FINISH;
  AREA[B]:=AREA[B]/BRICKDATA[C];
  'GO TO' NORMAL
  'END' BRICKS 8 AND 23;

B9:
  'BEGIN' 'COMMENT'******BRICK 9-CONSTANT AREA DUCT TOTAL PRESSURE
  LOSS-GIVEN DELTAP/(PTOTAL-PSTATIC) AS BRICKDATA[C]AND
  S.V.[A]CALCULATES S.V.[B]******;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B9')');
  ALPHA[B]:=ALPHA[A];
  W[B]:=W[A];
  TTOTAL[B]:=TTOTAL[A];
  AREA[B]:=AREA[A];
  PTOTAL[B]:=PTOTAL[A]-(PTOTAL[A]-PSTATIC[A])*BRICKDATA[C];
  S111217(B,B,-1);
  'GO TO' NORMAL
  'END' BRICK 9;

B15:
  'BEGIN' 'COMMENT'******BRICK 15-STATION VECTOR INPUT. EACH ITEM
  WITHIN A STATION VECTOR IS STORED IN A SEPARATE ARRAY. THERE ARE
  THUS EIGHT ARRAYS:ALPHA,W,PSTATIC,PTOTAL,TSTATIC,TTOTAL,
  VELOCITY,AREA(0:20).ANY ITEM NOT GIVEN A VALUE IS SET TO -1.
  A DUPLICATE SET IS STORED IN XALPHA.......XAREA******;
  'INTEGER' I,STATION,NUMBER;
  'SWITCH' VECTOR:=V0,V1,V2,V3,V4,V5,V6,V7;
  NEWLINE(5);
  TIMENOW;
  NEWLINE(5);
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B15')');
  'IF' FIRST 'THEN'
    'BEGIN' 'FOR' I:=0 'STEP' 1 'UNTIL' 20 'DO' ALPHA[I]:=W[I]
    :=PSTATIC[I]:=PTOTAL[I]:=TSTATIC[I]:=TTOTAL[I]
    :=VELOCITY[I]:=AREA[I]:=-1;
    COPYTEXT('('£$')');
    FIRST:='FALSE'
    'END';
  READSTATION: STATION:=READ;
  'IF' STATION=-1 'THEN' 'GO TO' FINISH;
  NUMBER:=READ;
  'GO TO' VECTOR[NUMBER+1];
  V0:ALPHA[STATION]:=READ;
  'GO TO' READSTATION;
  V1:W[STATION]:=READ;
  'GO TO' READSTATION;
  V2:PSTATIC[STATION]:=READ;
  'GO TO' READSTATION;
  V3:PTOTAL[STATION]:=READ;
  'GO TO' READSTATION;
  V4:TSTATIC[STATION]:=READ;
  'GO TO' READSTATION;
  V5:TTOTAL[STATION]:=READ;
  'GO TO' READSTATION;
  V6:VELOCITY[STATION]:=READ;
  'GO TO' READSTATION;
  V7:AREA[STATION]:=READ;
  'GO TO' READSTATION;
  FINISH:'FOR' I:=0 'STEP' 1 'UNTIL' 20 'DO'
    'BEGIN' XALPHA[I]:=ALPHA[I];
    XW[I]:=W[I];
    XPSTATIC[I]:=PSTATIC[I];
    XPTOTAL[I]:=PTOTAL[I];
    XTSTATIC[I]:=TSTATIC[I];
    XTTOTAL[I]:=TTOTAL[I];
    XVELOCITY[I]:=VELOCITY[I];
    XAREA[I]:=AREA[I]
    'END' LOOP CONTROLLED BY I;
  'GO TO' NORMAL
  'END' BRICK 15;

B16:
  'BEGIN' 'COMMENT'******BRICK 16-BRICK DATA INPUT. BRICK DATA IS
  STORED AS ONE ARRAY, BRICKDATA[0:100];
  'INTEGER' NUMBER;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B16')');
  START:NUMBER:=READ;
  'IF' NUMBER=-1 'THEN' 'GO TO' NORMAL;
  BRICKDATA[NUMBER]:=READ;
  'GO TO' START
  'END' BRICK 16;

B22:
  'BEGIN' 'COMMENT'******BRICK22-ARITHMETIC ON EV,SV AND BD******;
  'SWITCH' EV:=E0,E1,E2,E3,E4,E5,E6,E7;
  'SWITCH' SV:=S0,S1,S2,S3,S4,S5,S6,S7;
  'SWITCH' FUNCTION:=EVTIMESEV,EVDIVEV,EVPLUSEV,EVMINUSEV,MINUSEV,
  BDTOEV,EVTOBD,SVTOEV,EVTOSV;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B22')');
  'IF' C 'GE' 0 'AND' C 'LE' 8 'THEN' 'GO TO' FUNCTION[C+1];
  WRITE TEXT ('('''('2C')'WRONG%VALUE%FOR%C%IN%CODEWORD')');
  PRINT(K,2,0);
  'GO TO' NEXTPROGRAMME;
  EVTIMESEV:ENGINEVECTOR[D]:=ENGINEVECTOR[A]*ENGINEVECTOR[B];
  'GO TO' NORMAL;
  EVDIVEV:ENGINEVECTOR[D]:=ENGINEVECTOR[A]/ENGINEVECTOR[B];
  'GO TO' NORMAL;
  EVPLUSEV:ENGINEVECTOR[D]:=ENGINEVECTOR[A]+ENGINEVECTOR[B];
  'GO TO' NORMAL;
  EVMINUSEV:ENGINEVECTOR[D]:=ENGINEVECTOR[A]-ENGINEVECTOR[B];
```

```
  'GO TO' NORMAL;
  MINUSEV:ENGINEVECTOR[D]:=-ENGINEVECTOR[A];
  'GO TO' NORMAL;
  BDTOEV:ENGINEVECTOR[D]:=BRICKDATA[A];
  'GO TO' NORMAL;
  EVTOBD:BRICKDATA[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  SVTOEV:'GO TO' EV[B+1];
  E0:ENGINEVECTOR[D]:=ALPHA[A];
  'GO TO' NORMAL;
  E1:ENGINEVECTOR[D]:=W[A];
  'GO TO' NORMAL;
  E2:ENGINEVECTOR[D]:=PSTATIC[A];
  'GO TO' NORMAL;
  E3:ENGINEVECTOR[D]:=PTOTAL[A];
  'GO TO' NORMAL;
  E4:ENGINEVECTOR[D]:=TSTATIC[A];
  'GO TO' NORMAL;
  E5:ENGINEVECTOR[D]:=TTOTAL[A];
  'GO TO' NORMAL;
  E6:ENGINEVECTOR[D]:=VELOCITY[A];
  'GO TO' NORMAL;
  E7:ENGINEVECTOR[D]:=AREA[A];
  'GO TO' NORMAL;
  EVTOSV:'GO TO' SV[B+1];
  S0:ALPHA[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  S1:W[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  S2:PSTATIC[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  S3:PTOTAL[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  S4:TSTATIC[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  S5:TTOTAL[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  S6:VELOCITY[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL;
  S7:AREA[A]:=ENGINEVECTOR[D];
  'GO TO' NORMAL
  'END' BRICK 22;

B24:
  'BEGIN' 'COMMENT'******BRICK 24-INTAKE MOMENTUM DRAG.PRODUCES INTAKE

  MOMENTUM DRAG AS ENGINEVECTOR[D]******;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B24')');
  ENGINEVECTOR[D]:=(W[A]*VELOCITY[A])/GC;
  'GO TO' NORMAL
  'END' BRICK 24;

B25:
  'BEGIN' 'COMMENT'******BRICK 25-OFF-DESIGN CON-DI NOZZLE-GIVEN
  S.V.[A](NOZZLE INLET) OR A MINIMUM OF ALPHA[A],W[A],PTOTAL[A] AND
  TTOTAL[A], ALSO PSTATIC[0](AMBIENT), EITHER AREA[B+1](EXIT) OR
  AREA[B]/AREA[B+1](THROAT/EXIT) AS BRICKDATA[C], AND CD AND CT AS
  BRICKDATA[C+1] AND [C+2],CALCULATES S.V.[B] AND [B+1]. IF AREA[B+1]
  IS SPECIFIED, A JUMP EXIT OCCURS IF THIS IS INCOMPATIBLE WITH GIVEN
  INLET CONDITIONS. GROSS THRUST IS CALCULATED AS ENGINEVECTOR[D]*****;
  'REAL' DELTAH,ABP1,TEST,MB;
  'BOOLEAN' RATIOGIVEN;
  'IF' CHECK 'THEN' WRITE TEXT('('''('2C')'B25')');
  'IF' PTOTAL[A]<PSTATIC[0] 'THEN'
    'BEGIN' WRITE TEXT('('''('2C')'JET%VELOCITY%IMAGINARY'
    'GO TO' JUMP
    'END';
  RATIOGIVEN:='FALSE';
  ALPHA[B+1]:=ALPHA[B]:=ALPHA[A];
  W[B+1]:=W[B]:=W[A];
  PTOTAL[B+1]:=PTOTAL[B]:=PTOTAL[A];
  TTOTAL[B+1]:=TTOTAL[B]:=TTOTAL[A];
  S111217(B,B,0);
  'IF' AREA[B+1]<0 'THEN'
    'BEGIN' RATIOGIVEN:='TRUE';
    AREA[B+1]:=AREA[B]/BRICKDATA[C]
    'END'
  'ELSE' 'IF' AREA[B]>AREA[B+1] 'THEN'
    'BEGIN' WRITE TEXT('('''('2C')'GIVEN%EXIT%AREA<REQUIRED%THROAT
    %AREA')');
    'GO TO' JUMP
    'END';
  S111217(B+1,B+1,-2);
  'IF' RATIOGIVEN 'THEN'
  'GO TO' 'IF' PSTATIC[0]>PSTATIC[B+1] 'THEN' SUBSONIC 'ELSE' SUPER
  SONIC;
  'IF' PSTATIC[0]>PSTATIC[B+1] 'THEN'
    'BEGIN' WRITE TEXT('('''('2C')'GIVEN%EXIT%AREA%REQUIRES%EXIT%
    PRESSURE%BELOW%AMBIENT')');
    'GO TO' JUMP
    'END';
  SUPERSONIC:S111217(B+1,B+1,2);
  S10(B+1,-1,B+2);
  'IF' PSTATIC[B+2] 'GE' PSTATIC[0] 'THEN' 'GO TO' FINISH;
  PSTATIC[B+1]:=PSTATIC[0];
  ABP1:=AREA[B+1];
  TEST:=0.0001*ABP1;
  TSTATIC[B+1]:=TTOTAL[B+1]/1.03;
  SHOCKINDIV:DELTAH:=ENTH(ALPHA[B+1],TTOTAL[B+1])-
  ENTH(ALPHA[B+1],TSTATIC[B+1]);
  VELOCITY[B+1]:=SQRT(TWOGCJ*DELTAH);
  AREA[B+1]:=0.666808*W[B+1]*TSTATIC[B+1]/(PSTATIC[B+1]*VELOCITY[B+1]);
  'IF' ABS(AREA[B+1]-ABP1)>TEST 'THEN'
    'BEGIN' TSTATIC[B+1]:=TTOTAL[B+1]-(TTOTAL[B+1]-TSTATIC[B+1])*
    (AREA[B+1]/ABP1)↑2;
    'GO TO' SHOCKINDIV
    'END';
  PTOTAL[B+1]:=PSTATIC[B+1]*PRES(ALPHA[B+1],TTOTAL[B+1])/
  PRES(ALPHA[B+1],TSTATIC[B+1]);
  'GO TO' FINISH;
  SUBSONIC:PSTATIC[B+1]:=PSTATIC[0];
  S5(ALPHA[B+1],PSTATIC[B+1]/PTOTAL[B+1],TSTATIC[B+1],TTOTAL[B+1]);
  DELTAH:=ENTH(ALPHA[B+1],TTOTAL[B+1])-ENTH(ALPHA[B+1],TSTATIC[B+1]);
  VELOCITY[B+1]:=SQRT(TWOGCJ*DELTAH);
```

```
  AREA[B+1]:=0.666808*W[B+1]*TSTATIC[B+1]/(PSTATIC[B+1]*VELOCITY
  [B+1]);
  AREA[B]:=AREA[B+1]*BRICKDATA[C];
  S111217(B,B,-2);
  FINISH:AREA[B+1]:=AREA[B+1]/BRICKDATA[C+1];
  ENGINEVECTOR[D]:=((W[B+1]*VELOCITY[B+1])/GC+144.0*AREA[B+1]*
  (PSTATIC[B+1]-PSTATIC[0]))*BRICKDATA[C+2];
  'GO TO' NORMAL
  'END' BRICK 25;


B26:
  'BEGIN' 'COMMENT'******BRICK 26-SUPPLEMENTARY CODEWORD-FOR USE
  WITH BRICKS 27 OR 34. MEANING OF PARAMETERS IS:-
  BRICK 27:-
  A=EXIT PRESS. S.V. NO.
  B=TYPE NO(SEE BRICK 27)
  C=ACCESSORY WORK (HP) B.D.NO.
  D=DH/T(HP) E.V. RESULT NO.
  E=COMPRESSOR WORK(HP) E.V. DATA NO.
  BRICK 34:-
  A=HP COMPRESSOR INLET S.V. NO.
  B=FUEL FLOW IN MAIN COMBUSTOR E.V. NO.
  C=LAMDAA(ESTIMATED) B.D.NO.
  D=WB/WT(ESTIMATED),THENDH/TB E.V. NO.
  E=FUEL FLOW IN BYPASS DUCT E.V. NO.*******;
  'IF' CHECK 'THEN' WRITE TEXT('('''('2C')'B26')');
  AA:=A;
  BB:=B;
  CC:=C;
  DD:=D;
  EE:=E;
  'GO TO' NORMAL
  'END' BRICK 26;

B27:
  'BEGIN' 'COMMENT'******BRICK 27-TWO TURBINES IN SERIES-
  GIVEN S.V.[A] AND VELOCITY[B+1], ACCESSORY
  WORK(L.P.) AND OVERALL ADIABATIC EFFICIENCY AS BRICKDATA
  [C] AND [C+1], L.P. COMPRESSOR WORK AS ENGINEVECTOR[E],
  AUXILIARY CODEWORD(BRICK 26) TO SET AA,BB,CC,DD,EE,
  ACCESSORY WORK(H.P.) AND H.P. ADIABATIC EFFICIENCY AS BRICKDATA
  [CC] AND [CC+1], THE EXIT PLANE AS AA, TYPE NO.(=1 FOR TWIN SPOOL,
  2 FOR FREE L.PL) AS BB, H.P. COMPRESSOR WORK AS
  ENGINEVECTOR[EE], CALCULATES S.V.[B] AND [B+1], DELTAH(HP)/TA AS
  ENGINEVECTOR[DD], DELTAH(LP)/TB AND SHP(LP) AS ENGINEVECTOR
  [D] AND [D+1]******;
  'REAL' FUEL,TSBP1,DELTA1,DELTA2,DELTA3,DELTA4,DELTA5,
  DELTA6,DELTA7,DELTA8,DELTA9,DELTA0,TDBP1,TB;
  'IF' CHECK 'THEN' WRITE TEXT('('''('2C')'B27')');
  FUEL:=ALPHA[B+1]:=ALPHA[B]:=ALPHA[A];
  W[B+1]:=W[B]:=W[A];
  PSTATIC[B+1]:=PSTATIC[AA];
  S5(FUEL,PSTATIC[AA]/PTOTAL[A],TSBP1,TTOTAL[A]);
  DELTA1:=ENTH(FUEL,TSBP1)-ENTH(FUEL,TTOTAL[A]);
  DELTA2:=VELOCITY[B+1]↑2/TWOGCJ;
  DELTA3:=DELTA1+DELTA2;
  DELTA4:=DELTA3*BRICKDATA[C+1];
  S2(FUEL,TTOTAL[A],DELTA3,TDBP1);
  PTOTAL[B+1]:=PSTATIC[B+1]*PRES(FUEL,TDBP1)/PRES(FUEL,TSBP1);
  S2(FUEL,TTOTAL[A],DELTA4,TTOTAL[B+1]);
  S16(B+1);
  'IF' BB=1 'THEN' DELTA7:= ENGINEVECTOR[E]+BRICKDATA[C]
  'ELSE' 'IF' BB=2 'THEN' DELTA7:=0
  'ELSE'
    'BEGIN' WRITE TEXT('('''('2C')'INCORRECT%TURBINE%TYPE%
    NUMBER')');
    'GO TO' NEXTPROGRAMME
    'END';
  DELTA5:=-(ENGINEVECTOR[EE]+BRICKDATA[CC])/W[A];
  DELTA6:=DELTA5/BRICKDATA[CC+1];
  S2(FUEL,TTOTAL[A],DELTA6,TB);
  PTOTAL[B]:=PTOTAL[A]*PRES(FUEL,TB)/PRES(FUEL,TTOTAL[A]);
  S2(FUEL,TTOTAL[A],DELTA5,TTOTAL[B]);
  'IF' VELOCITY[B]>0 'THEN' S16(B) 'ELSE' 'IF' AREA[B]>0 'THEN'
  S111217(B,B,-1);
  DELTA8:=DELTA5-DELTA4;
  DELTA9:=DELTA8*W[B];
  DELTA0:=DELTA9-DELTA7;
  ENGINEVECTOR[DD]:=-DELTA5/TTOTAL[A];
  ENGINEVECTOR[D]:=DELTA8/TTOTAL[B];
  ENGINEVECTOR[D+1]:=DELTA0*2.54671;
  'GO TO' NORMAL
  'END' BRICK 27;

B30:
  'BEGIN' 'COMMENT'******BRICK 30-FILLS IN STATION VECTOR.REQUIRES
  ONLY STATION NUMBER,A******;
  'IF' CHECK 'THEN' WRITE TEXT ('('''('2C')'B30')');
  'GO TO' 'IF' W[A] > 0 'THEN' WK 'ELSE' 'IF' PSTATIC[A] > 0 'THEN'
  PSK 'ELSE' WPSNK;
  WPSNK:S2(ALPHA[A],TTOTAL[A],-VELOCITY[A]↑2/TWOGCJ,TSTATIC[A]);
  PSTATIC[A]:=PTOTAL[A]*PRES(ALPHA[A],TSTATIC[A])/
  PRES(ALPHA[A],TTOTAL[A]);
  W[A]:=PSTATIC[A]*AREA[A]*VELOCITY[A]/(0.666808*TSTATIC[A]);
  'GO TO' NORMAL;
  WK:'GO TO' 'IF' PSTATIC[A] > 0 'AND' AREA[A] > 0 'THEN' PSAK 'ELSE'
  'IF' AREA[A]>0 'THEN' AK 'ELSE' 'IF' PSTATIC[A]>0 'THEN'
  WPSK 'ELSE' PSANK;
  PSANK:S16(A);
  'GO TO' NORMAL;
  WPSK:'IF' VELOCITY[A]#0 'THEN'
  AREA[A]:=0.666808*W[A]*TSTATIC[A]/(PSTATIC[A]*VELOCITY[A]);
  'GO TO' ALLCALC;
  AK:S111217(A,A,-1);
  'GO TO' NORMAL;
  PSAK:VELOCITY[A]:=0.666808*W[A]*TSTATIC[A]/(PSTATIC[A]*AREA[A]);
  'GO TO' ALLCALC;
  PSK:W[A]:=PSTATIC[A]*AREA[A]*VELOCITY[A]/(0.666808*TSTATIC[A]);
  ALLCALC:S2(ALPHA[A],TSTATIC[A],VELOCITY[A]↑2/TWOGCJ,TTOTAL[A]);
  PTOTAL[A]:=PSTATIC[A]*PRES(ALPHA[A],TTOTAL[A])/PRES(ALPHA[A],
  TSTATIC[A]);
  'GO TO' NORMAL
  'END' BRICK 30;
```

```
    CODEWORD F,WHICH MUST BE BRICK 37,IF TTOTAL[B] HAS
    NOT BEEN CALCULATED CORRECTLY YET,OTHERWISE CALCULATES
    S.V.[B+1]******;
    'REAL' DELTAH;
    'IF' CHECK 'THEN' WRITE TEXT ('('('2C')'B38')');
    'IF' ABS(TTOTAL[B]-TTOTAL[B+1]) > 0.0001*TTOTAL[B] 'THEN'
    'GO TO' JUMP;
    BRICKDATA[C+1]:=-BRICKDATA[C+1];
    ALPHA[B+1]:=ALPHA[B];
    W[B+1]:=W[B];
    AREA[B+1]:=AREA[B];
    PTOTAL[B+1]:=BRICKDATA[C+1]*PTOTAL[B]-BRICKDATA[C+2];
    DELTAH:=W[A]*(ENTH(ALPHA[A],TTOTAL[A])-ENTH(ALPHA[A],TTOTAL[A+1]))/
    W[B];
    S2(ALPHA[B],TTOTAL[B],DELTAH,TTOTAL[B+1]);
    'IF' AREA[B+1]>0 'THEN' S111217(B+1,B+1,-1);
    'GO TO' NORMAL
    'END' BRICK 38;

  'END'************OF TURBOCODE MASTER PROGRAM************;

****
```