AD735293

AFFDL-TR-71-52
VOLUME III

COMBAT OPTIMIZATION AND ANALYSIS PROGRAM – COAP

VOLUME III: PROGRAMMER'S MANUAL

D. S. HAGUE, R. T. JONES, AND C. R. GLATT

AEROPHYSICS RESEARCH CORPORATION
BELLEVUE, WASHINGTON

TECHNICAL REPORT AFFDL-TR-71-52, VOLUME III

MAY 1971

DDC
RECEIVED
JUN 1
A

Approved for public release; distribution unlimited

AIR FORCE FLIGHT DYNAMICS LABORATORY
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
Springfield Va. 22151

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)	2a. REPORT SECURITY CLASSIFICATION
Aerophysics Research Corporation P. O. Box 187 Bellevue, Washington 98009	Unclassified 2b. GROUP N/A

3. REPORT TITLE

Combat Optimization and Analysis Program - COAP, Volume III: Programmer's Manual

4. DESCRIPTIVE NOTES *(Type of report and inclusive dates)*

Final Report

5. AUTHOR(S) *(First name, middle initial, last name)*

D. S. Hague, R. T. Jones, and C. R. Glatt

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
April 1971	488	None

8a. CONTRACT OR GRANT NO.	9a. ORIGINATOR'S REPORT NUMBER(S)
F33615-70-C-1036 b. PROJECT NO. 1431 c. d.	None 9b. OTHER REPORT NO(S) *(Any other numbers that may be assigned this report)* AFFDL-TR-71-52, Volume III

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
None	Air Force Flight Dynamics Laboratory Attn: FXG Wright-Patterson Air Force Base, Ohio

13. ABSTRACT

COAP is a program for the simulation and optimization of combative and cooperative two-vehicle flight paths. It includes single vehicle flight path problem capability as a subcase. Considerable emphasis is placed on the use of modern optimization.

The program has the ability to perform trajectory optimization by the variational steepest-descent method including search for optimal initial conditions; search for optimal arc (stage) lengths; constraints defined at terminal point, intermediate corners (stage points), or along the path; and optimum parameter (design variable) values. The program can solve two system (vehicle) problems with or without reacting feedback from the second system (vehicle).

Alternatively, the program may be used to apply the direct multivariable search approach to trajectory optimization. A variety of multivariable search algorithms are available in this mode including elemental perturbation (one parameter at a time); organized first- and second-order methods, and randomized methods. A method of solution for problems exhibiting multiple extremals and a procedure for the location of saddle points is also included.

Point mass equations of motion for a two-vehicle system are incorporated in the COAP program. Motion takes place about a rotating oblate planet having up to four harmonics in its gravitational field, non-uniform atmosphere (1959 or 1962 ARDC), and winds. Auxiliary computations for aerodynamic heating are included. The vehicles may have arbitrary and independent aerodynamic and propulsive characteristics. If desired two independent sets of planetary characteristics may be employed.

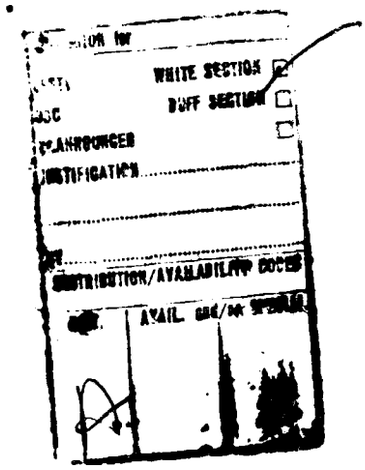
DD FORM 1 NOV 68 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.

Unclassified

Security Classification

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.



Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AIR FORCE: 8-10-71/150

13. Abstract (continued)

Combative logic is defined in terms of vehicle relative states. The logic defines feedback control on the basis of relative state. The COAP program may utilize variational optimization procedures to determine optimal open loop control against a reacting opponent employing feedback control defined by combat logic. Alternately, the combative feedback control logic may be parameterized permitting the use of multivariable search for the definition of optimal feedback control parameters against a reacting opponent. Finally, by parameterization of both vehicle feedback control logic, a "mini-max" mode of operation may be employed in which a solution is obtained by multivariable search for a saddle point.

All capabilities described are available in a single general purpose FORTRAN IV digital computer program developed for the CDC 6600 computer. This volume, Volume III, presents a programmer's manual containing instructions regarding

(a) Computer requirements

(b) Program structure

(c) Detailed subroutine descriptions

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Parameter optimization, non-linear programming, steepest-descent, "mini-max" solution, saddle point solution, multiple extremal solution						

AFFDL-TR-71-52
VOLUME III

COMBAT OPTIMIZATION AND ANALYSIS PROGRAM – COAP

VOLUME III: PROGRAMMER'S MANUAL

D. S. HAGUE, R. T. JONES, AND C. R. GLATT

AEROPHYSICS RESEARCH CORPORATION

Approved for public release; distribution unlimited

FOREWORD

The research project outlined in this report was completed in the period from September 1969 to December 1970 under the sponsorship of the Air Force Flight Dynamics Laboratory, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio. This report was prepared by Aerophysics Research Corporation, Bellevue, Washington, under the United States Air Force Contract F33615-70-C-1036. Mr. B. R. Benson and Mr. David T. Johnson of the Air Force Flight Dynamics Laboratory were the cognizant Air Force representatives for the study.

This report was authored by Mr. R. T. Jones, D.S. Hague, and C.R. Glatt of Aerophysics Research Corporation. The report was prepared and organized by Mrs. Jane Yonke of Aerophysics Research Corporation. The report and study benefit directly from a number of previous government-sponsored research studies including:

USAF Contract AF33(616)-6848,	for trajectory equation and program development
USAF Contract AF33(657)-8829,	for development of the variational optimization procedure and program
USAF Contract AF33(615)-3932,	for extension of the variational optimization program
NASA Contract NAS 2-4507,	for development of the parameter optimization procedure and program
NASA Contracts NAS 2-3691, NAS 2-4880, NAS 1-9936, and NAS 3-13331	for extension of the optimization procedures

This project has resulted in considerable extension of the previously available AFFDL generalized steepest-descent computer program. Notable extensions consist of the addition of equations of motion for a second vehicle, addition of self-contained combative logic, extension of the variational steepest-descent procedure to situations involving a reacting opponent, addition of parameter optimization capability by multivariable search, and the development of a saddle point search procedure for the solution of parametrically defined trajectory or combat performance problems.

The study results are reported in four volumes as follows:

Volume I - Trajectory, Combat and Variational Optimization Formulation

Volume II - Program User's Manual

Volume III - Programmer's Manual for Trajectory, Combat, and Variational Optimization Subprograms

Volume IV - Programmer's Manual for Parameter Optimization
Subprogram AESOP

This report was submitted by the authors March 1, 1971.

This technical report has been reviewed and is approved.

PHILIP P. ANTONATOS
Chief, Flight Mechanics Division
Air Force Flight Dynamics Laboratory

ABSTRACT

A program for trajectory optimization by the variational steepest-descent is described in detail. The program capability includes search for optimal initial conditions; search for optimal arc (stage) lengths; constraints defined at terminal point, intermediate corners (stage points), or along the path; payoff function at terminal point or intermediate corner (stage point); search for optimum parameter (design variable) values; and two system (vehicle) problems with or without reacting feed-back from the second system (vehicle).

The program also incorporates an alternative direct multivariable search approach to trajectory optimization employing a variety of multivariable search algorithms including elemental perturbation (one parameter at a time), organized first- and second-order methods, and randomized methods. A method for solution for problems exhibiting multiple extremals and a procedure for the location of saddle points is also included in the program.

Point mass equations of motion for a two-vehicle system are available in the program. Motion takes place about a rotating oblate planet having up to four harmonics in its gravitational field, non-uniform atmosphere (1959 or 1962 ARDC), and winds. Auxiliary computations for aerodynamic heating are included. The vehicles may have arbitrary and independent aerodynamic and propulsive characteristics.

Combative logic defined in terms of vehicle relative states is available in the program. The logic defines feedback control on the basis of relative state. Variational optimization procedures may be employed to determine optimal open loop control against a reacting opponent employing feedback control defined by combat logic. Alternately, the combative feedback control logic can be parameterized permitting the use of multivariable search for the definition of optimal feedback control parameters against a reacting opponent. By parameterization of both vehicle feedback control logic, a "mini-max" situation capable of solution by multivariable search for a saddle point can be considered.

All capabilities described are available in a single general purpose FORTRAN IV digital computer program developed for the CDC 6600 computer.

TABLE OF CONTENTS

Section		Page
I	INTRODUCTION	1
II	COMPUTER AND SYSTEM REQUIREMENTS	3
III	PROGRAMMING CONCEPTS	4
	1. The Use of COMMON	4
	2. Table and Table Usage	4
	3. Symbolic Input	4
	4. Trajectory Printing Method	5
	5. Tape Usage	5
	6. Overlay Listing	7
	7. Program Organization	12
	8. Program Generation, Modification, and Execution	13
	9. Data Format	20
	10. Table Format	22
	11. Brief Write-Up on System Routines	23
	12. ENCODE/DECODE	25
IV	MAIN PROGRAM AND SUBROUTINE MAIN2	27
	1. MAIN and MAIN2	27
	2. BDATA1 - Block Data Subroutine	29
	3. BDATA2 - Block Data Subroutine	29
	4. BDATA3 - Block Data Subroutine	29
	5. BDATA4 - Block Data Subroutine	29
	6. BDATA5 - Block Data Subroutine	29
	7. BDATA6 - Block Data Subroutine	29
	8. BDATA7 - Block Data Subroutine	29
	9. SPRANG - Random Number Generator	29
	10. CHAIN - Overlay Control Program	30
	11. MSGONE - Combat Message Subroutine, Vehicle 1	32
	12. MSGTWO - Combat Message Subroutine, Vehicle 2	32
V	PROGRAM MAIN1 AND SUBROUTINE MAIN12	33
	1. READA and READA2 - Input Routines for Vehicle 1 and Vehicle 2 Data	37
	2. DORDER - Directory Order Output Routine	45
	3. PACKL - Pack Routine	46
	4. DSERCH and DSERCH2 - Directory Search Routine for Subscripts	47
	5. STOP1 - General Stop Routine	49
	6. EXERR - Error Routine	51
	7. TSRCH and TSRCH2 - Directory Search for Table Subscript	52
	8. READB and READB2 - Binary Stage Data Input Routine	54
	9. BAESOP - Parameter Optimization Input Subroutine	56
	10. DIPLAC - Integer Shift Routine	57
	11. DEF and DEF2 - Heading and Page Eject	59
	12. TABRE and TABRE2 - Table Dimension Subscript Routines	60

Section		Page
	13. LINES and LINES2 - Lines Accounting Routines	62
	14. PACBCD - Packs Six Character Words	64
	15. PACKR - Packs BCD Characters	66
	16. READ31 - Binary Conversion Routine	68
	17. SVI and SVI2 - Block Save Routines	70
	18. BIBLOCK - Data Output in Blocks	72
	19. SHELL - Numeric Sorting Subroutine	74
VI	PROGRAM EXE AND SUBROUTINE EXE2	75
	1. DIFEQ - Differential Equation Selector	82
	2. MANTGT and MANTGT2 - Maneuvering Target	84
	3. CTVS and CTVS2 - Control Variable Routine for EXE	90
	4. IFCS - In-Flight Constraints	93
	5. PARTS - Partial Derivatives	96
	6. LINCOM and LINCOM2 - Linear Combination Routines	104
	7. SLACK and SLACK2 - Slack Variable Routines	107
	8. PENAL and PENAL2 - Penalty Function Routine	109
	9. PLTS and PLTS2 - Data Cathering Routine	112
	10. OBSFUN and OBSFUN2 - Observation Function Routine	115
	11. FILTER and FILTER2 - Repeater Routines for h-Transformation	120
	12. STGTST and STGTST2 - Stage Testing Routine	122
	13. EXTRAN and EXTRAN2 - Driver for h-Transformation	125
	14. INTGRT, INTGRT2 and INTGRTR - Interface for Integration Routine	128
	15. CODES and CODES2 - Code Print Routines	131
	16. ITEMS and ITEMS2 - Variable Print Routines	133
	17. COMBAT and COMBAT2 - Combat Control Routine	135
	18. MIMINF and MIMINF2 - Integration Routine	143
	19. TIMID and TIMID2 - Step Function Routines for Time Points	147
	20. VALUES and VALUES2 - Value Print Routines	149
	21. MISCUT and MISCUT2 - Abortion Routine	151
	22. ONLINED - On Line Display Routine	153
	23. DIFEQ1 and DIFEQ2 - Point Mass Equations of Motion	154
	24. DIFEQ3 - Dummy Subroutine	164
	25. TLUREV - Two Dimensional Table Look-Up Routine	165
	26. ACOS - Arc Cosine Routine	167
	27. ASIN - Arc Sine Routine	169
	28. TLU and TLU2 - Two Dimensional Table Look-Up Routine	170
	29. ATAN2 - Arctangent Routine	172
	30. TLU1 - Two Dimensional Table Look-Up Routine	174
	31. TIMREV and TIMREV2 - Time Point Collection Routine	177
	32. PSUBR - Evaluation of Partial Routine	179
	33. PRPACK - Blocking Routine for Partials	182
	34. FLUSH1 and FLUSH12 - Buffer Flush Routine for PRPACK	184
	35. SETGRD - Paper Plot Grid Size Routine	186
	36. PAPERP - Printer-Plot Control Routine	187

Section		Page
37.	DETECT and DETECT2 - Sensor Control Program	188
38.	ROLE1 and ROLE2 - Role Selection Subprogram	190
39.	HLIMIT and HLIMIT2 - Minimum Altitude Constraint Routine	194
40.	ANGLES and ANGLES2 - Relative Angular Orientation Routines	196
41.	CRATE and CRATE2 - Finite Control Rate Routine	199
42.	TIM001 and TIM0012 - Tabular Time Point Routine	204
43.	DEQPRE and DEQPRE2 - Equation of Motion Pre-Data Initialization	206
44.	FIRFUN and FIRFUN2 - Fire Control Subprogram	207
45.	GAM91 and GAM92 - Flight through Vertical Routine	211
46.	DEQINI and DEQINI2 - Equation of Motion Post-Data Initialization	213
47.	DEQBCI and DEQBCI2 - Derivative Calculation Before Control Definition	214
48.	FPPS and FPPS2 - Flight Plan Programmer	215
49.	ONETWO - Transformation of Selected Vehicle 2 Variable to Vehicle 1 COMMON	218
50.	CTLITR and CTLITR2 - Control Dependent Derivative Calculation	219
51.	DEQACI and DEQACI2 - Derivative Calculation after Control Definition	220
52.	FPPG and FPPG2 - Gamma Command Flight Plan Programmer	222
53.	DEQSIP and DEQSIP2 - Derivative Evaluation Initial Point	224
54.	DEQCOD and DEQCOD2 - Trajectory Code Print	225
55.	DEQVAL and DEQVAL2 - Trajectory History Print	226
56.	DEQIV and DEQIV2 - Integrated Variable Specification	227
57.	DEQHT and DEQHT2 - Trajectory h-Transformation Subroutines	228
58.	ERROR and ERROR2 - General Table Error Routine	229
59.	PTBEQN and PTBEQN2 - Driver Routines for Equations	231
60.	TWOONE - Transformation of Selected Vehicle 1 to Vehicle 2 COMMON	234
61.	IZERO and IZERO2 - Packs Non-Zero Numbers	235
62.	PPLNLN - Main Paper Plot Routine	237
63.	SENSOR and SENSOR2 - Vehicle Sensor Routines	239
64.	VISION and VISION2 - Pilot Vision Routines	242
65.	PASSV1 and PASSV2 - Passive Tactics Routine	244
66.	DEFEN1 and DEFEN2 - Defensive Tactics Routine	246
67.	EVADE1 and EVADE2 - Evasive Tactics Routine	249
68.	OFFEN1 and OFFEN2 - Offensive Tactic Routine	252
69.	ATTAC1 and ATTAC2 - Attacking Tactics Routine	255
70.	OALPBA and OALPBA2 - Subprogram for Instantaneous Control Vector Iteration	258
71.	HIHO and HIHO2 - N-Dimensional Table Call Routine	262
72.	TMTX - Transformation Matrix Routine	264
73.	TRNPOS - A 3 x 3 Matrix Transpose Routine	267

Section		Page
	74. MULT31 - A Matrix Multiplication Routine	268
	75. HETS and HETS2 - Heating Computations	269
	76. TFFS and TFFS2 - Single Engine Thrust and Fuel Flow	276
	77. SACS and SACS2 - Aerodynamic Routines	279
	78. LATS and LATS2 - Geodetic-Geocentric Conversion	288
	79. ATMS and ATMS2 - Atmosphere Selector	290
	80. GVSP and GVSP2 - Gravitational Routine	291
	81. ANITR and ANITR2 - Throttle Dependent Derivative and Thrust Vector Calculation	293
	82. BAITR and BAITR2 - Bank Angle Dependent Derivative Calculation	294
	83. ASRCH and ASRCH2 - Directory Search Routines for BCD Characters	295
	84. GRIDXY - Paper Plot Grid Routine	297
	85. PLCPTS - Paper Plot Point Placing Program	299
	86. IPICK - Random Tactic Selector	300
	87. DEFEN11 and DEFEN21 - First Defensive Tactic	301
	88. FIXEDR and FIXEDR2 - Fixed Role Selection Routine	307
	89. EVADE11 and EVADE21 - First Evasive Tactic	308
	90. OFFEN11 and OFFEN21 - Lag-Pursuit Offensive Tactic	312
	91. OFFEN12 and OFFEN22 - Lead-Pursuit Offensive Tactic	313
	92. OFFEN13 and OFFEN23 - Reference Vector Offensive Tactic	314
	93. ATTAC11 and ATTAC21 - First Attacking Tactic	315
	94. ATTAC12 and ATTAC22 - Second Attacking Tactic	316
	95. CTLOPT - Internal AESOP Optimization Loop	318
	96. NDTLU - N-Dimensional Table Lookup Routine	320
	97. CHEMP and CHEMP2 - Chemical State Computation	323
	98. CONV - Non-Linear Equation Solver	326
	99. TFFM and TFFM2 - Multiengine Thrust and Fuel Flow	329
	100. ATMS59 and ATMS592 - 1959 Atmosphere Calculation Routine	333
	101. ATMS62 and ATMS622 - 1962 Atmosphere Calculation Routine	335
	102. PUT - Character Manipulation Routine	337
	103. GET - Character Manipulation Routine	338
	104. OPTBA and OPTBA2 - Bank-angle Iteration Routine	339
	105. IMAINOP - Internal Parameter Optimization Interface Routine	341
	106. ISELECT - Interval Search Selection Routine	343
	107. MULT33 - Matrix Multiplication Routine	344
VII	PROGRAM CTLS	345
	1. CTLS1 - Original Control System	347
	2. CTLS2 - Arbitrary Control System	357
	3. DISPLAY - Console Display Routine	362
	4. SEARCH - Search Routine	363

Section		Page
	5. CARDS - Restart Cards Routine	367
	6. DALCAL - Delta Alpha Calculation	369
	7. KCALC - Step Size Logic Routine for Control System 1	374
	8. INVERT - $I_{\psi\psi}$ Inversion	379
	9. DECIDE - Driver for Decision Routines	384
	10. OFFSW - Display Drop Routine	385
	11. TLUU - Two-Dimensional Table Look-Up Routine	386
	12. UNBLOCK - Unblocking Routine for λ's	388
	13. SUMOLA - Linear Combination Routine for λ's	390
	14. PLOT - Point Plot Subroutine	392
	15. MATINV - Matrix Inversion Routine	398
	16. DECID3 - Step Size Routine for Control Systems	400
	17. DECID2 - Arbitrary Decision Routine	406
	18. PACK - Integer Word Conversion Routine	407
	19. WNORM - Weighting Matrix Norm Calculation	409
	20. UPDK - Convergence Control Routine	411
VIII	PROGRAM REV	428
	1. CTVSR - Control Variable Routine for REV	434
	2. ADJEQ - Adjoint Equations Routine	436
	3. UNPART - Unblocking Routine for Partials	442
	4. MIMINR - Integration Routine for REV	444
	5. WMA - Weighting Matrix Routine	446
	6. DALPACK - Blocking Routine for λ's	453
	7. FLUSH - Buffer Flush Routine	455
	8. IZUNPK - Switch Testing Routine	456
IX	PROGRAM GRAPH	458
	1. Mapping Routines	459
	2. Arrow, Line, and Point Plotting Routines	459
	3. Character Plotting Routines	459
	4. Absolute Plotting Routines	459
	5. Utility Routines	460
	6. Internal Routines	460
	7. PAPLT - Parameter Collection Routine	472
	8. PRPLT - Partials Collection Routine	474
	9. PLTCUR - Microfilm Plotting Routine	476
	10. Dummy Plot Routines	479
X	DIRECTORY	482
	1. CTAE - Parameter Optimization Control Program	483
XI	ALPHABETIC INDEX OF SUBROUTINES	486

SECTION I

INTRODUCTION

This report describes a generalized digital computer program for the simulation and/or optimization of arbitrarily defined vehicle flight paths. Two simultaneously coupled three-dimensional point mass trajectories about a rotating oblate planet having a multi-layered atmosphere may be employed. Levels of reduced complexity varying from the generalized problem to straightforward planar point mass, single vehicle, vacuum trajectories may be studied with the aid of the program.

The program itself has evolved over a period of some ten years, mainly on the basis of four Air Force Flight Dynamics Laboratory (AFFDL) sponsored contractor research studies. The program has also been supported by National Aeronautics and Space Administration studies at Ames Research Center, Langley Research Center, and the Manned Spacecraft Center and has received wide distribution throughout the aerospace industry.

In addition to the point mass capability, compatible trajectory equation options of increased complexity up to and including a six-degree-of-freedom single vehicle option employing generalized vehicle and planetary characteristics and a generalized trajectory error and dispersion analysis are available from AFFDL.

PURPOSE

The program reported in this document extends the existing AFFDL single-vehicle point mass program to the two-vehicle point mass problem. Specifically, the original program has been extended to *provide a generalized two-vehicle (one-on-one) combative engagement simulation and optimization capability.* The combative encounter may be defined at several levels of complexity short of differential game formulation including:

OPTION (A): Self-contained role and tactic selection based on relative vehicle states

OPTION (B): Parameterization of one vehicle's role and tactic selection rules followed by the application of multivariable search procedures to obtain the optimal parameter values. This option defines optimal parameters against a specified opponent employing fixed combat logic parameters.

OPTION (C): Parameterization of both opponent's role and tactic selection rules followed by the application of a multivariable saddle point search technique. This option defines a "mini-max" optimal procedure for opponents employing variable combat logic parameters.

OPTION (D): Open loop, continuous control optimization by the variational calculus against an opponent performing a pre-specified maneuver, the "maneuvering target" option.

OPTION (E): Open loop, continuous control optimization by the variational calculus against a reacting opponent employing fixed parameters and self-contained combat tactics.

It should be noted that the formulation and program include as subcases two-vehicle cooperative problems. This leads to

OPTION (F): Cooperative two-vehicle parametric control

OPTION (G): Cooperative two-vehicle open loop continuous control

These last two options permit the optimization of two-vehicle rendezvous problems and are equally applicable to aircraft or spacecraft problems. Single-vehicle problems may also be studied by means of the program. In this reduced mode, both parametric and variational optimization formulations can be employed.

SECTION II

COMPUTER AND SYSTEM REQUIREMENTS

The combat simulation program has been written for use with the CDC 6600 series computer system using the run compiler. Except for three small subroutines, GET, PUT, and OFFSW, the program is written in CDC FORTRAN IV.

Computer resource requirements are

1. A CDC 6600 computer with 131k (decimal) core
2. A card reader
3. A line printer
4. A card punch (if restart cards are to be punched)
5. Twenty-two tape transports or a disk to simulate magnetic tape
6. A display console (This requirement is omitted when the display subroutine SCOPE is replaced by a dummy subroutine).

SECTION III

PROGRAMMING CONCEPTS

1. The Use of COMMON

Whenever possible, a variable is placed in the FORTRAN "COMMON" area. There are several reasons for this:

a. The communication between subroutines is simplified

b. The structure of the directory is simplified. Since the number of variables in COMMON is quite large, all COMMON cards are not placed in each assembly/compilation. Instead, access to any quantity in the COMMON blocks is through individual "EQUIVALENCE" statements placed in each deck of source cards. This has in a small manner reduced the number of COMMON cards in each deck.

In order that the user may better obtain the required COMMON location for a given variable, a system of COMMON subscripts is used. The first location of COMMON has a subscript of 1; the second, 2, . . . The listing of the suffix directory can be obtained by a program user if the data quantity "DLIST" is input as 1.0 to the program.

The data for vehicle one is stored in "blank" COMMON. The data for vehicle two is stored in named COMMON/COMMON2/. Blank COMMON and /COMMON2/ each use 3000 core locations. The COMMON subscript of a given variable is the same for both vehicles. For example, bank angle, "BA77D," is the sixty-fourth location of blank COMMON for vehicle one and is the sixty-fourth location of /COMMON2/ for vehicle 2.

2. Table and Table Usage

One of the usual required modifications of any program is the change of table sizes. With this in mind, a COMMON block of locations has been set aside,and the required number of cells required for each table is specified with data (check TABRE for data preparation). This requires no re-assembly or recompilation unless the total number of cells required exceeds the COMMON block. This has been set at 4000 cells for each vehicle. Table data for vehicle one is stored in numbered COMMON /5/, and table data for vehicle two is stored in numbered COMMON /52/.

3. Symbolic Input

Although the FORTRAN system itself has a system of input routines, the program does the actual translation of cards using special coded routines. Input data may be read using a system of symbols which is designed to give engineering meaning to the analyst. The symbols are internally referenced to actual locations by the use of COMMON and subscripts.

4. Trajectory Printing Method

The printing of a trajectory may be divided into five categories:

a. Initial printing - The printing of specific values of the first stage and at each subsequent major stage. Initial print is designed to print certain values which will be constant during the trajectory and serve as a reminder of what values have been used for these constants.

b. Code printing - The printing of codes which will identify the variables which are to be obtained in the coming time history print. Only those variables to be printed in the time history will have a code name printed. All subprograms being used as well as the differential equations routine will print a set of codes.

c. Time history printing - The printing of values specified at the requested points of the trajectory. If a certain variable is not desired as output, it is not printed, and other desired variables are moved in the print format accordingly.

d. Observation functions - A maximum of thirty-two variables for each vehicle may be designated as observation functions. At the end of each major stage, the time history of all observation functions are printed in tabular format. Observation functions may also be plotted on the line printer. The object of observation function printout is the construction of compact histories for the major trajectory variables.

e. Diagnostic error printing - The printing of errors detected by the program.

The entire printing is controlled to print on a page 11 x 14 inches and will print a maximum of fifty-four lines per page. Page ejection and lines control are provided by the subroutines LINES and DEF.

In addition to the above printout, all input data involved for a case may be printed on the output page preceding the computation of the first stage printout. This printout is user-controlled and will be omitted when IPNAML = 0. This print will occur on the first cycle only.

5. Tape Usage

This section will describe the tape usage other than the FORTRAN system. All modification of tapes required may be made with control cards placed in front of the program before submitting to the computer.

TAPE	EQUIPMENT	VEHICLE	USAGE
Tape 5	Disk or tape	1 & 2	Data input
Tape 6	Disk or tape	1 & 2	Printed output

TAPE	EQUIPMENT	VEHICLE	USAGE
Tape 10	Disk or tape	1	Save the α history for that valid step for use with all trials until another valid step is found.
Tape 11	Disk or tape	1	This unit is used in the reverse integration to save data for use in computing the mode shape in the control system.
Tape 12	Disk or tape	1	Partials and control variables are written on this unit for use in the reverse integration
Tape 13	Disk or tape	1	Target data is written on this tape when using the maneuvering target option.
Tape 14	Disk or tape	1 & 2	Used to save the AESOP data base for the outer AESOP parameter optimization loop.
Tape 15	Disk or tape	1	This tape is used for restart cards. Restart cards for the last completed cycle are written on this tape. The program call card may be used to automatically assign this tape to the punch file.
Tape 16	Disk or tape	1	Used to prepare the input tape in binary for use in the iterative procedure.
Tape 17	Disk or tape	1	Collects data for plotting in overlay 5, 0 (GRAPH)
Tape 18	Disk or tape	1 & 2	Save the AESOP parameter and performance history for the summary report at the end of an optimization cycle.
Tape 19	Disk or tape	1 & 2	Save the AESOP parameter and performance history for the final summary report.
Tape 20	Disk or tape	2	Same as unit 10.
Tape 21	Disk or tape	2	Same as unit 11

TAPE	EQUIPMENT	VEHICLE	USAGE
Tape 22	Disk or tape	2	Same as unit 12.
Tape 23	Disk or tape	2	Same as unit 13.
Tape 25	Disk or tape	2	Same as unit 15.
Tape 26	Disk or tape	2	Same as unit 16.
Tape 27	Disk or tape	2	Same as unit 17.
FILMPL	Disk or tape	1	Used by the CDC 280 recorder and display system.

6. Overlay Listing

The following is a list of the overlay structure of the combat simulation program including the main program, subprograms, subroutines, and system routines used. (Note: System routines appear in *italics*).

OVERLAY 00.00

1.	MAIN	2.	BDATA1
3.	BDATA2	4.	BDATA3
5.	BDATA4	6.	BDATA5
7.	BDATA6	8.	BDATA7
9.	SPRANG	10.	MAIN2
11.	CHAIN	12.	MSGONE
13.	MSGTWO	14.	*SYSTEM*
15.	*FINBIN*	16.	*ACGOER*
17.	*EXP*	18.	*ALNLOG*
19.	*SQRT*	20.	*SECOND*
21	*OUTPTC*	22.	*OVERLAY*
23.	*REWINM*	24.	*OUTPB*
25.	*SIO$*	26.	*GETBA*
27.	*KODER*	28.	*OVERLOD*

OVERLAY 01.00

1.	MAIN1	2.	READA
3.	DORDER	4.	PACKL
5.	DSERCH	6.	STOP1
7.	EXERR	8.	TSRCH
9.	READB	10.	MAIN12
11.	BAESOP	12.	DIPLAC
13.	DEF	14.	TABRE
15.	LINES	16.	PACBCD
17.	PACKR	18.	READ31
19.	SVI	20.	BIBLOCK
21.	SHELL	22.	SHELLX
23.	READA2	24.	DSERCH2
25.	TSERCH2	26.	READB2

27. TABRE2
28. SVI2
29. *BACKSP*
30. *INPUTB*
31. *INPUTC*
32. *OUTPTS*
33. *IFENDE*
34. *INPUTN*
35. *OUTPTN*
36. *INPUTS*
37. *KRAKER*

OVERLAY 02.00

1. EXE
2. DIFEQ
3. MANTGT
4. CTVS
5. IFCS
6. PARTS
7. LINCOM
8. SLACK
9. PENAL
10. PLTS
11. OBSFUN
12. FILTER
13. EXE2
14. STGTST
15. EXTRAN
16. READB
17. INTGRT
18. DEF
19. LINES
20. CODES
21. ITEMS
22. COMBAT
23. COMBAT2
24. OBSFUN2
25. MIMINF
26. MIMINF2
27. STGTST2
28. TIMID
29. VALUES
30. MISCUT
31. MISCUT2
32. ONLINED
33. DIFEQ1
34. DIFEQ2
35. DIFEQ3
36. DIFEQ4
37. DIFEQ5
38. DIFEQ6
39. STOP1
40. TSRCH
41. TLUREV
42. ACOS
43. ASIN
44. TLU
45. ATAN2
46. TLU1
47. TIMREV
48. DSERCH
49. EXERR
50. ONETWO
51. PSUBR
52. PRPACK
53. FLUSH1
54. SETGRD
55. PAPERP
56. MANTGT2
57. CTVS2
58. IFCS2
59. PARTS2
60. LINCOM2
61. SLACK2
62. PENAL2
63. PLTS2
64. FILTER2
65. EXTRAN2
66. READB2
67. INTGRT2
68. CODES2
69. ITEMS2
70. TIMID2
71. VALUES2
72. DETECT
73. ROLE1
74. HLIMIT
75. ANGLES
76. CRATE
77. DETECT
78. ROLE2
79. HLIMIT2
80. ANGLES2
81. CRATE2
82. DSERCH2
83. TIM001
84. DEQPRE
85. FIRFUN
86. GAMA91
87. DEQINI
88. DEQBCI

89. FPPS
90. CTLITR
91. DEQACI
92. FPPG
93. DEQSIP
94. DEQCOD
95. DEQVAL
96. DEQIV
97. DEQHT
98. DEQPRE
99. FIRFUN2
100. GAMA92
101. DEQINI2
102. DEQBCI2
103. FPPS2
104. CTLITR2
105. DEQACI2
106. FPPG2
107. DEQSIP2
108. DEQCOD2
109. DEQVAL2
110. DEQIV2
111. DEQHT2
112. ERROR
113. PTBEQN
114. TWOONE
115. IZERO
116. PPLNLN
117. TSRCH2
118. TLU2
119. TIMREV2
120. TIMOOI2
121. SENSOR
122. VISION
123. PASSV1
124. DEFEN1
125. EVADE1
126. OFFEN1
127. ATTAC1
128. OALPBA
129. HIHO
130. TMTX
131. TRNPOS
132. MULT31
133. SENSOR2
134. VISION2
135. PASSV2
136. DEFEN2
137. EVADE2
138. OFFEN2
139. ATTAC2
140. OALPBA2
141. HETS
142. TFFS
143. SACS
144. LATS
145. ATMS
146. GVSP
147. ANITR
148. BAITR
149. HETS2
150. TFFS2
151. SACS2
152. LATS2
153. ATMS2
154. GVSP2
155. ANITR2
156. BAITR2
157. ASRCH
158. PTBEQN2
159. GRIDXY
160. PLCPTS
161. ERROR2
162. IPICK
163. DEFEN11
164. FIXEDR
165. EVADE11
166. OFFEN11
167. ATTAC11
168. CTLOPT
169. NDTLU
170. DEFEN21
171. FIXEDR2
172. EVADE21
173. OFFEN21
174. ATTAC21
175. CHEMP
176. CONV
177. TFFM
178. ATMS59
179. ATMS62
180. CHEMP2
181. TFFM2
182 HIHO2
183. ATMS592
184. ATMS622
185. PUT
186. GET
187. ASRCH2
188. OPTBA
189. INITOP
190. BESTAF

191. IMAINOP
192. WARPS
193. PENLTY
194. OPTBA2
195. PGAIN
196. ALFLIM
197. INFVAL
198. FESET
199. SAVALF
200. PATERN
201. ISELECT
202. PSRCH
203. WEIGHT
204. OUTALF
205. OUTFUN
206. PCYCLE
207. FNEVAL
208. PFINAL
209. SUMARY
210. SECCON
211. CREEPR
212. OUTWGT
213. OUTSUM
214. RANDUM
215. BOUND
216. SECTON
217. ASVCHK
218. FESAV
219. RGEN
220. SHELL
221. FUNTYP
222. STDALF
223. *INPUTB*
224. *BACKSP*
225. *SINCOS*
226. *RBAIEX*
227. *IFENDF*
228. *OUTPTS*
229. *SCOPE*
230. *RBAREX*
231. *TAN*
232. *IBAIEX*
233. *TANH*
234. *ATAN*
235. *ENDFIL*
236. *CPC*

OVERLAY 03.00

1. CTLS
2. CTLS1
3. CTLS2
4. DEF
5. LINES
6. DISPLAY
7. SEARCH
8. STOP1
9. CARDS
10. DALCAL
11. KCALC
12. INVERT
13. DECIDE
14. OFFSW
15. DSERCH
16. TIUU
17. UNBLOCK
18. SUMOLA
19. PLOT
20. MATINV
21. DECID3
22. DECID2
23. PACK
24. WNORM
25. UPDK
26. *INPUTB*
27. *ENDFIL*
28. *LOCF*
29. *OUTPTS*
30. *RBAREX*
31. *INPUTS*
32. *KRAKER*

OVERLAY 04.00

1. REV
2. CTVSR
3. ADJEQ
4. DEF
5. LINES
6. UNPART
7. INTGRTR
8. TLUREV
9. MIMINR
10. CODES
11. VALUES
12. TLU1
13. WMA
14. DALPACK
15. FLUSH
16. IZUNPK
17. STOP1
18. *BACKSP*
19. *INPUTB*

OVERLAY 05.00

1. GRAPH
2. PAPLT
3. PRPLT
4. FRAME
5. PLTCUR
6. ABSBEAM
7. ABSVECT
8. MAP
9. LINEOPT
10. CHAROPT
11. SETBEAM
12. NUMBER
13. SYMBOL
14. VECTOR
15. *INPUTB*

OVERLAY 06.00

1. DGAMES

OVERLAY 07.00

1. CTAE
2. STOP1
3. INITOP
4. BESTAF
5. MAINOP
6. WARPS
7. CODES
8. VALUES
9. CODES2
10. VALUES2
11. SADDLE
12. PENLTY
13. SADDLE
14. PGAIN
15. ALFLIM
16. INEVAL
17. FESET
18. SAVALF
19. PATERN
20. SELECT
21. PSRCH
22. WEIGHT
23. ENDCYC
24. FNEVAL
25. OUTALF
26. OUTFUN
27. SUMARY
28. DEF
29. SECCON
30. MAGNFY
31. STDESC
32. CREEPR
33. QUADRA
34. DAVIDN
35. RPOINT
36. RANRAY
37. RAYSEC
38. BAESOP
39. OUTWGT
40. OUTSUM
41. PCYCLE
42. PFINAL
43. RANDUM
44. BOUND
45. SECTON
46. DERIV
47. WMAT
48. INISTD
49. ASVCHK
50. FESAV
51. MINALP
52. ALPERT
53. ALFNUL
54. QDTRAN
55. QUADOP
56. MAXRDP
57. DMATRX
58. SAVDER
59. RGEN
60. RANCOS
61. SHELL
62. FUNTYP
63. STDALF
64. MATMLT
65. QMXINV
66. *RBAIEX*
67. *INPUTN*
68. *OUTPTN*
69. *ENDFIL*
70. *INPUTB*

7. Program Organization

The combat simulation computer program is written in CDC FORTRAN IV except for one small display routine OFFSW and two small character manipulation routines GET and PUT. The program takes advantage of the overlay feature to minimize core requirements.

This section describes the overall organization of the program from the viewpoint of control cards, tape usage, and deck set up. The program is broken into eight overlays as follows:

1. MAIN - MAIN zeros out all of the common locations and moves the directory data to common locations.

2. MAIN1 - Does some data initialization and prepares an input tape for use in the iterative procedure.

3. EXE - Solves the equations of motion and computes the partial derivatives.

4. CTLS - To compute the step size and new control variable table for the next trajectory being computed.

5. REV - The reverse integration program computes integrals which determine the mode shape of the changes in the control variables and puts this out on tape for use in CTLS.

6. GRAPH - Dummy

7. DGAMES - Dummy

8. CTAE - Parameter optimization AESOP

a. Storage References

All variables requiring arrays have been arranged in the standard FORTRAN convention. For example, an array A_i is stored in increasing storage locations for increasing i. Matrices are stored columnwise.

b. Integers

All integers are assumed to be in a 60 bit word right justified.

c. COMMON

In order to decrease the length and time required in calling sequences, liberal use of COMMON has been made. Three types of COMMON are used: blank, numbered, and labeled. For the actual variables and their arrangement in COMMON, the user is referred to the program listing.

d. Variable Names

Becuase any variable may be referred to by FORTRAN, all integer variable names begin with the leading letters I, J, K, L, M, or N. This

does not mean that all non-integer variable names begin with letters other than I, J, K, L, M, or N. They may, in some subprograms, be declared integer or real.

8. Program Generation, Modification, and Execution

The following examples are designed to aid the user in utilizing the features of the CDC 6000 series computer for modification and execution in the combat simulation program. All examples shown are based upon an overlayed program organized as follows:

```
OVERLAY(TRAJOPT, 0, 0)
MAIN
OVERLAY(TRAJOPT, 1, 0)
MAIN1
OVERLAY(TRAJOPT, 2, 0)
EXE
OVERLAY(TRAJOPT, 3, 0)
CTLS
OVERLAY(TRAJOPT, 4, 0)
REV
OVERLAY(TRAJOPT, 5, 0)
GRAPH
OVERLAY(TRAJOPT, 6, 0)
DGAMES
OVERLAY(TRAJOPT, 7, 0)
CTAE
```

All control cards are left-justified in card column 1. The end of record is a 7, 8, 9 punched in column 1 and an end of file card is a 6, 7, 8, 9 punched in column 1. In the control card examples, an *end of record* and an *end of file* will be used in place of these cards.

a. Building the Program Overlay File

In constructing the program overlay file a CDC-developed utility program COPYLIB is used. COPYLIB is a user library simulation copy routine developed by CDC to take most of the work out of building overlay or normal load files. However, COPYLIB is not a standard CDC utility program; hence, a short description is supplied below with examples of its use.

COPYLIB is called by a control card and reads text cards.

(1) Control Card

COPYLIB(*OUTFILE, LIB1, LIB2, . . . LIBi*)

where *OUTFILE* is the name of the disk or tape file upon which the output (the program overlay file) is to be written, and *LIB1, LIB2, . . ., LIBi* ($1 \leq i \leq 6$) are the names of the user-supplied library files containing subprograms output from a CDC 6600 compiler or assembler in relocatable object form (odd parity).

(2) Text Cards

The order and content of the text cards define the output file. They are free form in columns 1 through 72, blanks ignored. The text cards are listed as follows:

ident

where *ident* is a subprogram name. The purpose of the *ident* card is to name the main program. Once the main program name is known, it and all the routines it calls or references and all they call or reference that were available in the library files are copied onto the output file specified.

Usually, only the one card naming the main program need be given except for certain cases such as Block Data routines that are necessary but not specifically called or referenced by any program. (In the case of an otherwise unnamed Block Data routine, the additional *ident* card would contain only BLKDATA). Another instance might be one in which the order of loading was important to guarantee that the longest named COMMON reference would come first. The order would be forced by the insertion of additional cards containing the names of the routines in the order required.

OVERLAY(*fn*, *I1*, *I2*)

Overlay text cards are necessary to properly define the structure of the file to be built for overlay loading. These cards cause an overlay loader directive record containing all the information on the text card to be written on the out file. The order and form of this text card must be exactly as defined in the Scope Reference Manual or the FORTRAN Reference Manual, with the exception of the starting column.

An *ident* text card containing the name of the main program in the overlay must follow each overlay text card.

Given correct *overlay* and *ident* text cards, COPYLIB will correctly build an overlay structure file; no routine needed or defined in a more fundamental overlay will be placed in a less fundamental level. If an *ident* card incorrectly attempts to call a routine that has somehow either through a previous *ident* text card or through a call by a subroutine at that level already been placed in the more fundamental level, the *ident* card is ignored and an informative diagnostic is printed. It is possible to have several 0, 0 level overlay cards in the text stream if the purpose is to build different overlay structured programs.

WEOF

This text card causes an end of file to be written on the OUTFILE after all the preceding text cards are processed. (A file mark might be

between two separate overlay structured programs being output in a single run).

(3) Interesting Details and Limitations

For perhaps 98 per cent of all the times COPYLIB is used, 50000$_8$ will be sufficient field length. COPYLIB will abort if the following internal tables overflows:

Name of Table	Size
Library Subprogram Name	768
Subprogram Entry Points	1280
Subprogram External References	3840
Current Overlay Need Stack	383
Working Storage Buffer	variable (see below)

The working storage buffer size can be determined by subtracting 40500$_8$ from the field length. It is difficult to determine what the minimum size required will be unless the lengths of the relocatable binary records present on the user library files are known. The length of the longest record determines the minimum working storage buffer size. (Note: this length is not the amount of core required to load the subprogram for execution but the number of words output by the compiler or assembler.) In other words, it is proportional to the number of binary cards that would be punched out,were the subprogram punched out, not *necessarily* related to the size of any arrays dimensioned inside the subprogram. This length can be obtained exactly, if necessary, from the information output by a "LIBLIST" of the library files, and should be rounded upward to the nearest 1000$_8$ when figuring the minimum field length necessary for COPYLIB.

COPYLIB rewinds each user library file starting with the first mentioned then transfers every routine contained in it to a random access file, rewinds the library file, and then repeats this process with the next user library file mentioned, for every file given.

If during the transfer process a subprogram is found that has a name duplicating one found previously, the latter subprogram is skipped, an informative diagnostic printed, and the process continues. This is handily put to use when one wishes to use a newer version of a routine instead of the version contained in one of the user library files, e.g., by placing the name of the newer library file to the left of the older version, the user causes the duplicate routines on the later file to be ignored.

Entry points must be unique to one subprogram. If two or more have the same entry point names, COPYLIB output may be scrambled. The responsibility for proper overlay text card sequence is entirely the user's. Incorrect sequencing, as defined in the Scope and FORTRAN Reference manuals, will not be flagged until an attempt is made to load the outfile.

The out file is rewound at the beginning and end of COPYLIB. It will be ended with one end-of-file mark unless more are forced through *WEOF* cards at the end of the text cards.

The random access file mentioned earlier is called RANSCR and *must* be a disk file. However, at the conclusion of COPYLIB it can be rewound and copied by the normal control cards (REWIND and COPYBF) if the user wishes to save a new version of the user library. This file contains all of the routines found in the library files input to COPYLIB minus any duplicate routines, overlay cards, and compiler or assembler error records.

The present version does not allow the use of INPUT (the card reader) as a library file.

EXAMPLE 1

The initial installation of COPYLIB as a permanent file.

```
RFL,60000.
UPDATE(N,C)
FIN(I=COMPILE)
LOAD(LGO)
NOGO.
CATALG(COPYLIB,COPYLIB,ID=ARCAFFDL,EX=ARC1,
       CN=ARC1,MD=ARC1,RP=999)
end of record
{COPYLIB source deck}
end of record
```

EXAMPLE 2

The initial installation of the COMBAT simulation program.

The FORTRAN source decks are arranged, sequenced, and stored alphabetically. The following deck set up is used for the initial generation of the COMBAT simulator program overlay file NEWPGM on tape ARC01.

```
REQUEST NEWPGM,HI.  (ARC01/RING)
RFL,60000.
RUN(S,,,,,,77000)
ATTACH(COPYLIB,COPYLIB)
COPYLIB(NEWPGM,LGO)
{COMBAT source decks arranged alphabetically
```

```
OVERLAY(TRAJOPT,0,0)
MAIN
OVERLAY(TRAJOPT,1,0)
MAIN1
OVERLAY(TRAJOPT,2,0)
EXE
OVERLAY(TRAJOPT,3,0)
CTLS
OVERLAY(TRAJOPT,4,0)
REV
OVERLAY(TRAJOPT,5,0)
GRAPH
OVERLAY(TRAJOPT,6,0)
DGAMES
OVERLAY(TRAJOPT,7,0)
CTAE
```

COPYLIB text cards

end of record
end of file

EXAMPLE 3

Modification of the combat simulation program

The following deck set up is used when making modifications to the combat simulation program.

```
REQUEST OLDPGM,HI.  (ARC01/NORING)
REQUEST NEWPGN,HI.  (ARC02/RING)
RFL,60000.
RUN(S,,,,,,77000)
ATTACH (COPYLIB,COPYLIB)
COPYLIB(NEWPGM, LGO, OLDPGM)
```

end of record

Modified COMBAT source decks

end of record

```
OVERLAY(TRAJOPT,0,0)
MAIN
OVERLAY(TRAJOPT,1,0)
MAIN1
OVERLAY(TRAJOPT,2,0)
EXE
OVERLAY(TRAJOPT,3,0)
CTLS
OVERLAY(TRAJOPT,4,0)
REV
OVERLAY(TRAJOPT,5,0)
GRAPH
OVERLAY(TRAJOPT,6,0)
DGAMES
OVERLAY(TRAJOPT,7,0)
CTAE
```

COPYLIB text cards

end of record
end of file

EXAMPLE 4

The following deck set up is used to modify and execute the combat simulation program.

```
REQUEST OLDPGM,HI.  (ARC01/NORING)
REQUEST NEWPGM,HI.  (ARC02/RING)
REQUEST ABSPGM,HI.  (ARC03/RING)
RFL,60000.
RUN(S,,,,,,77000)
ATTACH(COPYLIB,COPYLIB)
COPYLIB(NEWPGM,LGO,OLDPGM)
REWIN (NEWPGM)
RFL,317000.
SET(0)
MODE(1)
LOAD(NEWPGM)
NOGO.
RFL,20000.
REWIND(TRAJOPT)
REWIND(ABSPGM)
COPYBF(TRAJOPT,ABSPGM,1)
REWIND(TRAJOPT)
RFL,317000.
TRAJOPT.
end of record
```

{ Modified COMBAT source decks

```
end of record

OVERLAY(TRAJOPT,0,0)
MAIN
OVERLAY(TRAJOPT,1,0)
MAIN1
OVERLAY(TRAJOPT,2,0)
EXE
OVERLAY(TRAJOPT,3,0)
CTLS
OVERLAY(TRAJOPT,4,0)
REV
OVERLAY(TRAJOPT,5,0)
GRAPH
OVERLAY(TRAJOPT,6,0)
DGAMES
OVERLAY(TRAJOPT,7,0)
CTAE
end of record
```

COPYLIB Text Cards

{ *data deck for COMBAT program*

```
end of record
end of file
```

EXAMPLE 5

The following deck set up is used to execute the combat simulation program from a tape containing the absolute program element.

```
REQUEST ABSPGM, HI.  (ARC03/NORING)
REWIND(ABSPGM)
REWIND(TRAJOPT)
RFL,20000.
COPYBF(ABSPGM,TRAJOPT,1)
RFL,317000.
MODE(1)
TRAJOPT.
end of record
                    } data deck for combat
                    } simulation program
end of record
end of file
```

(4) Program Call Card

In the two previous examples the combat simulation program was executed by use of the program call card

```
TRAJOPT.
```

where TRAJOPT is the name of a disk file that contains the absolute program element. The program call card may also be used to override file name parameters which appear on the program card of the source deck. The file parameters appear as follows:

```
 PROGRAM MAIN (INPUT=1001, OUTPUT=1001, TAPE5=INPUT, TAPE6=OUTPUT,
$             TAPE10=1001, TAPE11=1001, TAPE12=1001, TAPE13=1001,
$             TAPE14=1001, TAPE15=1001, TAPE16=1001, TAPE17=1001,
$             TAPE18=1001, TAPE12=1001,
$             TAPE20=1001, TAPE21=1001, TAPE22=1001, TAPE23=1001,
$             TAPE25=1001, TAPE26=1001, TAPE27=1001, FILMPL=1001)
```

The restart cards for the combat program are written on TAPE15. The following call card may be used with any 6000 machine which has a card punch and recognizes the PUNCH file to automatically punch the restart cards at the end of the job.

```
TRAJOPT(,,,,,,,,, PUNCH)
```

9. Data Format

Card Format - The program input routine (READA) expects the following format.

Card Columns	1-6	7	8-10	11	12-66	67-72	73-80
Field	I	II	III	IV	V	VI	VII

Card Field I — Contains the symbolic name of the variable into which data contained in Field V begins loading.

Example:

Card Column	1	12
	GAM7D	-1.23
	SIG7D	90.

Card Field II — Not used.

Card Field III — Contains the words DEC, OCT, BCD, TRA, INT, PAR, or is blank depending on the type of data to be loaded. The word OCT indicates that the data is to be interpreted as octal numbers. The word BCD specifies that N binary coded decimal words (N punched in column 12) beginning in column 13 are to be loaded. The word TRA denotes to the input routine that all data has been input and to return control to the calling program. The word PAR indicates that the input quantity is to be treated as a free parameter in a multivariable optimization study. The word DEC and blank are equivalent and specifies that data loaded is decimal data.

OCT Example

Card Column	1	8	12
	NPOINT	OCT	17

BCD Example

Card Column	1	8	12
	REM	BCD	2

The 2 in column 12 specifies two words where each word is considered to be six characters including blanks. The largest number of six character words that can be loaded from one card is nine. The analysts should be very careful to see that the BCD information does not get punched into Field VI. This will cause an input error.

DEC Example

Card Column	1	8	12
	ATABO1	DEC	2,-100.,1.,100.,1.

Note that the first character in column 12 is an integer and the input routine will load only one integer per DEC card and that has to be the first number punched in Field V.

```
ATAB01   DEC    2.,-100.,1.,100.,1.
```

If the above card is punched, the two will now be loaded into the machine as a binary floating point number. Likewise, the other numbers will be loaded in the same manner with the decimal point assumed right justified.

If anything other than OCT, BCD, INT, TRA, or blank appears in Field II, then the word DEC is assumed.

INT Example

Card Column	1	8	12
	JPSCUT	INT	1
	JPSCUT		1
	JPSCUT	INT	1, 1, 1, 1

When the word INT is used, it is assumed that all numbers on the card will be loaded as integers. If only one integer is punched per card,the INT may be punched or omitted.

PAR Example

Card Column	1	8	12
	GAM7D	PAR	2

The PAR in column 8 indicates that the input quantity GAM7D is a free parameter. The 2 in column 12 indicates that GAM7D is the second free parameter.

Card Field IV - Not used.

Card Field V - The actual input data to the program is punched in the Field V. DEC, INT, and OCT must always be left adjusted, that is, it must start in column 12 on the input card. All numbers are separated by a "comma,"and the field terminates with the first blank. BCD information begins in column 13, and the maximum number of six character words per card is nine. Note that since Field V ends with the first blank, the user may punch any comments in the remainder of the field.

Card Field VI - This field specifies the initial subscript of the data in Field V. If this field is blank, an initial subscript of 1 is implied. The subscript may appear anywhere within the field.

Example

Card Column	1	12	67
	ATAB01	4,0,20.,10.,18.	1 or blank
	ATAB01	20.,17.,30.,15.	6

In the example above the integer 4 is loaded into the first cell of the array ATAB01. On the second card 20. is loaded into the sixth cell of the array. The one and six punched in Field VI indicate the subscript for the array ATAB01.

Card Field VII - Not used as far as the input routine is concerned. This may be used as a sequence number for the card.

10. Table Format

The various types of tables used by the program may be classed as follows:

Two dimensional table.

Example	TTAB01	T = f(t)
Card Column	1	12
	TTAB01	N, t_1, T_1, t_2, T_2, t_s, T_s, . . ., t_n, T_n

N equals fixed point number equal to two times the number of independent variables. For a 20 point table N would equal 40. The total number of machine cells required for this table is 41.

t_i = independent variable values

T_i = corresponding dependent values

N-dimensional table.

Example	ATAB80	C = f(x,y)
Card Column	1	12
	IA80X	NX
	IA80Y	NY
	ATAB80	X_1, X_2, X_3, . . ., X_{nx}
	ATAB80	Y_1, Y_2, Y_3, . . ., Y_{ny}
	ATAB80	$C_{x=1,y=1}$, $C_{x=2,y=1}$, . . ., $C_{x=nx,y=1}$
	ATAB80	$C_{x=1,y=2}$, $C_{x=2,y=2}$, . . ., $C_{x=nx,y=2}$
	.	. . .
	.	. . .
	.	. . .
	ATAB80	$C_{x=1,y=ny}$, $C_{x=2,y=ny}$, . . ., $C_{x=nx,y=ny}$

NX and NY are fixed point numbers of independent variables. $C_{x=1,y=1}$ . . . $C_{x=nx,y=ny}$ equal values of independent variables. The table subscripts would apply to the N-dimensional table as well as the two dimensional. The total number of machine cells required for an N-dimensional table equal XN(NY) + NX + NY.

Examples

c = f(x,y)	NX points for x = 2
	NY points for y = 2

Machine cells requires 2 x 2 + 2 + 2 = 8 cells

c = f(X, Y, Z) NX = points for X = 20
NY = points for Y = 10
NZ = points for Z = 15

Machine cells required = 20 x 10 x 15 + 20 + 10 + 15 = 3045 cells.

All tables must have at least two points per table. A check for each individual table should be made to see if an indicator is necessary for the program to read the table.

11. Brief Write-up on System Routines

This section is a brief write-up of some of the system subroutines used by the CDC 6000 software. Since the system software is in a state of constant change, this information may be of little or no use.

a. INPUTB

Only one logical record is read each time INPUTB is called. If the list is longer then the logical record, the excess words in the list are ignored by the routine.

An attempt to read past an end of file will cause a program short. The end of file condition can be cleared by testing for end file after the file mark has been read and *prior* to another attempt to read.

INPUTB calls routines GETBA (to locate Buffer Argument address), UMERR (to output Unassigned Medium diagnostic), MARKFI (to end file OUTPUT and position file INPUT), and XRCL (to go on Recall while an I/O transmission is in progress and computation can not continue).

b. INPUTC

INPUTC handles the I/O transmission for the input to the computer. It calls routine KRAKER to perform conversion.

An attempt to read past an end of file will cause the program to be aborted.

The end of file condition can be cleared by testing for end file *after* the file mark has been read and *prior* to another attempt to read. The end file condition for the file INPUT is set by either an end of file mark or a short record (end of logical record).

INPUTC also calls routines GETBA (to locate Buffer Argument address), UMERR (to output Unassigned Medium diagnostic), MARKFI (to end file output and position file INPUT on program abort), and XRCL (to go on Recall while on I/O transmission is in progress and computation can not continue).

c. INPUTS

INPUTS does the core to core transmissions. It calls routine KRAKER to perform conversion.

The parameter specifying the record length may be an arbitrary number of BCD characters less than 150. The record starts with the leftmost character of the location specified by Format and continues ten BCD characters per computer memory word for the BCD characters or until a zero character is encountered. If the record ends in the middle of a word, the remaining characters are ignored. Each record begins with a new computer word. The number of records processed by each call to INPUTS depends on the Format and the length of the list. If the number is greater than 150 characters, the routine aborts the program and gives a diagnostic.

INPUTS also calls routines CONADD (to convert the calling address) and MARKFI (to end file OUTPUT and position file INPUT).

d. KODER

KODER is a data conversion routine. The necessary conversion is specified by a Format. Any transmission of data is handled by the routine making the call to KODER.

KODER calls routines CONADD (to convert the calling address) and MARKFI (to end file OUTPUT and position file INPUT on the program abort).

e. KRAKER

KRAKER is a data conversion routine. The necessary conversion is specified by a Format. KRAKER is a DECODE conversion routine,and KODER is an ENCODE conversion routine.

KRAKER calls routines CONADD (to convert the calling address) and MARKFI (to end file OUTPUT and position file INPUT on program abort).

f. OUTPTB

OUTPTB does the I/O for binary output. One logical record is written each time OUTPTB is called; to decrease the number of I/O transmissions, data should be blocked in large arrays before outputting to some I/O device.

OUTPTB calls routines GETBA (to locate Buffer Arguments address), UMERR (to output error message), and XRCL (to go on recall while an I/O transmission is in progress).

g. OUTPTC

OUTPTC does the I/O transmission for the output file. The information is stacked into a buffer, and I/O is done when the buffer is filled. It calls KODER for data conversion. For the output file only, a line count is kept, and, if this count is exceeded, it causes an abort.

OUTPTC also calls routines GETBA (to locate Buffer Argument address), UMERR (to output error diagnostic), MARKFI (to end file OUTPUT and position

file INPUT on program abort, and XRCL (to go on recall while an I/O transmission is in progress).

h. OUTPTS

OUTPTS performs the reverse of INPUTS. ENCODE routine makes use of OUTPTS where DECODE routine calls INPUTS.

OUTPTS also calls routines CONADD (to convert the calling address) and MARKFI (to end file OUTPUT and position file INPUT).

i. STOP

STOP flushes the buffer and places a file mark on the files named OUTPUT and PUNCH if they were declared on the Program Header Card.

j. END

The purpose of END is to position the file named INPUT to the beginning of the next logical record if it has not been so positioned by the program and if it has been declared on the Program Header Card.

k. EXIT

EXIT enters a dayfile message with the name of the routine and, for END and STOP, follows the routine name with an actual number if one appeard on the source statement.

No other files are terminated or positioned by these routines. It is the programmer's responsibility to insure that other buffers are flushed and that their files are properly terminated or positioned at the end of a program.

12. ENCODE/DECODE

ENCODE and DECODE are system routines and are comparable to the BCD write/read statements with the essential difference that no peripheral equipment is used in the data transfer. Information is transferred under Format specifications from one area of storage to another.

Entry is made to the routines by the following statements:

```
ENCODE(C,N,V)LIST
DECODE(C,N,V)LIST
```

where N is a Format statement number, a variable identifier, or a formal parameter representing the associated Format list. LIST is the input/output list. V is a variable identifier or an array identifier that supplies the starting location of the records. The identifier may be subscripted.

C is an unsigned integer or an integer variable, simple or subscripted, specifying the length of a record. C may be an arbitrary number of BCD characters. The first record starts with the leftmost character of the location specified by V and contains BCD characters.

a. ENCODE

ENCODE converts the information in the list according to Format list N and stores it in locations starting at V,C BCD characters per record. If the Format list attempts to convert more than C characters per record, a diagnostic occurs. If the number of characters converted by the Format list is less than C, the remainder of the record is filled with blanks.

When C is not a multiple of four, the last record does not fill a computer word; the remainder of the word is blank-filled.

b. DECODE

DECODE converts and edits information from records consisting of C consecutive BCD characters starting at Address V according to Format list N and stores it in the I/O list. When the Format list specifies more than C characters per record, a diagnostic is provided. If DECODE attempts to process a character illegal under a given conversion specification, a diagnostic occurs. When fewer than C characters are specified, the remainder of the word is ignored.

Since these routines are system routines, no flow charts will be furnished, and all diagnostics will come from the FORTRAN system.

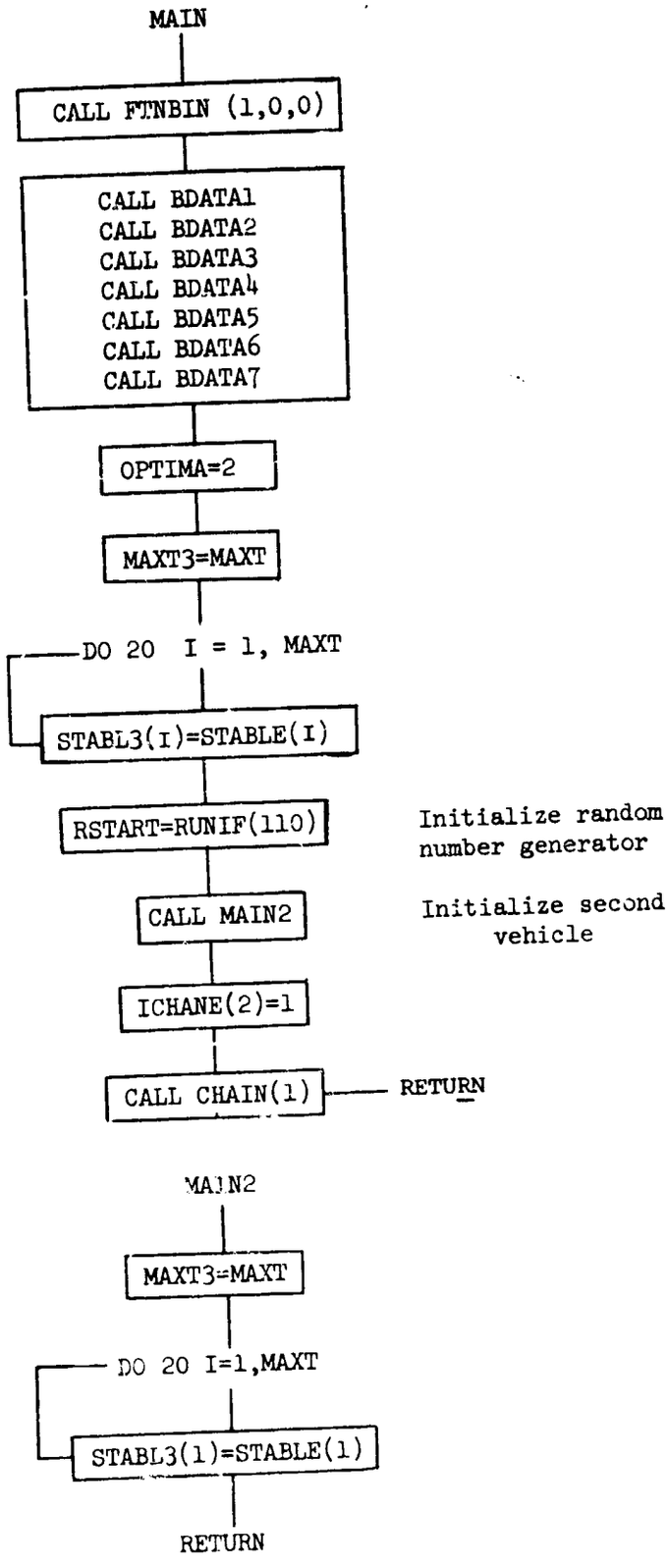
SECTION IV

MAIN PROGRAM AND SUBROUTINE MAIN2

1. MAIN and MAIN2

The MAIN program is a requirement of the CDC system. When called by the system, all of Blank and Numbered COMMON is zeroed. The MAIN program may consist of as many subroutines as necessary. The only functions that MAIN serves to this program besides meeting the system requirement is to move the vehicle table directory from Labeled COMMON to Numbered COMMON to initialize program block data, to initialize the random number generator, and to call MAIN2.

Subroutine MAIN2 moves the vehicle 2 table directory from Labelled COMMON to Numbered COMMON.


MAIN
CALL FTNBIN (1,0,0)
CALL BDATA1
CALL BDATA2
CALL BDATA3
CALL BDATA4
CALL BDATA5
CALL BDATA6
CALL BDATA7
OPTIMA=2
MAXT3=MAXT
DO 20 I = 1, MAXT
STABL3(I)=STABLE(I)
RSTART=RUNIF(110)
Initialize random number generator
CALL MAIN2
Initialize second vehicle
ICHANE(2)=1
CALL CHAIN(1)
RETURN
MAIN2
MAXT3=MAXT
DO 20 I=1,MAXT
STABL3(1)=STABLE(1)
RETURN

2. BDATA1 - Block Data Subroutine

This block data subroutine establishes all basic directory available maneuvers.

3. BDATA2 - Block Data Subroutine

This block data subroutine establishes all data table names.

4. BDATA3 - Block Data Subroutine

This block data subroutine establishes control variable table names CTABLE and DALALF, for vehicle 1.

5. BDATA4 - Block Data Subroutine

This block data subroutine is a dummy routine.

6. BDATA5 - Block Data Subroutine

This block data subroutine establishes control variable table names CTABLE and DALALF for vehicle 2.

7. BDATA6 - Block Data Subroutine

This block data subroutine establishes all additional directory variable names.

8. BDATA7 - Block Data Subroutine

This block data subroutine sets all AESOP parameter optimization variables.

9. SPRANG - Random Number Generator

Random number generator for AESOP program. See Volume IV.

10. CHAIN - Overlay Control Program

Purpose:

Controls program overlay structure.

Method:

CHAIN controls the following seven program overlays:

OVERLAY 1, MAIN-MAIN PROGRAM FOR TRAJECTORY CALCULATION

OVERLAY 2, EXE-EXECUTIVE PROGRAM FOR TRAJECTORY CALCULATION

OVERLAY 3, CTLS-VARIATIONAL OPTIMIZATION CONTROL PROGRAM

OVERLAY 4, REV-TRAJECTORY REVERSE INTEGRATION

OVERLAY 5, GRAPH-GRAPHICAL ROUTINES

OVERLAY 6, DGAMES-DUMMY OVERLAY

OVERLAY 7, CTAE-PARAMETER OPTIMIZATION CONTROL

Usage:

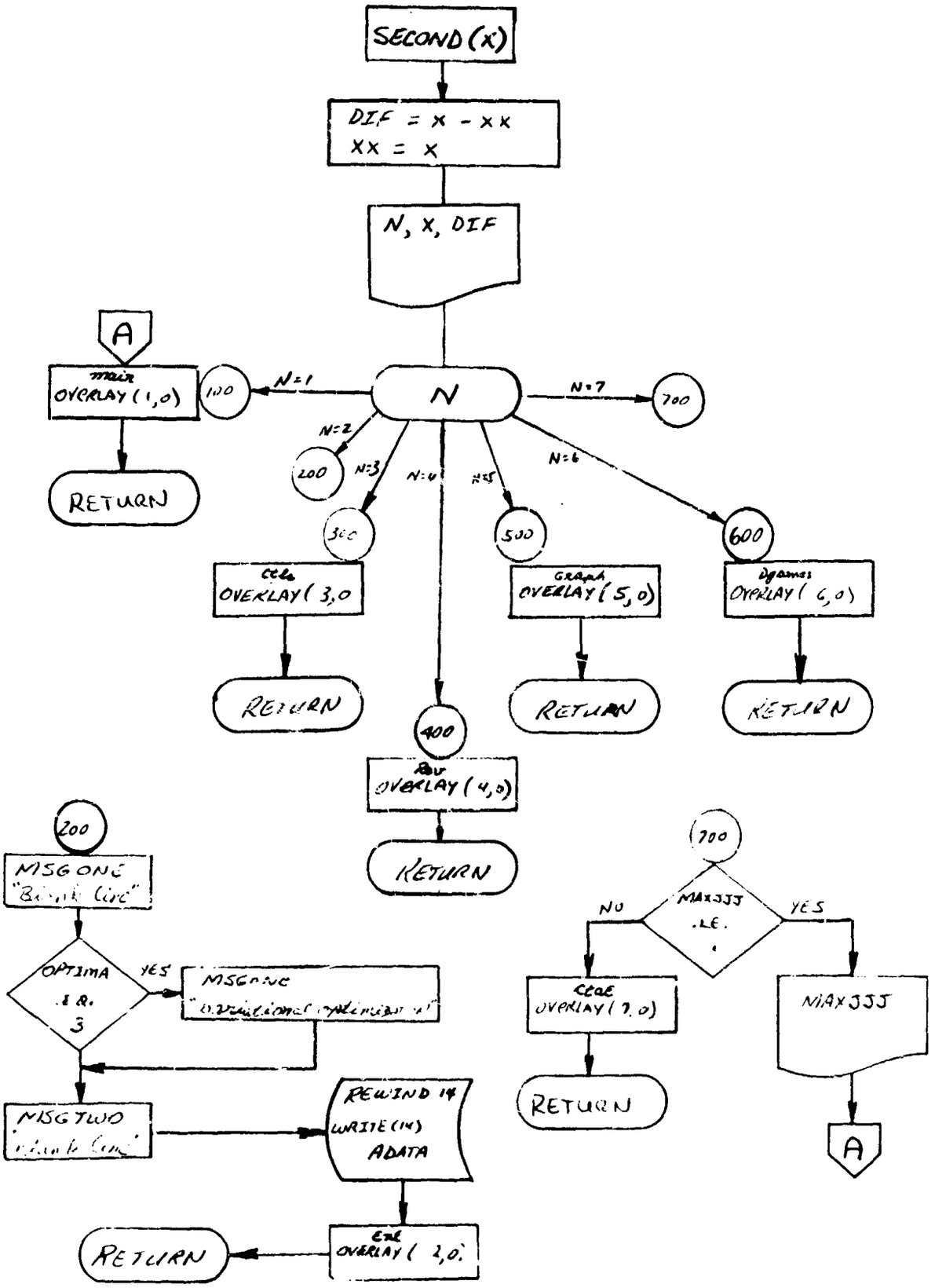
Entry is made to the routine with the following statement:

```
CALL CHAIN(N)
```

where

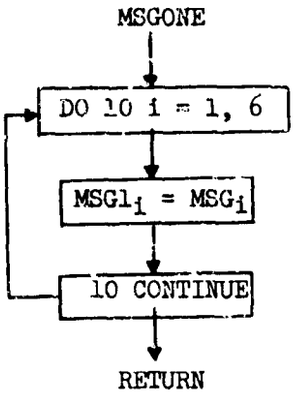
N = Overlay number to be loaded

CHAIN (N)


SECOND (X)
DIF = X - XX
XX = X
N, X, DIF
N
N=1
N=2
N=3
N=4
N=5
N=6
N=7
100
200
300
400
500
600
700
A
OVERLAY (1,0)
RETURN
OVERLAY (3,0
OVERLAY (5,0)
OVERLAY (6,0)
OVERLAY (4,0)
MSGONE
OPTIMA
3
YES
MSGTWO
REWIND 14
WRITE (14)
ADATA
OVERLAY (2,0)
MAXJJJ
.LE.
NO
YES
OVERLAY (7,0)
MAXJJJ

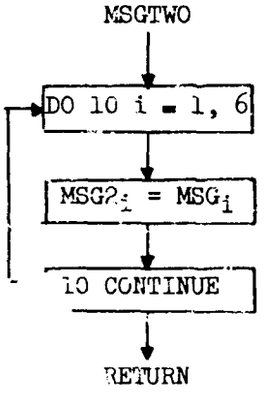
11. MSGONE - Combat Message Subroutine, Vehicle 1

Preserves the current combat message for vehicle 1. Message is subsequently printed by EXE



12. MSGTWO - Combat Message Subroutine, Vehicle 2

Preserves the current combat message for vehicle 2. Message is subsequently printed by vehicle 2.



SECTION V

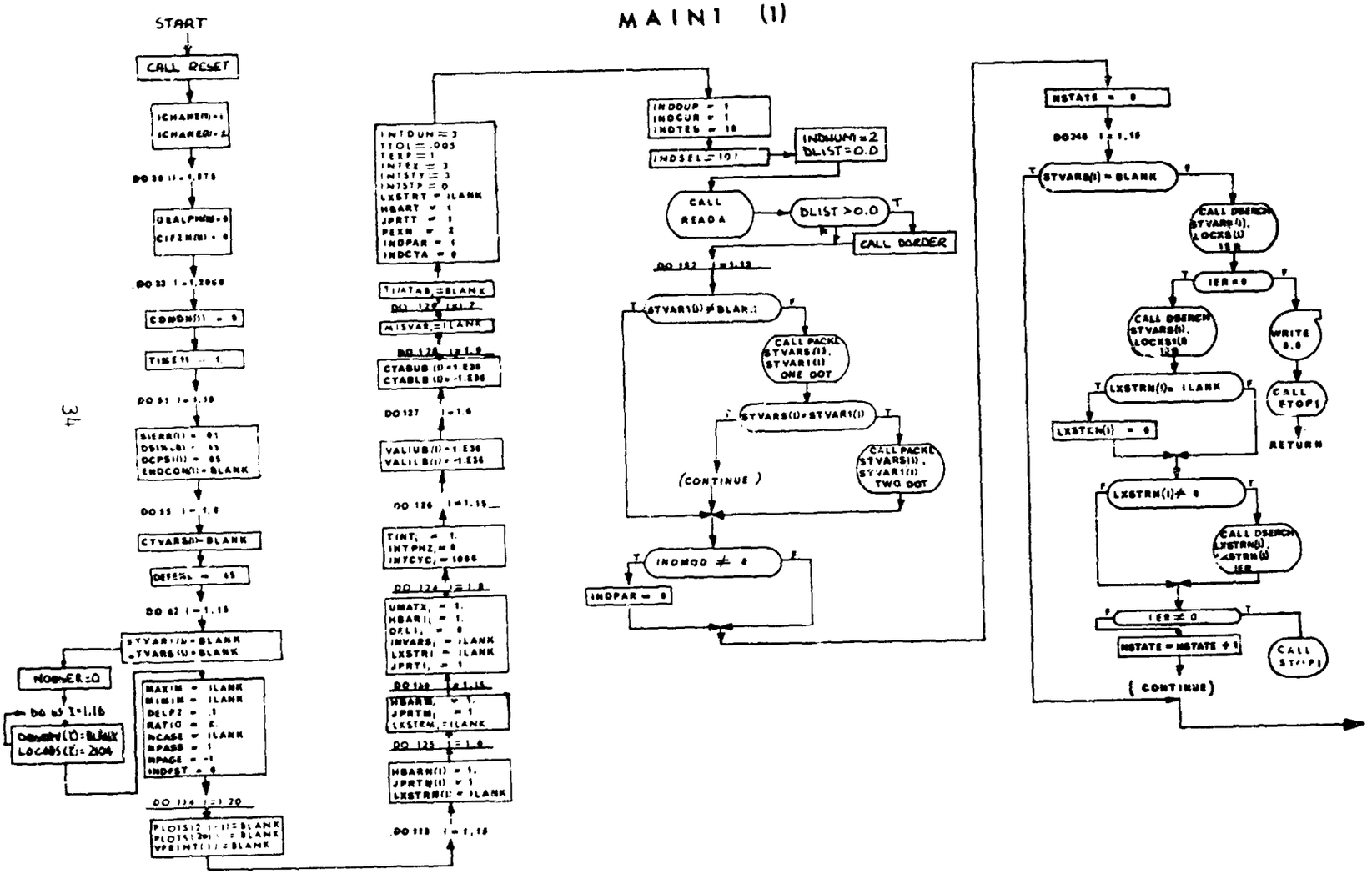
PROGRAM MAIN1 AND SUBROUTINE MAIN12

This program is executed once per case. The nominal values are set, and subscripts to BCD word input are searched for. DATA2 is read in,and a tape is prepared in binary to be read again in program EXE for each trajectory. DATA1 is read in this chain and never read again for that case. The MAIN1 program determines whether or not the two vehicle option is requested (INDNOM=2) and whether or not a variational optimization formulation option is to be employed (OPTIMA=2).

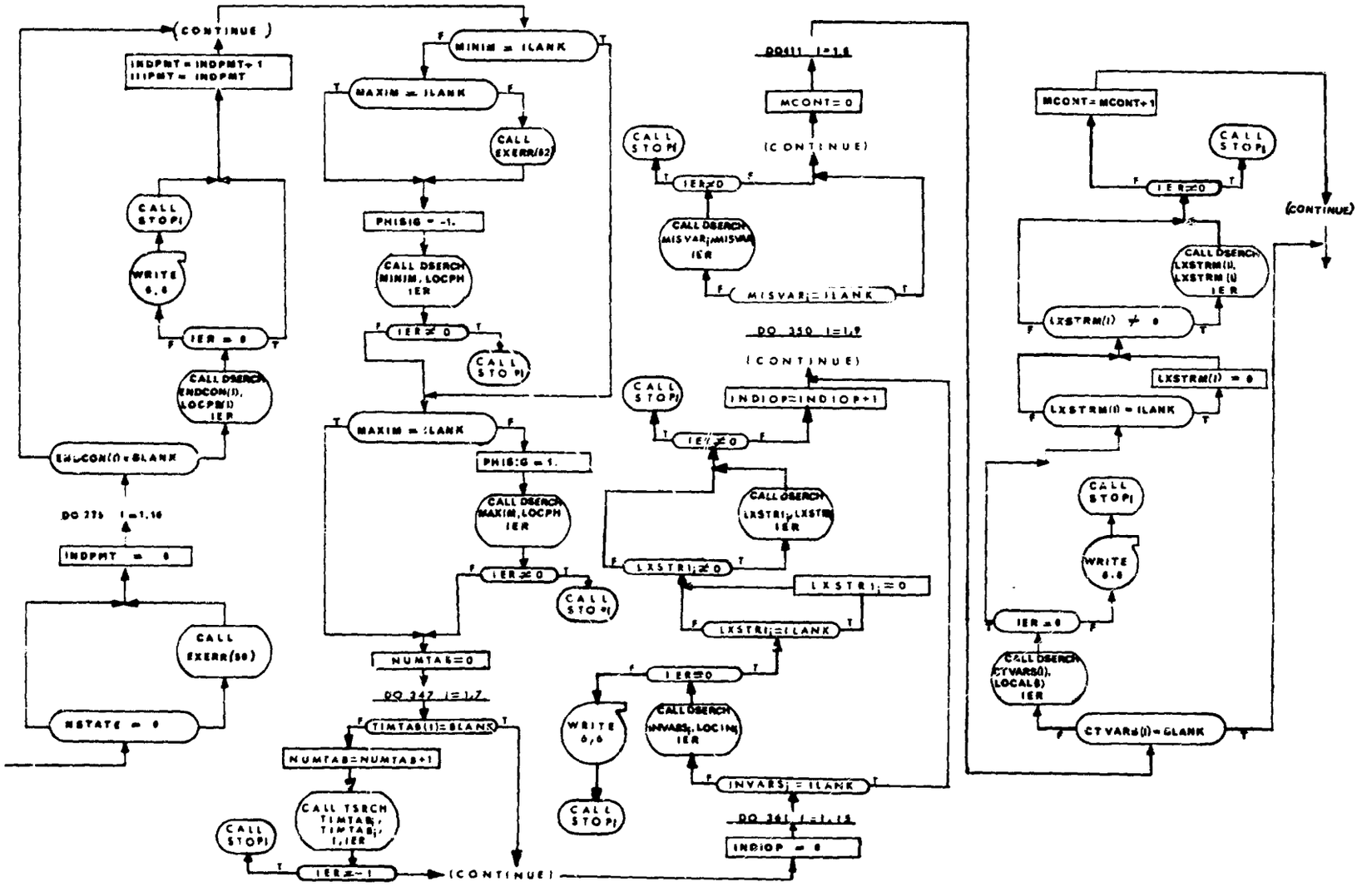
Subroutine MAIN12 performs the MAIN1 function for vehicle 2. However, since MAIN12 is only called when a second vehicle exists, the test for a second vehicle is omitted. Again, since a variational option is requested, it must be exercised through the *first* vehicle; the variational test is omitted from MAIN12.

A flow chart for MAIN1 follows; MAIN12 is identical to MAIN1 with the exceptions noted above.

MAIN1 (1)

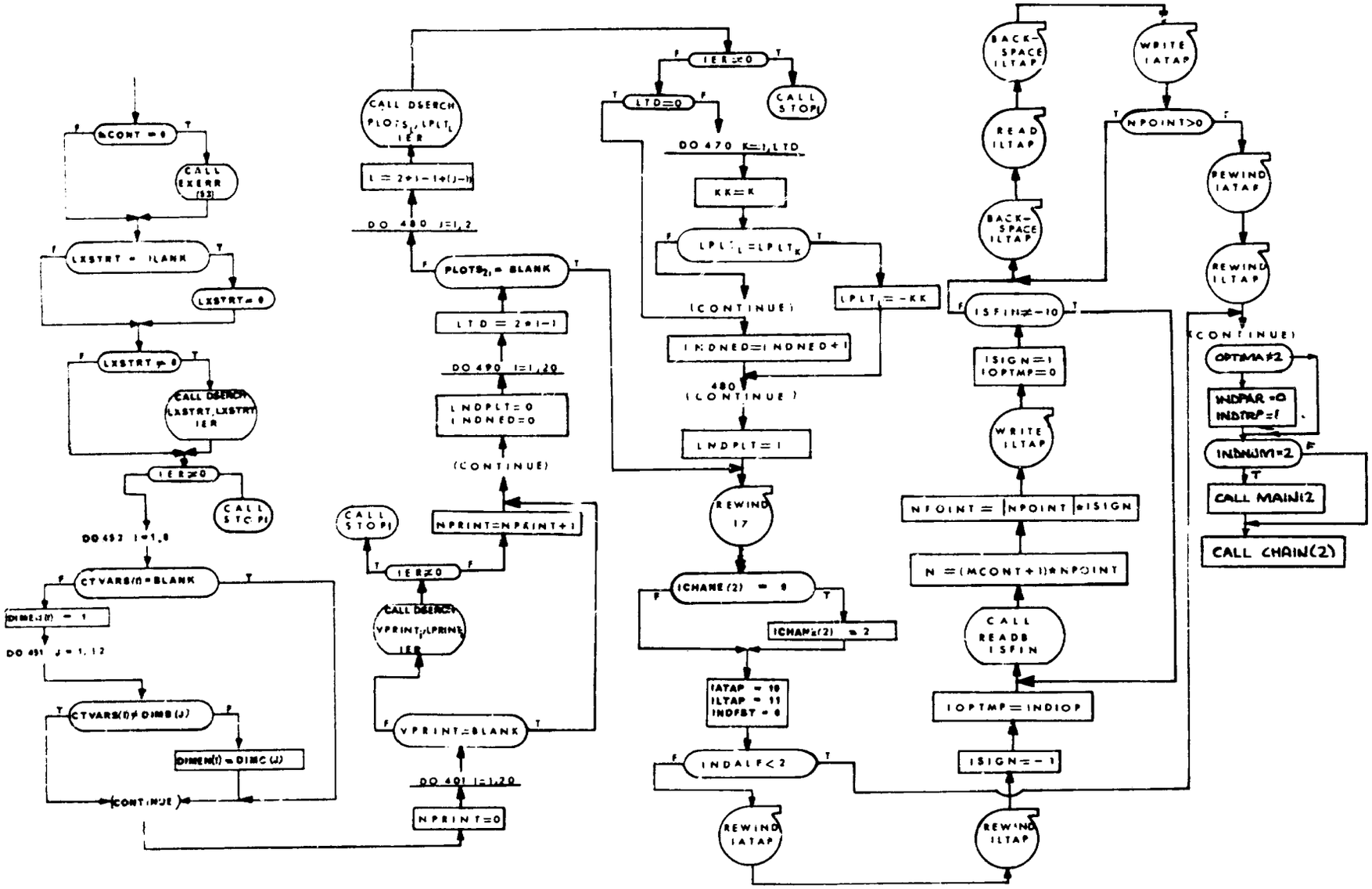

START
CALL RESET
CALL READA
DLIST > 0.0
CALL BORDER
CALL PACKL STVARS(I), STVAR1(I) ONE DOT
CALL PACKL STVARS(I), STVAR1(I) TWO DOT
(CONTINUE)
INDMOD ≠ 0
INDPAR = 0
NSTATE = 0
STVARS(I) = BLANK
IER = 0
WRITE 6,6
CALL STOP1
RETURN
LXSTRN(I) = ILANK
LXSTRN(I) = 0
LXSTRN(I) ≠ 0
IER ≠ 0
NSTATE = NSTATE + 1
(CONTINUE)
CALL STOP1
INDDUP = 1
INDCUR = 1
INDTES = 10
INDSEL = 101
INDNUM = 2
DLIST = 0.0
INTDUN = 3
TTOL = .005
TEXP = 1
INTEX = 3
INTSTY = 3
INTSTP = 0
LXSTRT = ILANK
HBART = 1
JPRTT = 1
PEXN = 2
INDPAR = 1
INDCTA = 0
TIMATAB = BLANK
MISVAR = ILANK
CTABUB (I) = 1.E36
CTABLB (I) = -1.E36
VALIUB (I) = 1.E36
VALILB (I) = -1.E36
TINT = 1.
INTPHZ = 0
INTCYC = 1000
UMATX = 1.
HBARI = 1.
DELI = 0
INVARS = ILANK
LXSTRI = ILANK
JPRTI = 1
HBARN = 1.
JPRTN = 1
LXSTRN = ILANK
HBARN(I) = 1.
JPRTN(I) = 1
LXSTRN(I) = ILANK
MAXIM = ILANK
MIMIM = ILANK
DELP2 = .1
RATIO = 2.
NCASE = ILANK
NPASS = 1
NPAGE = -1
INDFST = 0
NOBSER = 0
LOCABS(I) = 2104

MAIN1 (2)



(CONTINUE)
INDPMT = INDPMT+1
IIIPMT = INDPMT
CALL STOPI
WRITE 6,6
IER = 0
CALL DSERCH ENDCON(I), LOCPH(I) IER
ENDCON(I) = BLANK
DO 275 I=1,16
INDPMT = 0
CALL EXERR(50)
NSTATE = 0
MINIM = ILANK
MAXIM = ILANK
CALL EXERR(52)
PHISIG = -1.
CALL DSERCH MINIM, LOCPH IER
IER ≠ 0
CALL STOPI
MAXIM = ILANK
PHISIG = 1.
CALL DSERCH MAXIM, LOCPH IER
IER ≠ 0
CALL STOPI
NUMTAB=0
DO 247 I=1,7
TIMTAB(I)=BLANK
NUMTAB=NUMTAB+1
CALL TSRCH TIMTAB, TIMTAB, I, IER
CALL STOPI
IER = -1
(CONTINUE)
DO411 I=1,6
MCONT=0
(CONTINUE)
CALL STOPI
IER≠0
CALL DSERCH MISVAR, MISVAR IER
MISVAR = ILANK
DO 350 I=1,9
(CONTINUE)
CALL STOPI
INDIOP=INDIOP+1
IER ≠ 0
CALL DSERCH LXSTRI, LXSTRI IER
LXSTRI ≠ 0
LXSTRI = 0
LXSTRI = ILANK
IER=0
WRITE 6,6
CALL STOPI
CALL DSERCH INVARS, LOCIN IER
INVARS = ILANK
DO 361 I=1,15
INDIOP = 0
MCONT=MCONT+1
CALL STOPI
IER≠0
(CONTINUE)
CALL DSERCH LXSTRM(I), LXSTRM(I) IER
LXSTRM(I) ≠ 0
LXSTRM(I) = 0
LXSTRM(I) = ILANK
CALL STOPI
WRITE 6,6
IER = 0
CALL DSERCH CTVARS(I), LOCALS IER
CTVARS(I) = BLANK

MAIN1 (3)



CALL EXERR (53)
LXSTRT = BLANK
LXSTRT = 0
LXSTRT ≠ 0
CALL DSERCH LXSTRT, LXSTRT IER
IER≠0
CALL STOP1
DO 452 I=1,8
CTVARS(I)=BLANK
DIMEN(I) = 1
DO 451 J = 1, 12
CTVARS(I)≠DIMB(J)
DIMEN(I) = DIMC(J)
(CONTINUE)
CALL DSERCH PLOTS1, LPLT1 IER
L = 2*I-1+(J-1)
DO 480 J=1,2
PLOTS2, = BLANK
LTD = 2*I-1
DO 490 I=1,20
LNDPLT=0
INDNED=0
(CONTINUE)
NPRINT=NPRINT+1
CALL STOP1
IER≠0
CALL DSERCH VPRINT1, LPRINT1 IER
VPRINT=BLANK
DO 401 I=1,20
NPRINT=0
IER≠0
CALL STOP1
LTD=0
DO 470 K=1,LTD
KK=K
LPLT1=LPLTK
LPLT1=-KK
(CONTINUE)
INDNED=INDNED+1
480 (CONTINUE)
LNDPLT=1
REWIND 17
ICHANE(2) = 0
ICHANE(2) = 2
IATAP = 10
ILTAP = 11
INDFST = 0
INDALF<2
REWIND IATAP
REWIND ILTAP
ISIGN=-1
IOPTMP=INDIOP
CALL READB ISFIN
N =(MCONT+1)*NPOINT
NPOINT = |NPOINT|*ISIGN
WRITE ILTAP
ISIGN=1
IOPTMP=0
ISFIN≠-10
BACK-SPACE ILTAP
READ ILTAP
BACK-SPACE ILTAP
WRITE IATAP
NPOINT>0
REWIND IATAP
REWIND ILTAP
(CONTINUE)
OPTMA≠2
INDPAR=0
INDTRP=1
INDNUM=2
CALL MAIN12
CALL CHAIN(2)

1. READA and READA2- Input Routines for Vehicle 1 and Vehicle 2 Data

Purpose

To provide a general method of reading a variable field data card and assigning variable length table. Data may be read into symbolic locations in memory.

Method

Decimal, Octal, and Integer numbers are converted to binary integers. BCD information is stored in six character words.

Card Formats

The variable name punched in columns 1-6 is the location into which the first data word will be loaded. The variable field information is for relocation. A fixed point integer punched anywhere in the field (67-72) will be treated as a subscript to the variable name punched in column 1-6. Negative integers punched in columns 67-72 will be in violation of the subroutine. The first blank character found in the card to the right of column 12 terminates loading from the card. One exception to this is BCD data. Nine 6 character words may be loaded including blanks.

The general character of the data to be loaded is determined by a three letter pseudo-operation punched in columns 8-10. The pseudo-operations are: DEC or blank, OCT, INT, BCD, PAR, and TRA. The pseudo-operation TRA is a method of exit from the subroutine.

Decimal Data

Decimal data beginning in column 12 and ending in column 66 is converted to binary and loaded into the symbolic location punched in column 1-6 subscripted by the integer punched in column 67-72. Signs are indicated by + and - preceding the number. All unsigned numbers are treated as positive. If either the characters E or . or both appear in the decimal data word, the word is converted to a floating binary number. The decimal exponent used in the conversion is the number which follows immediately after the character E. This number may have a + or - sign preceding it. If the character E does not appear the exponent is assumed to be zero. If a decimal point does not appear it is assumed to be at the right of the number. Unless it is the only word or the first word on a card then it is assumed to be an integer.

All the examples below are equivalent.

1. 12.345E03
2. 12.345E+03
3. 12.345E3
4. 12345E00
5. 12345.
6. 1.2345E4
7. 1234500E-02
8. +1234500E-2

Note that in the examples above all decimal words have decimal points. If the first word on a card, or if it happens to be the only word on a card and it does not contain a decimal point, the word will be converted to binary integer.

Octal Data - OCT

The Octal data is loaded the same as decimal data but must have OCT punched in column 8, 9, and 10. All data is converted to binary with binary point assumed at the right end of each word.

Hollerith Data - BCD

Hollerith information is loaded from column 13 through 66 and assigned consecutive locations for every 6 characters. A maximum of nine 6 character words may be punched on any one card and the number of words must be punched in column 12. A subscript may also be punched in column 67-72.

Transfer - TRA

The purpose of the TRA card is to transfer control from the subroutine back to the main program. TRA must be punched in column 8, 9, and 10. The subscript field is not used. A REWIND may be punched beginning in column 12. Only the R is checked and the only use is for the rewind of a data tape.

Integer - INT

Integer data begins in column 12 and ends in column 66. INT is punched in column 8, 9, and 10. It may be relocated with respect to the BCD name by punching a subscript integer in column 67 through 72. If only one data word is punched per card, column 8, 9, and 10 may be left blank.

Parameter - PAR

Integer data begins in column 12. PAR is punched in columns 8,9,and 10. It may be relocated with respect to the BCD name by punching a subscript integer in columns 67 through 72.

Error Messages

A message is written on the output tape describing the type of error encountered. If an error is encountered, execution of the case is deleted and the subroutine only searches for other possible errors in the data.

The following error messages are possible.

1. Symbol not in directory.

2. Column 12 is blank.

If a bad pseudo-operation is punched in column 8, 9, and 10 the subroutine will treat it as decimal data.

All checking for redundancies, end of tape, format errors, etc., is handled by FORTRAN system input/output routines.

Usage

No initialization is required, the entry is established by a:

CALL READA or CALL READA2

Subroutines called and used by READA and READA2 other than normal FORTRAN system routines.

DIPLAC	DSERCH(2)	PACBCD
DEF	PACKR	SVI(2)
TABRE(2)	READ31	
LINES	BIBLOCK	

Data Preparation

The first card expected by READA or READA2 is a

STCASE TAB

with the S beginning in column 1 and TAB punched in 8, 9, and 10. Following this card is a set of cards which define the table sizes necessary for that case.

Example: TTABO1 10

TTABO2 20

On the above example TTABO1 is punched beginning in column 1. The numbers 10 and 20 are punched in column 12 and indicate the number of machine cells necessary for that table. Any number of tables may be assigned as long as the total number of machine cells does not exceed 4000. Follow all table assignments with a TRA punched in column 8, 9, and 10. The next data required by the subroutine is DATA2 (this is data that is read at the beginning of each trial and valid step) this may be made up with any combination of OCT, BCD, INT cards. All this data is written on a data tape and reserved for future use. If the last TRA card has a R punched in column 12 this data tape will be rewound as soon as the test is made. This is not necessary for the program to work, just more efficient if used.

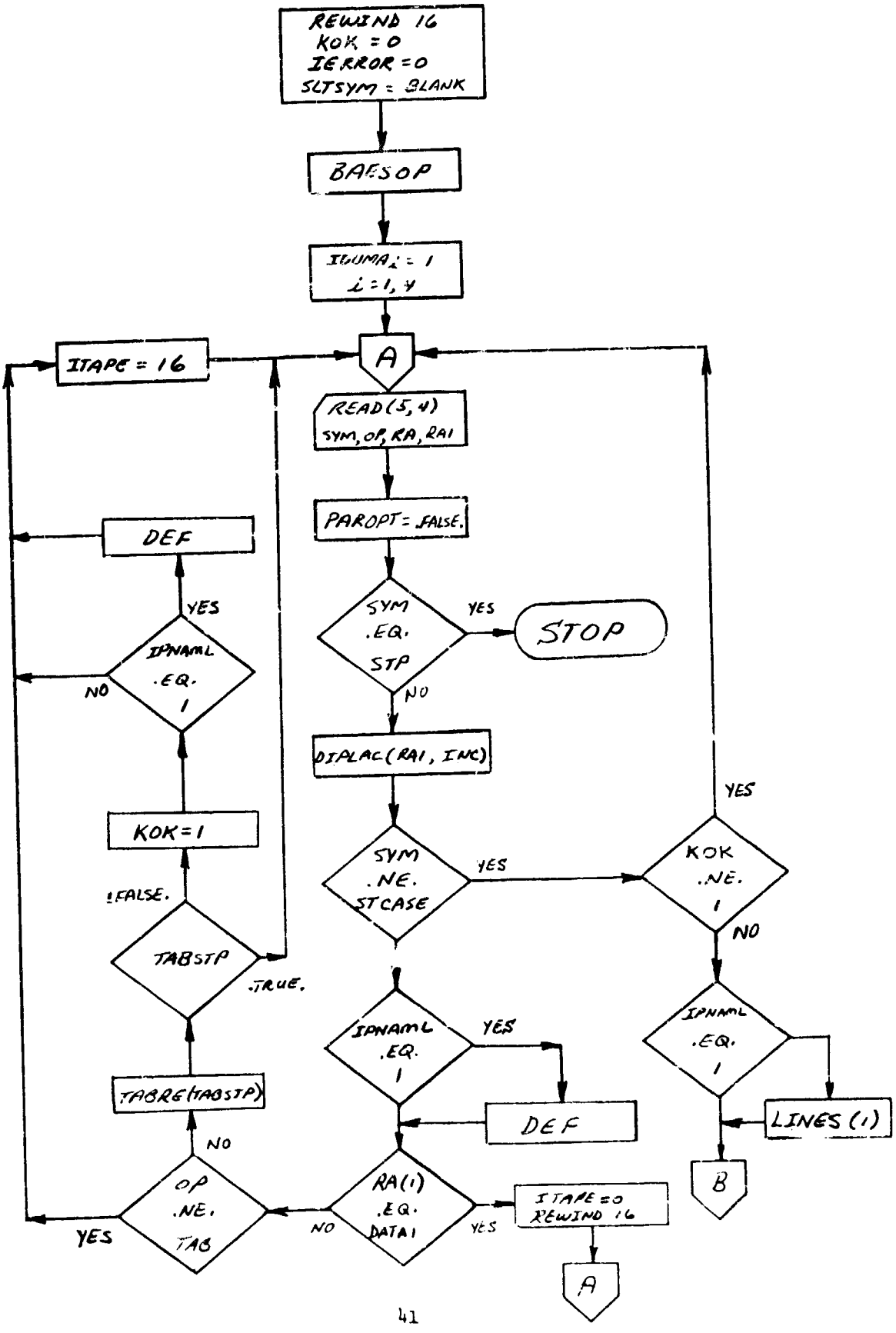
The next set of data expected by the subroutine is DATA1 (this is data which is only read one time per case). This data should begin with:

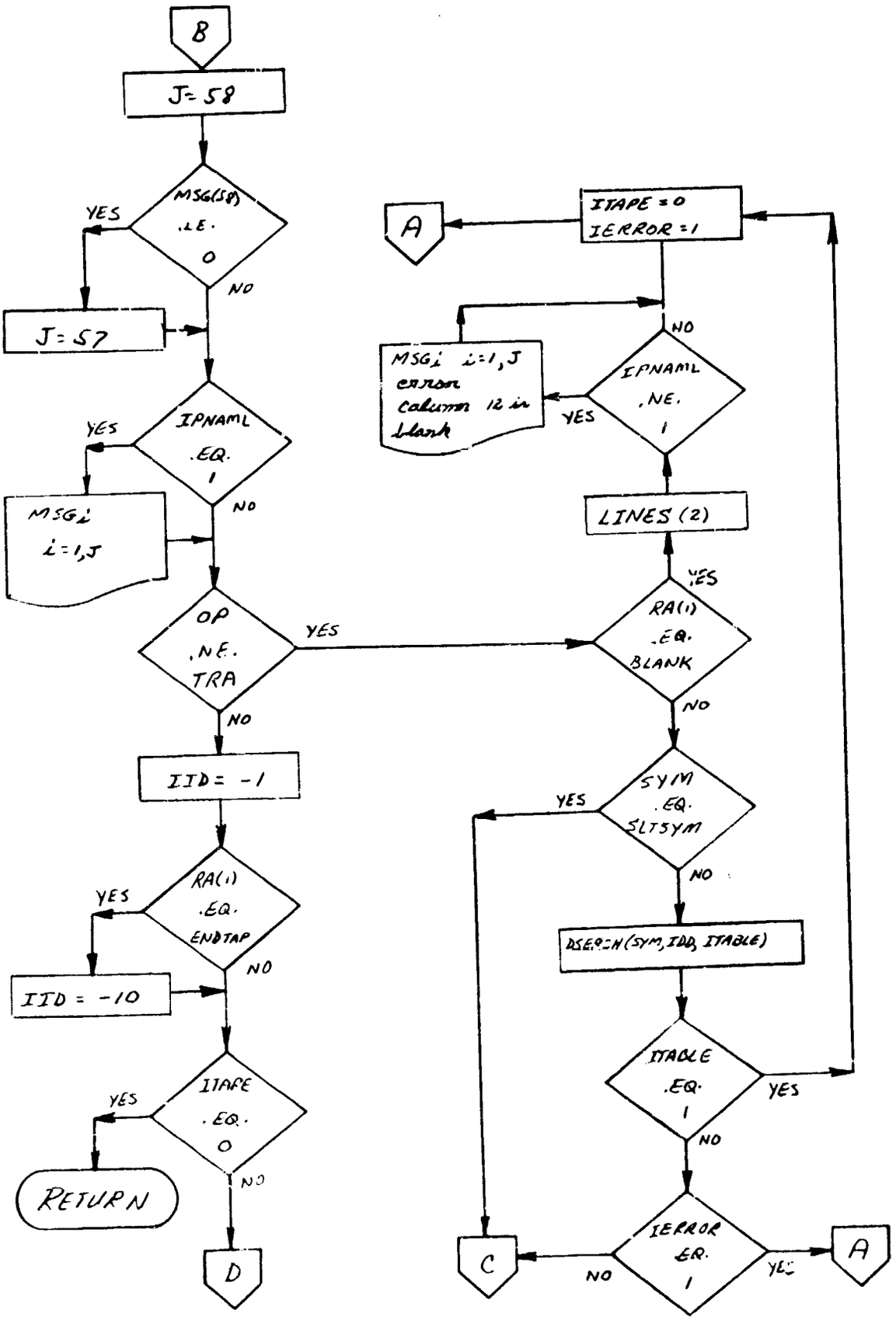
STCASE DATA1

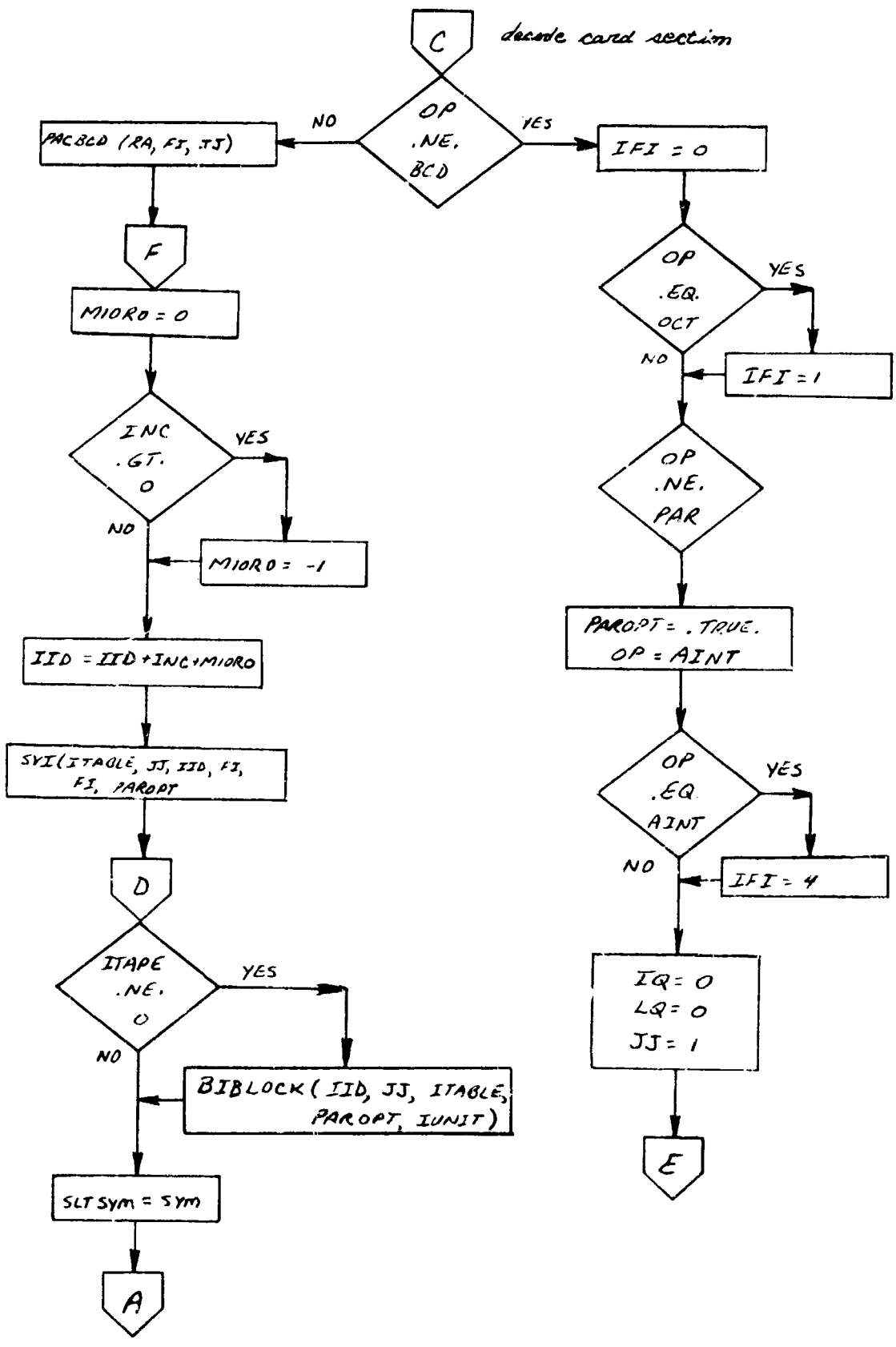
The STCASE is punched beginning in column 1 and DATA1 beginning in column 12. This data is terminated with a TRA beginning in column 8.

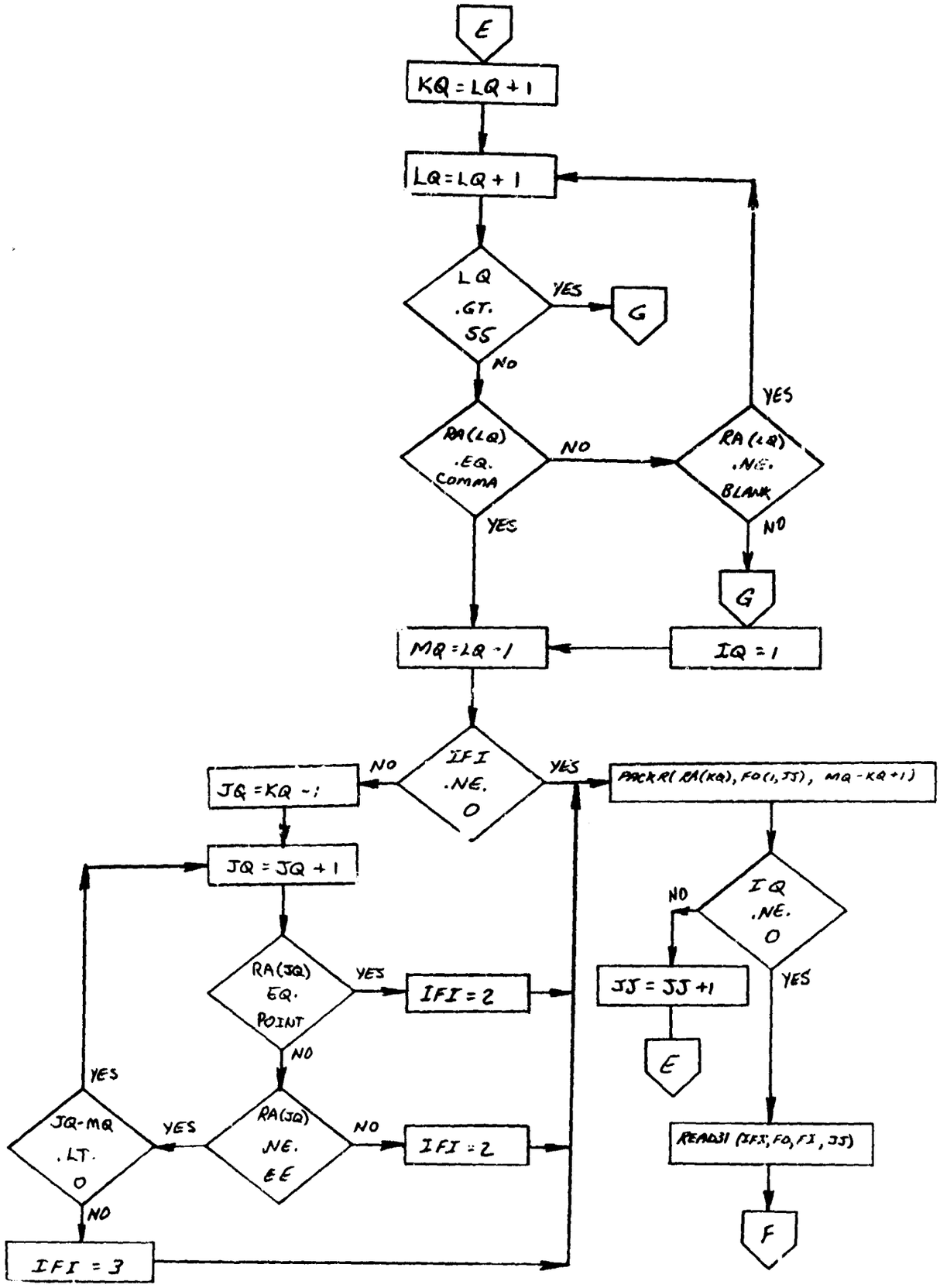
A flow chart for READA is provided. Subroutine READA2 is identical to READA except for the COMMON block, tape units, and auxiliary routines employed.

READA


REWIND 16
KOK = 0
IERROR = 0
SLTSYM = BLANK
BAESOP
IGUMA(I) = 1
I = 1, 4
A
ITAPE = 16
READ(5,4)
SYM, OP, RA, RAI
PAROPT = FALSE.
DEF
SYM .EQ. STP
YES
STOP
NO
IPNAML .EQ. 1
YES
NO
DIPLAC(RAI, INC)
KOK = 1
SYM .NE. STCASE
YES
KOK .NE. 1
YES
NO
!FALSE.
TABSTP
.TRUE.
IPNAML .EQ. 1
YES
IPNAML .EQ. 1
TABRE(ITABSTP)
DEF
LINES(1)
NO
OP .NE. TAB
YES
RA(1) .EQ. DATA1
NO
YES
ITAPE = 0
REWIND 16
B
A


B
J=58
MSG(58) .LE. 0
YES
NO
J=57
IPNAML .EQ. 1
YES
NO
MSGi i=1,J
OP .NE. TRA
YES
NO
IID=-1
RA(1) .EQ. ENDTAP
YES
NO
IID=-10
IIAPE .EQ. 0
YES
NO
RETURN
D
A
ITAPE=0
IERROR=1
MSGi i=1,J error column 12 is blank
IPNAML .NE. 1
YES
NO
LINES(2)
RA(1) .EQ. BLANK
YES
NO
SYM .EQ. SLTSYM
YES
NO
DSERCH(SYM,IDD,ITABLE)
ITABLE .EQ. 1
YES
NO
IERROR .EQ. 1
NO
YES
C
A


C
decode card section
OP .NE. BCD
NO
YES
PACBCD (RA, FI, JJ)
IFI = 0
F
MIORO = 0
OP .EQ. OCT
YES
NO
IFI = 1
INC .GT. 0
YES
NO
MIORO = -1
OP .NE. PAR
IID = IID+INC+MIORO
PAROPT = .TRUE.
OP = AINT
SVI(ITABLE, JJ, IID, FI, FI, PAROPT
OP .EQ. AINT
YES
NO
IFI = 4
D
IQ = 0
LQ = 0
JJ = 1
ITAPE .NE. 0
YES
NO
BIBLOCK (IID, JJ, ITABLE, PAROPT, IUNIT)
E
SLT Sym = Sym
A


E
KQ = LQ + 1
LQ = LQ + 1
LQ .GT. 55
YES
G
NO
RA(LQ) .EQ. COMMA
NO
RA(LQ) .NE. BLANK
YES
NO
YES
G
IQ = 1
MQ = LQ - 1
IFI .NE. 0
NO
JQ = KQ - 1
YES
PACKR(RA(KQ), FO(I,JJ), MQ - KQ + 1)
JQ = JQ + 1
RA(JQ) EQ. POINT
YES
IFI = 2
NO
JQ-MQ .LT. 0
YES
RA(JQ) .NE. EE
NO
IFI = 2
YES
NO
IFI = 3
IQ .NE. 0
NO
YES
JJ = JJ + 1
E
READ3I (IFI, FO, FI, JJ)
F

2. DORDER - Directory Order Output Routine

Purpose:

To provide an ordered listing of the directory on user request.

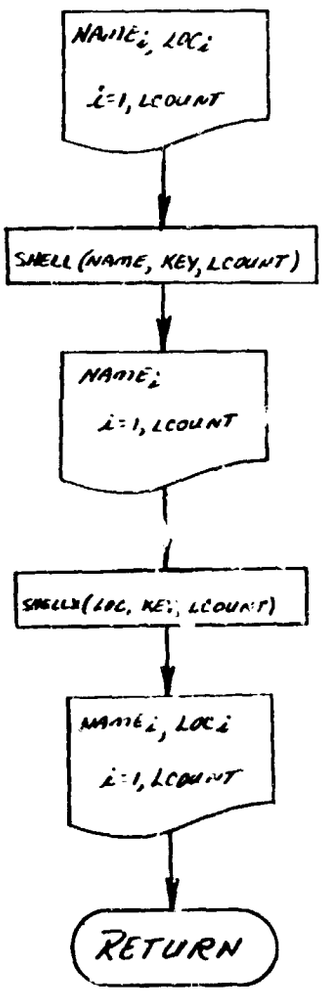
Method:

When the input DLIST = 1, an ordered directory listing is output ahead of the trajectory print. The directory is first sorted in numerical order and printed; then it is sorted in alphabetic order and printed. The sort routines SHELL and SHELLX perform the numeric and alphabetic sorts, respectively.

Remarks:

It is assumed that vehicle 1 and vehicle 2 directories are identical; hence, only one directory order output routine is provided.

DORDER



3. PACKL - Pack Routine

Purpose:

This routine packs two words into one.

Method:

The routine takes the first five characters of one word and the first character of the second word and packs this into one word.

Usage:

Entry is made to this routine by the following statement:

```
CALL PACKL (A,ADOT,D)
```

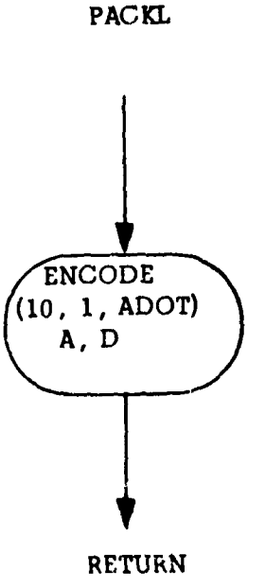
where,

A = the five character word.

D = the single character word.

ADOT = the result of the packed word.

This subroutine uses ENCODE which is a system subroutine. For more information, check ENDODE write-up.



4. DSERCH and DSERCH2- Directory Search Routine for Subscripts

Purpose:

To provide a method of searching the directory to find the subscript corresponding to a BCD argument.

Method:

The routine searches the directory for the exact BCD name required by the argument. When an equal compare has been found, the corresponding subscript is returned as a fixed point integer. An error message will result if the BCD name is not in the directory.

Usage:

Entry is made to the routine with the following statement:

```
CALL DSERCH (SYM,LOC,ICOM)
```

where

SYM = BCD name being searched for.

LOC = The location the corresponding subscript will be stored in.

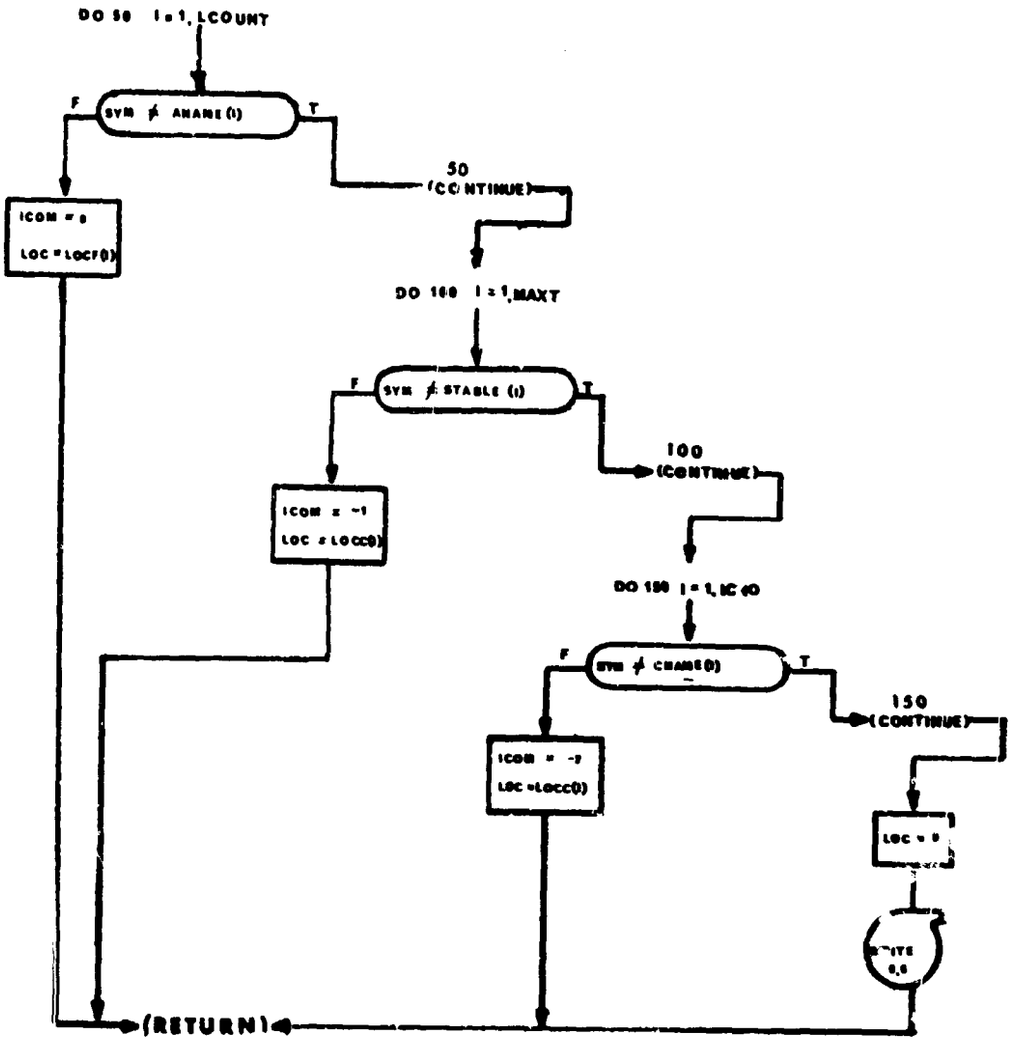
ICOM = Error code in case a compare is not found.

If an error occurs in the BCD name, LOC is set to zero. Location ICOM is set to one and a message is printed stating that the BCD name is not in the directory.

No subroutines are called from these subroutines.

Subroutine DSERCH2 is identical to DSERCH except for the COMMON blocks employed. A flow chart of DSERCH is provided.

DSERCH


DO 50 I = 1, LCOUNT
F
SYM ≠ ANAME (I)
T
50
(CONTINUE)
ICOM = 0
LOC = LOCF(0)
DO 100 I = 1, MAXT
F
SYM ≠ STABLE (I)
T
100
(CONTINUE)
ICOM = -1
LOC = LOCC(0)
DO 150 I = 1, NC (0
F
SYM ≠ CHAR(0)
T
150
(CONTINUE)
ICOM = -2
LOC = LOCC(0)
LOC = 0
(RETURN)

5. STOP1 - General Stop Routine

Purpose:

To print out a stop number and return to the next case.

Method:

To initialize ICHANE, print a stop number and return the program to segment MAIN1 for re-initialization for the next case.

Note: STOP1 was used for the name of this routine because there was a system routine by the name of STOP.

Usage:

Entry is made to this routine by the statement:

```
CALL STOP1
```

If this routine is called a statement is printed:

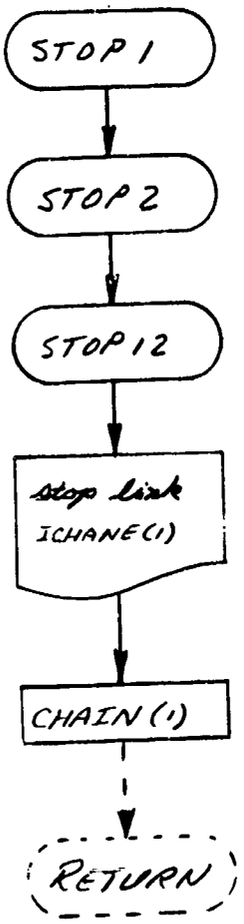
```
STOP LINK N
```

where,

N = The segment link STOP1 was called from

N = 1	MAIN1	segment
N = 2	EXE	segment
N = 3	CTLS	segment
N = 4	REV	segment
N = 5	GRAPH	segment
N = 6	DGAMES	segment
N = 7	CTAE	segment

Subroutines LINES, CHAIN and Normal I/O FORTRAN routines are called from this routine.


STOP 1
STOP 2
STOP 12
stop link
ICHANE(1)
CHAIN (1)
RETURN

6. EXERR - Error Routine

Purpose:

To provide a method of printing a stop number code and ending the execution of a given case.

Method:

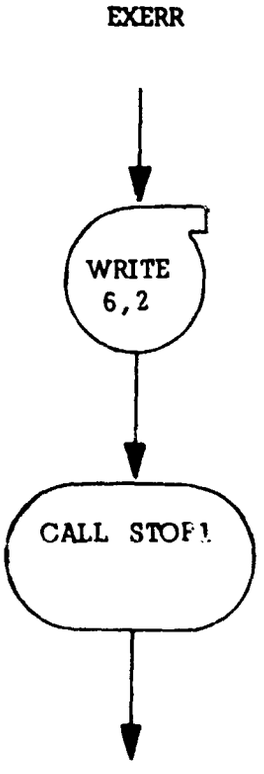
The routine obtains the required stop number from the calling program, prints it, and calls the routine STOP1. The statement STOP NUMBER XXXXX is printed.

Usage:

Entry is made to this routine by the statement:

CALL EXERR(N)

where N is the stop number desired to print.
This routine calls STOP1 and the normal I/O FORTRAN routines.



7. TSRCH and TSRCH2 - Directory Search for Table Subscript

Purpose:

To provide a method of searching the directory for table subscripts.

Method:

The routine searches the directory for the exact BCD name required by the argument. When an equal has been found the corresponding subscript is returned as a fixed point integer.

Usage:

Entry is made to the routine with the following statement:

```
CALL TSRCH (SYM2,LOC2,N2,IER)
```

where

SYM2 = BCD name of argument.

LOC2 = Location of first subscript.

N2 = Number of sequential subscripts to return with.

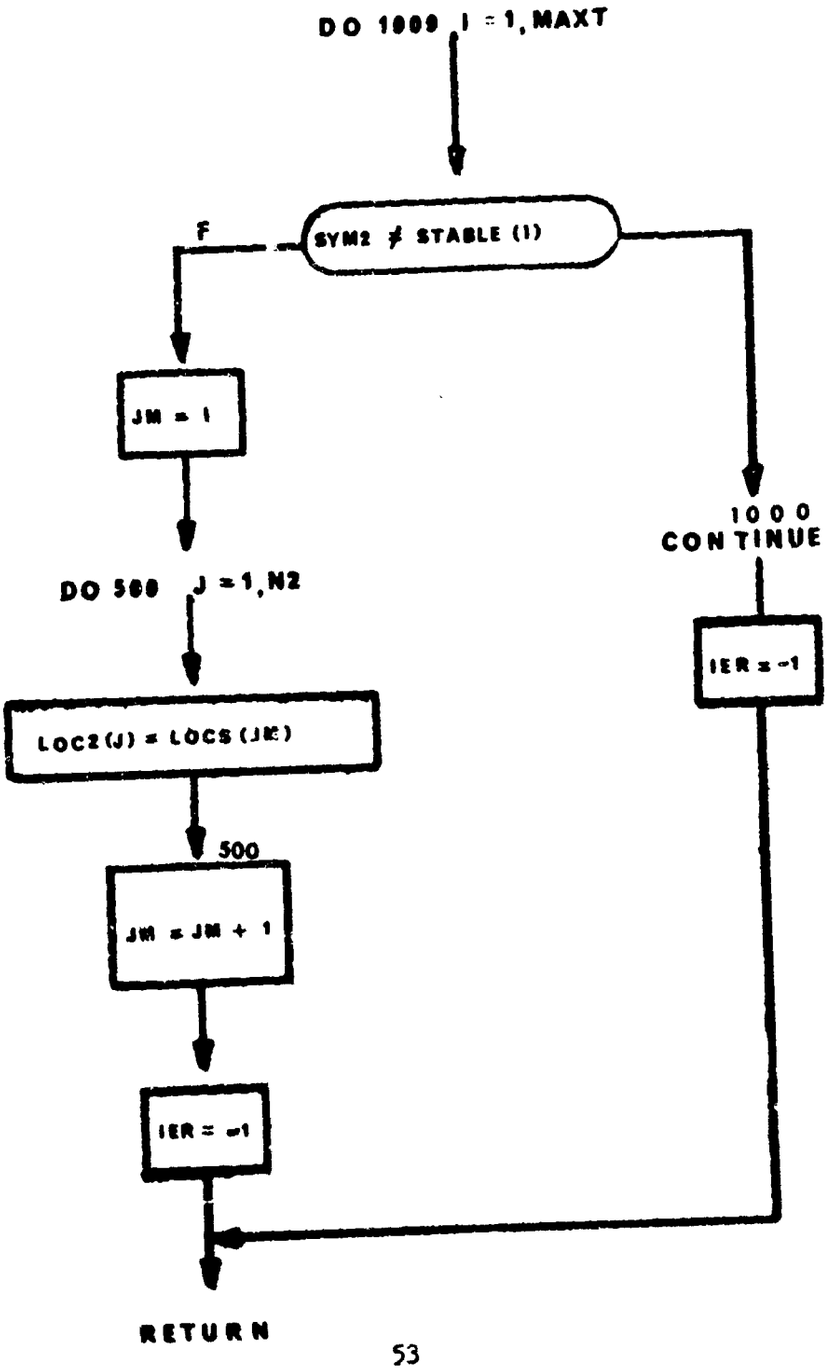
IER = Error Code:

IER is set to a plus 1 if the BCD argument does not compare.
IER is set to a minus 1 if the BCD argument does compare.

No other subroutines are called from these subroutines.

TSRCH2 is identical to TSRCH except for the COMMON blocks employed. A flow chart for TSRCH is provided.

TSRCH


DO 1000 I = 1, MAXT
F
SYM2 ≠ STABLE (I)
JM = 1
DO 500 J = 1, N2
LOC2(J) = LOCS(JM)
500
JM = JM + 1
IER = -1
1000
CONTINUE
IER = -1
RETURN

8. READB and READB2 - Binary Stage Data Input Routines

Purpose:

To read the stage data from the data tape (TAPE16 or TAPE26).

Usage:

```
CALL READB (ISFIN)
```

ISFIN is returned with the following value:

ISFIN < 0 always.

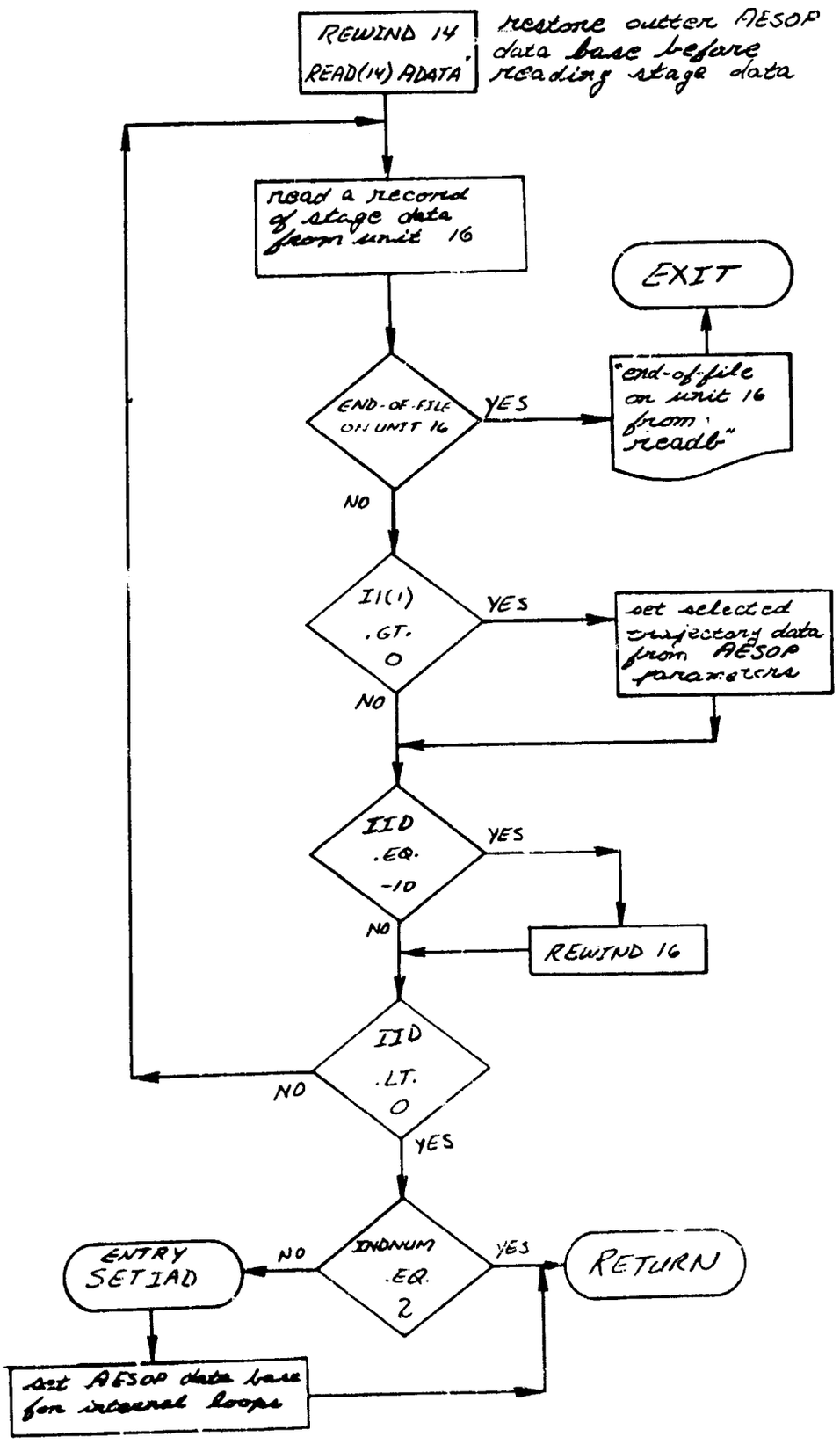
ISFIN = -10 data for the last stage has been read.

READB rewinds TAPE16 when the data for the last stage has been read. For each call to READB, the stage data for one stage is read in. Stage data is ordered sequentially on TAPE16 in the order that the stage data appears in the input deck.

READB may be called only after READA has been called once, since READA prepares TAPE16 (via the ad-hoc blocking routine BIBLOCK).

READB2 is identical to READB except for the tapes and COMMON blocks employed. A flow chart for READB is presented.

READB


REWIND 14
READ(14) ADATA
restore outter AESOP data base before reading stage data
read a record of stage data from unit 16
EXIT
END-OF-FILE ON UNIT 16
YES
"end-of-file on unit 16 from 'readb"
NO
II(1) .GT. 0
YES
set selected trajectory data from AESOP parameters
NO
IID .EQ. -10
YES
NO
REWIND 16
IID .LT. 0
NO
YES
INDNUM .EQ. 2
NO
ENTRY SETIAD
YES
RETURN
set AESOP data base for internal loops

9. BAESOP - Parameter Optimization Input Subroutine

Purpose:

To read in the parameter optimization program AESOP, Volume IV, input data.

Method:

Data is read in conventional FORTRAN NAMELIST manner. NAMELIST name is IAESOP.

10. DIPLAC - Integer Shift Routine

Purpose:

To let the routine right justify a number so that the displacement (col. 67-72) on an input card may be punched anywhere in the field.

Method:

The displacement field is read into the machine with an A format and right justified before the conversion is made to a binary integer.

Usage:

Entry is made to this routine with the following statement:

CALL DIPLAC (RA1,INC,BLANK)

where,

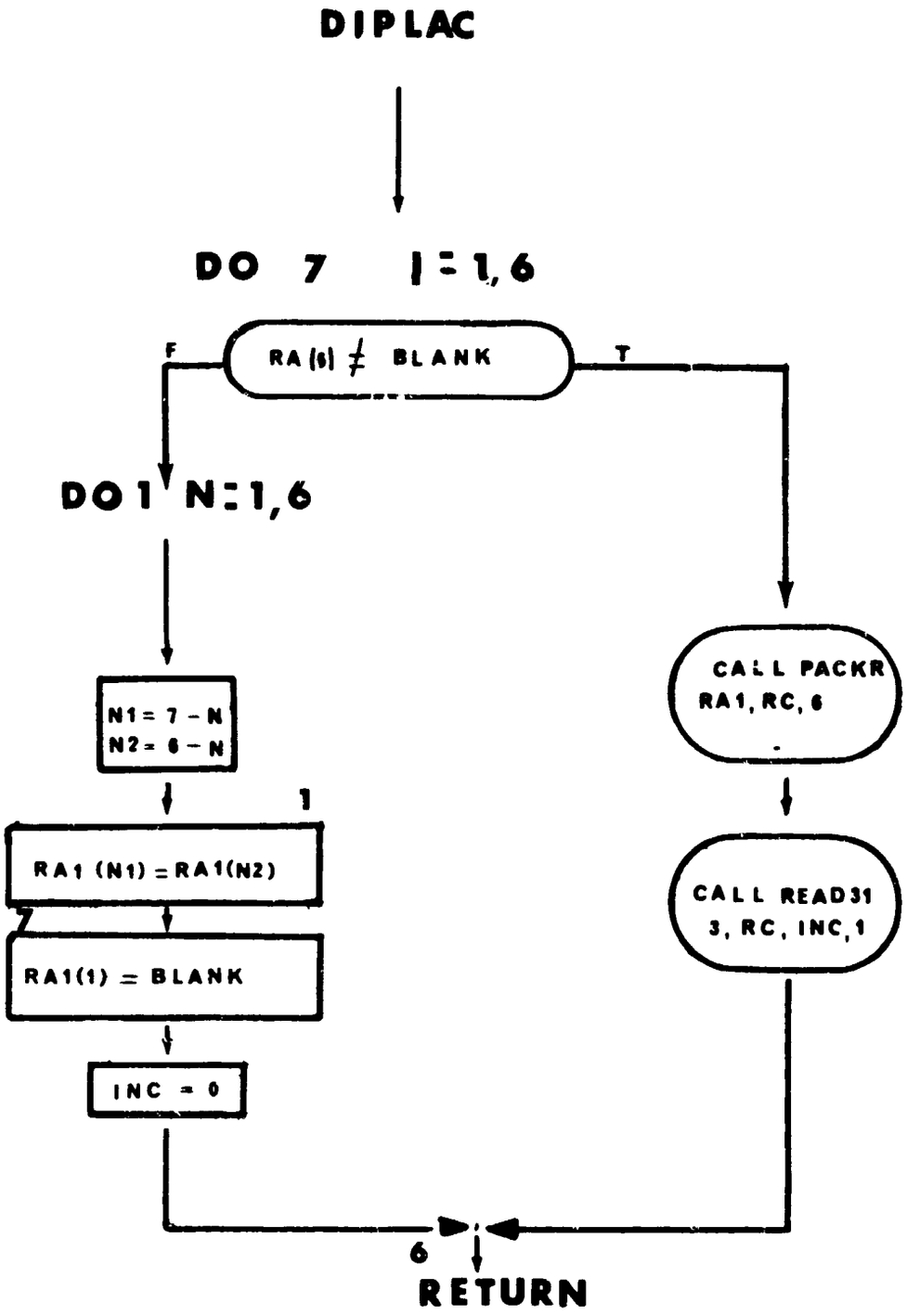
RA1 - The six character BCD array.

INC - Location that the converted integer will be stored in.

BLANK - Blank character used for comparing.

Subroutines Called:

PACKR READ31


DIPLAC
DO 7 I = 1,6
F
RA(I) ≠ BLANK
T
DO 1 N = 1,6
N1 = 7 - N
N2 = 8 - N
1
RA1 (N1) = RA1(N2)
RA1(1) = BLANK
INC = 0
CALL PACKR
RA1, RC, 6
CALL READ31
3, RC, INC, 1
6
RETURN

11. DEF and DEF2 - Heading and Page Eject

Purpose:

To provide page ejection and title print.

Method:

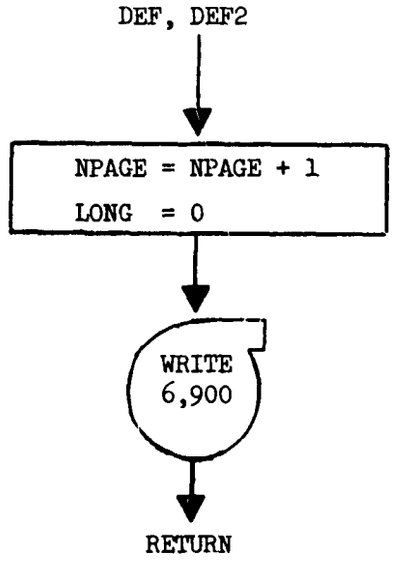
Initially the current page number (NPAGE) is incremented by 1. The page is ejected and return is made to the calling program.

Usage:

Entry is made by the following statements:

CALL DEF or CALL DEF2

Only the normal I/O FORTRAN routines are used with this routine. DEF2 is an entry point used by vehicle 2.



12. TABRE and TABRE2 - Table Dimension Subscript Routines

Purpose:

This subroutine is called from READA or READA2 and computes subscripts such that the table dimension requirements may be variable.

Method:

Uses input data prepared by the user to compute subscripts for variable table assignments.

Usage:

This subroutine is called from subroutine READA or READA2 one time per case, and linkage is obtained by:

```
CALL TABRE (TABSTP)
```

where TABSTP is an indicator set false by TABRE or TABRE2 when READA or READA2 will not recall TABRE or TABRE2.

Subroutines Called:

DIPLAC LINES

Error Messages:

1. Symbol does not exist in table list.
2. Total table size N exceeds maximum size N1 where N is the required and N1 is the maximum.

Data Preparation:

Control will be transferred to subroutine TABRE or TABRE2 when READA or READ processes a control card:

STCASE TAB

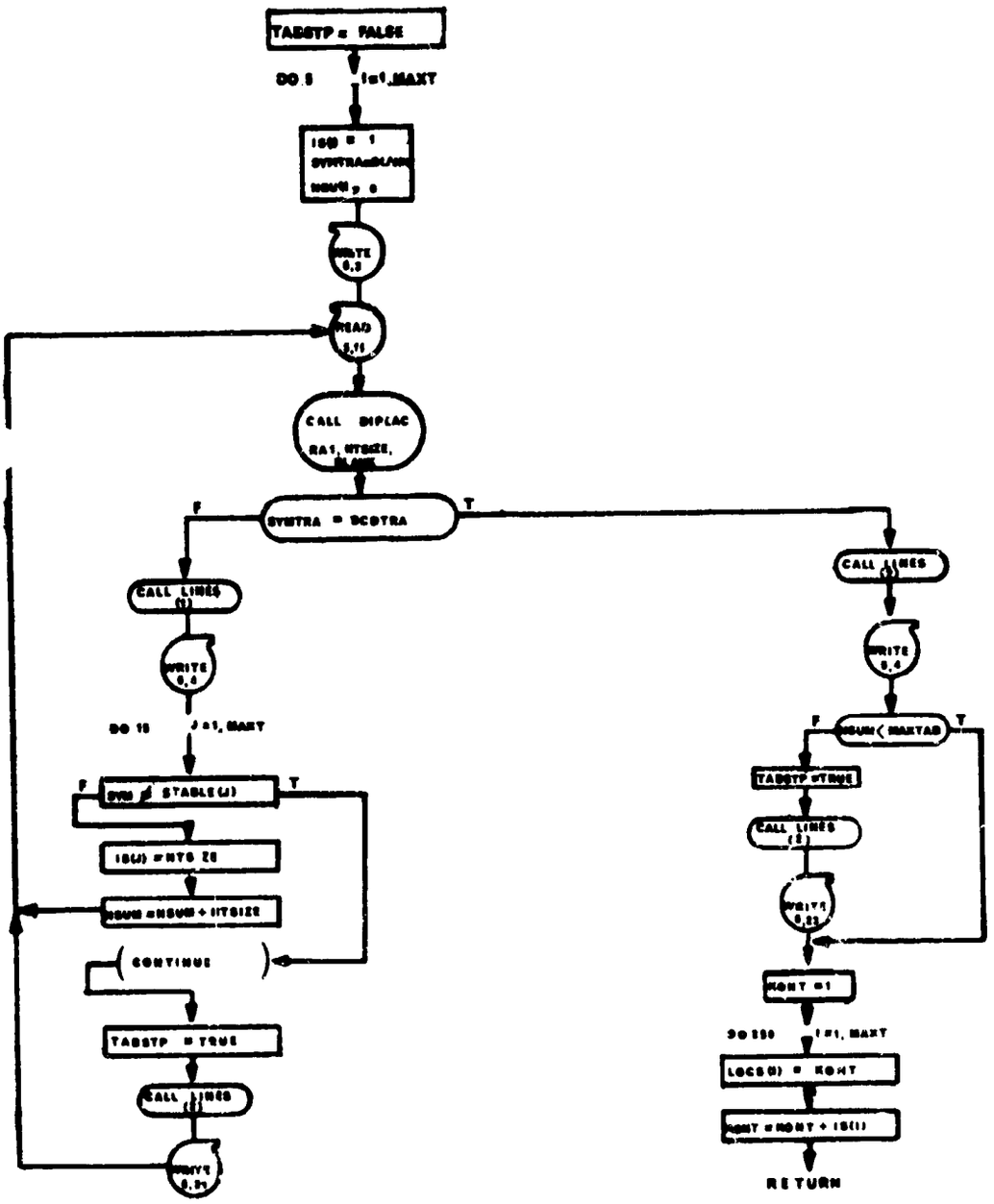
STCASE beginning in column 1 and TAB punched in column 8, 9 and 10. Following this card will be the cards requesting table sizes.

TTABO1 10

ATABO1 20

TTABO1 and ATABO1 punched beginning in column 1 and the required machine cells (10 and 20 in this case) punched beginning in column 12. Anything punched past column 15 will not be used. After all table assignments a TRA should be punched in column 8, 9 and 10.

TABRE


TABSTP = FALSE
DO 5
I=1,MAXT
WRITE
READ
CALL DIPLAC
SYMTRA = SCDTRA
F
T
CALL LINES
WRITE
DO 15
J=1, MAXT
STABLE(J)
CONTINUE
TABSTP = TRUE
CALL LINES
WRITE
CALL LINES
WRITE
NSUM < MAXTAB
TABSTP = TRUE
CALL LINES
WRITE
KONT = 1
I=1, MAXT
LOCS(I) = KONT
KONT = KONT + IS(I)
RETURN

13. LINES and LINES2 - Lines Accounting Routines

Purpose:

To keep an accounting of the number of lines printed per page, and to provide for page control.

Method:

If the number of lines to be printed (LCOUNT) is such that it will not fit on the current page, the page is ejected (via DEF) and printing will begin on the new page. Initially, the location LONG, should be set to zero, indicating that currently no lines have been printed on the present page.

Usage:

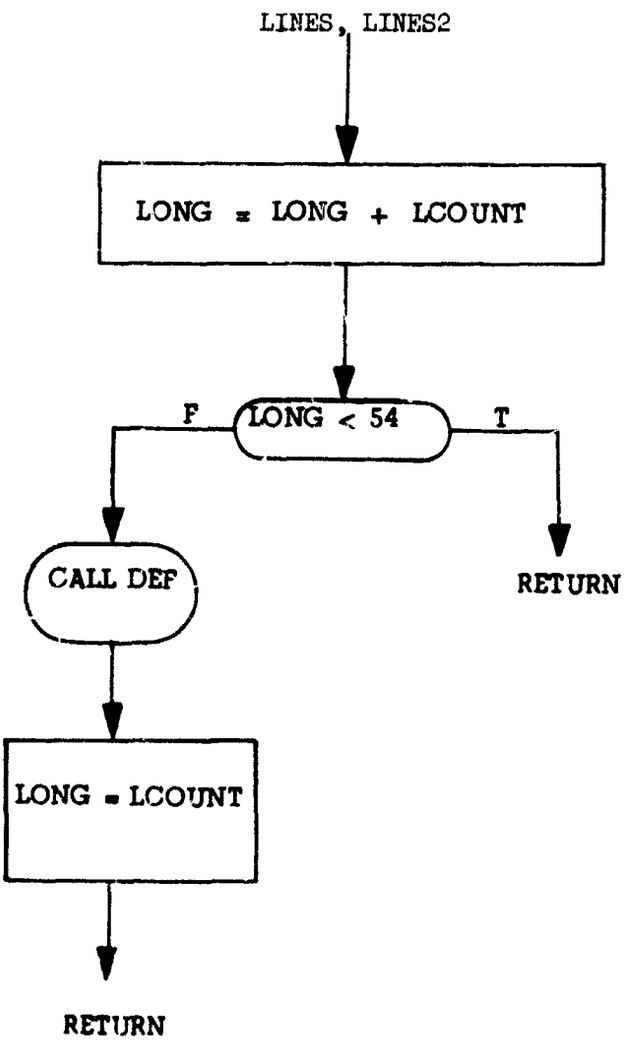
Entry is made to the routine by the following statement:

CALL LINES (LCOUNT) or CALL LINES2 (LCOUNT)

where,

LCOUNT = A fixed point variable or constant indicating the number of lines to be printed.

Subroutine DEF is called from this routine. LINES2 is an entry point used by the second vehicle.


LINES, LINES2
LONG = LONG + LCOUNT
F
LONG < 54
T
CALL DEF
RETURN
LONG = LCOUNT
RETURN

14. PACBCD - Packs Six Character Words

Purpose:

To pack BCD words into six character words.

Method:

The first character is converted to an integer by the system routine, DECODE. This integer is the number of six character words contained on the card. ENCODE is then used to pack that number of six character BCD words.

Usage:

Entry is made to this routine by the following statement:

```
CALL PACBCD (RA,FI,JJ)
```

where,

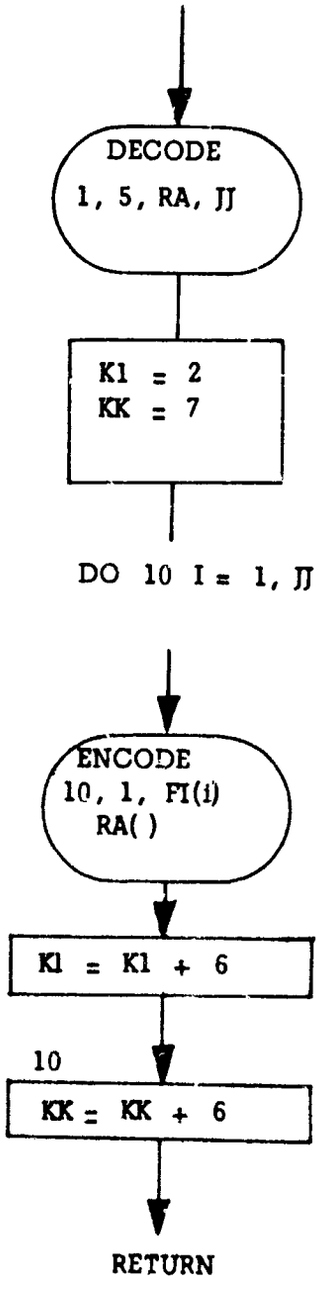
RA - is the first location of the array being converted.

FI - is the first location where the results will be stored.

JJ - is the number of six character words.

This subroutine uses ENCODE and DECODE which are system routines. For more information, check write-up for ENCODE and DECODE.

PACBCD


DECODE
1, 5, RA, JJ
K1 = 2
KK = 7
DO 10 I = 1, JJ
ENCODE
10, 1, FI(i)
RA( )
K1 = K1 + 6
10
KK = KK + 6
RETURN

15. PACKR - Packs BCD Characters

Purpose:

Packs BCD Characters into words.

Method:

This routine uses ENCODE to convert BCD information to a binary integer. No I/O transmission takes place.

Usage:

Entry is made to this routine by the statement:

```
CALL PACKR (I1,I2,N)
```

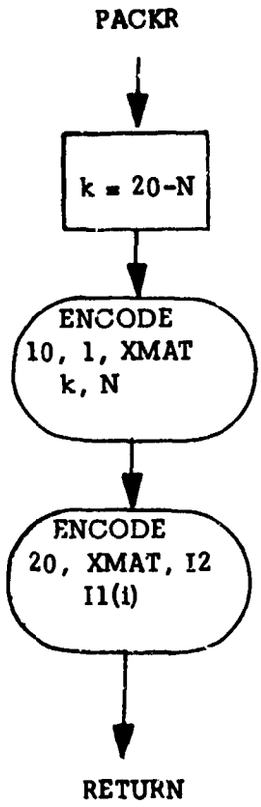
where,

I1 - Contains the BCD integers to be converted.

I2 - Location of where converted numbers are to be restored.

N - Number of words to be converted.

This routine uses ENCODE which is a system routine. For more information check ENCODE write-up.


PACKR
k = 20-N
ENCODE
10, 1, XMAT
k, N
ENCODE
20, XMAT, I2
I1(1)
RETURN

16. READ31 - Binary Conversion Routine

Purpose:

To convert from BCD to Octal, Floating Point and Integer numbers.

Method:

This routine takes the BCD display code and converts it to the necessary binary data depending on the operation code.

Usage:

Entry is made to this routine by the following statement.

```
CALL READ31 (IFI,FJ,FI,JJ)
```

where,

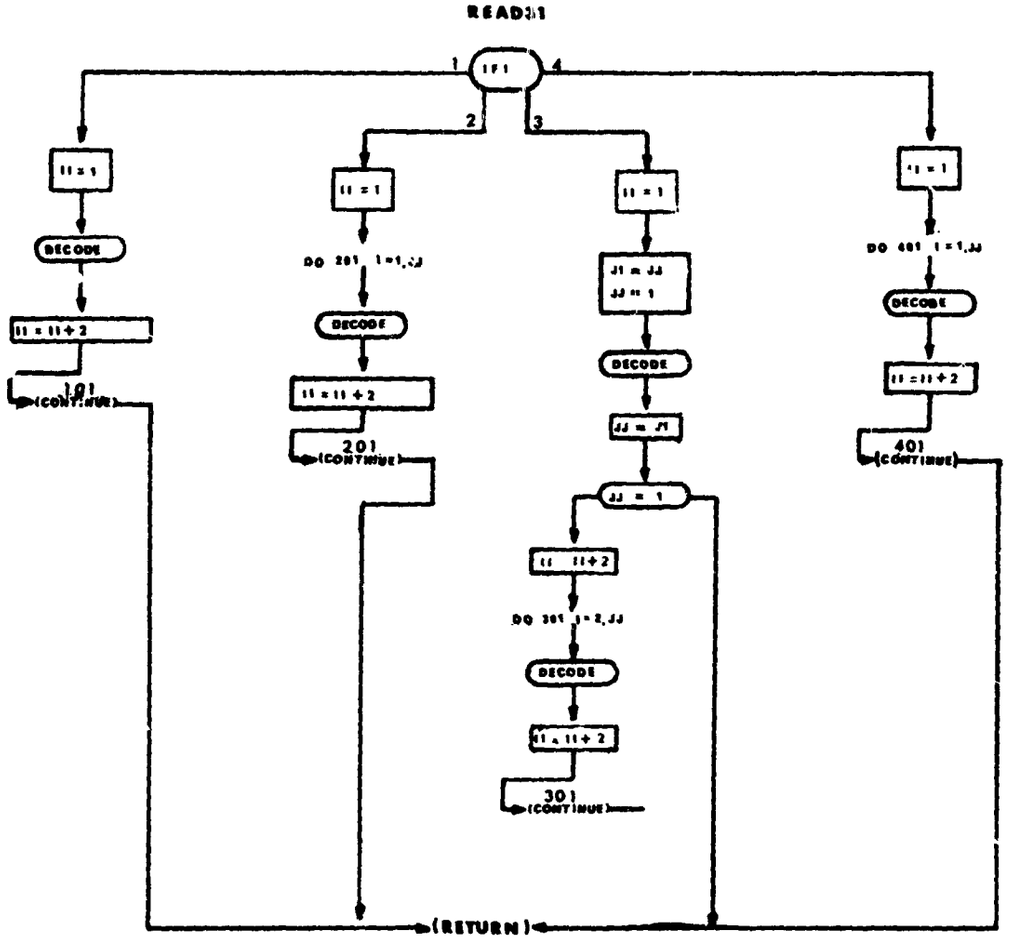
IFI - is a code that determines if its Octal, Floating Point or Integer conversion.

FI - is the location where the answers are stored.

FJ - contains the BCD characters to be converted.

JJ - the number of words to be converted.

This subroutine uses the system routine DECODE. For more information on DECODE check the write-up for DECODE.


READ31
IF1
1
2
3
4
II = 1
DECODE
II = II + 2
101
(CONTINUE)
II = 1
DO 201 I=1,JJ
DECODE
II = II + 2
201
(CONTINUE)
II = 1
J1 = JJ
JJ = 1
DECODE
JJ = J1
JJ = 1
II = II+2
DO 301 I=2,JJ
DECODE
II = II + 2
301
(CONTINUE)
II = 1
DO 401 I=1,JJ
DECODE
II = II + 2
401
(CONTINUE)
(RETURN)

17. SVI and SVI2 - Block Save Routines

Purpose:

To place a number of variables into an array. The location of each piece of data is contained in any array. The routines also equate data flagged by the PAR input subject to the appropriate optimizing parameter.

Method:

Each piece of data is picked up sequentially beginning with A and stored into an array beginning with the subscript ISTART.

Usage:

Entry is made to the routine with the following statement.

CALL SVI (ICOM,N,ISTART,A, IA,PAROPT)

where,

ICOM = Type of data being saved.

N = Number of words being saved.

ISTART = Common subscript for A.

A = Location where value is being moved from.

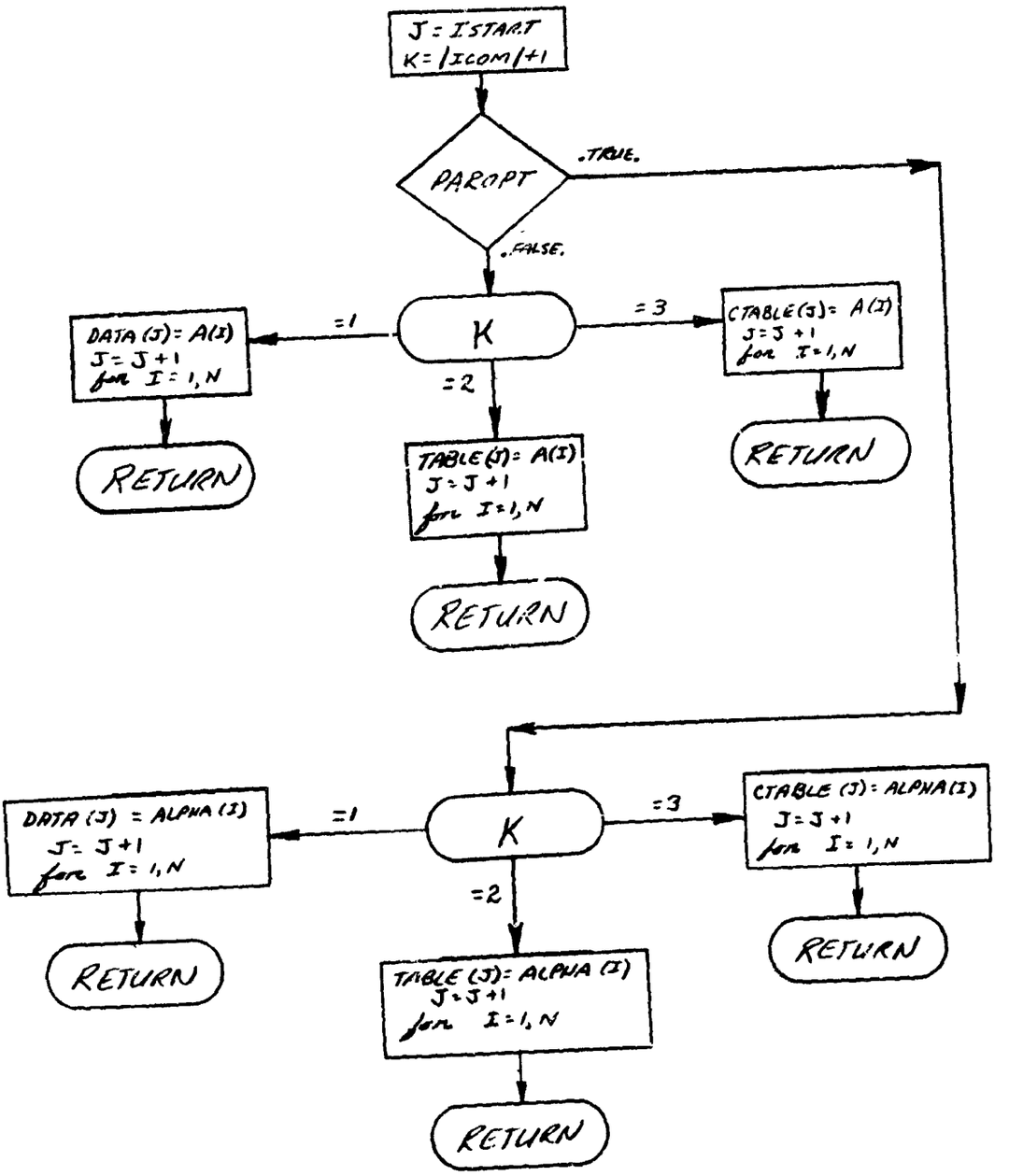
IA =Optimization parameter suffix area.

PAROPT =Flag indicating data is to be set equal to an optimizing parameter.

No other routines are called from this routine.

Subroutine SVI2 is identical to SVI except for certain COMMON blocks. A flow chart is presented for SVI.

SVI


J = ISTART
K = ICOM + 1
PAROPT
.TRUE.
.FALSE.
K
=1
=2
=3
DATA (J) = A(I)
J = J + 1
for I = 1, N
RETURN
TABLE (J) = A(I)
J = J + 1
for I = 1, N
RETURN
CTABLE (J) = A(I)
J = J + 1
for I = 1, N
RETURN
K
=1
=2
=3
DATA (J) = ALPHA (I)
J = J + 1
for I = 1, N
RETURN
TABLE (J) = ALPHA (I)
J = J + 1
for I = 1, N
RETURN
CTABLE (J) = ALPHA (I)
J = J + 1
for I = 1, N
RETURN

18. BIBLOCK - Data Output in Blocks

Purpose

To prepare and output data in large blocks rather than a card at a time.

Method

Three dimension arrays are used by this routine to input data too. The data is read into the machine in BCD and then decoded depending on the pseudo-operation punched on the card. After decoding all data is treated as integers, then stored in one of the three dimension arrays. Just prior to this call, a subscript is stored which defines which cell in the array the number will be stored into when it is read back into the machine. When either of three buffers are filled, all three buffers are flushed out on tape. If anyone of the buffers fail to have anything stored in it a dummy cell is set up. This is necessary because of the way the Fortran system works. It is impossible to store zero words on tape.

Usage

Linkage is made by the following call:

CALL BIBLOCK(IID,JJ,FI,ITABLE)

where

IID = the subscript which relocates the integer in core.

JJ = the number of integers to be stored in the array.

FI = the integer to be stored in array.

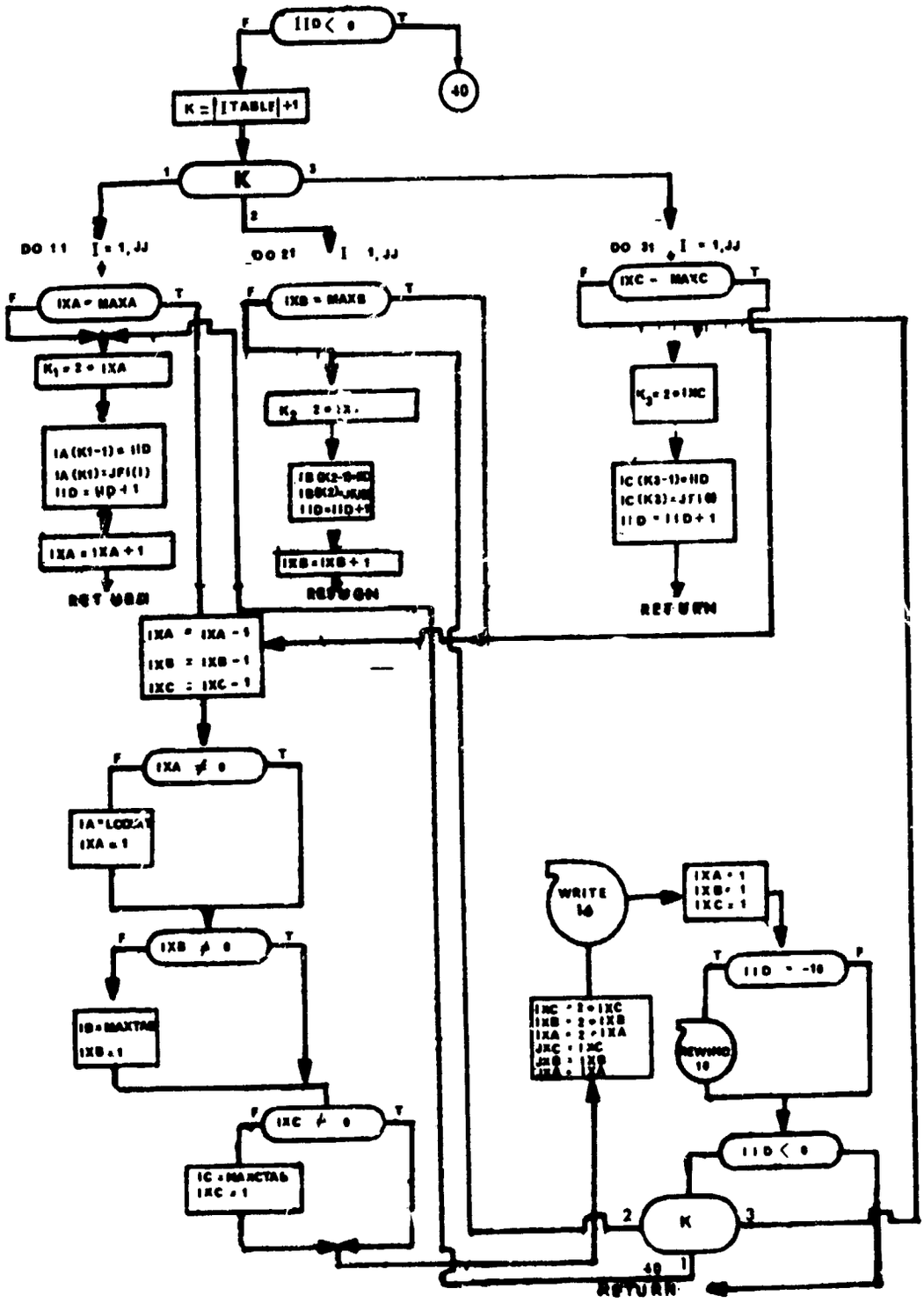
ITABLE = contains a number defining which of the three arrays to store into. In this case ITABLE is either 0, 1, or 2.

No initialization or printing is necessary for this routine.

Subroutines Called

The only routines called are the normal output FORTRAN routines.

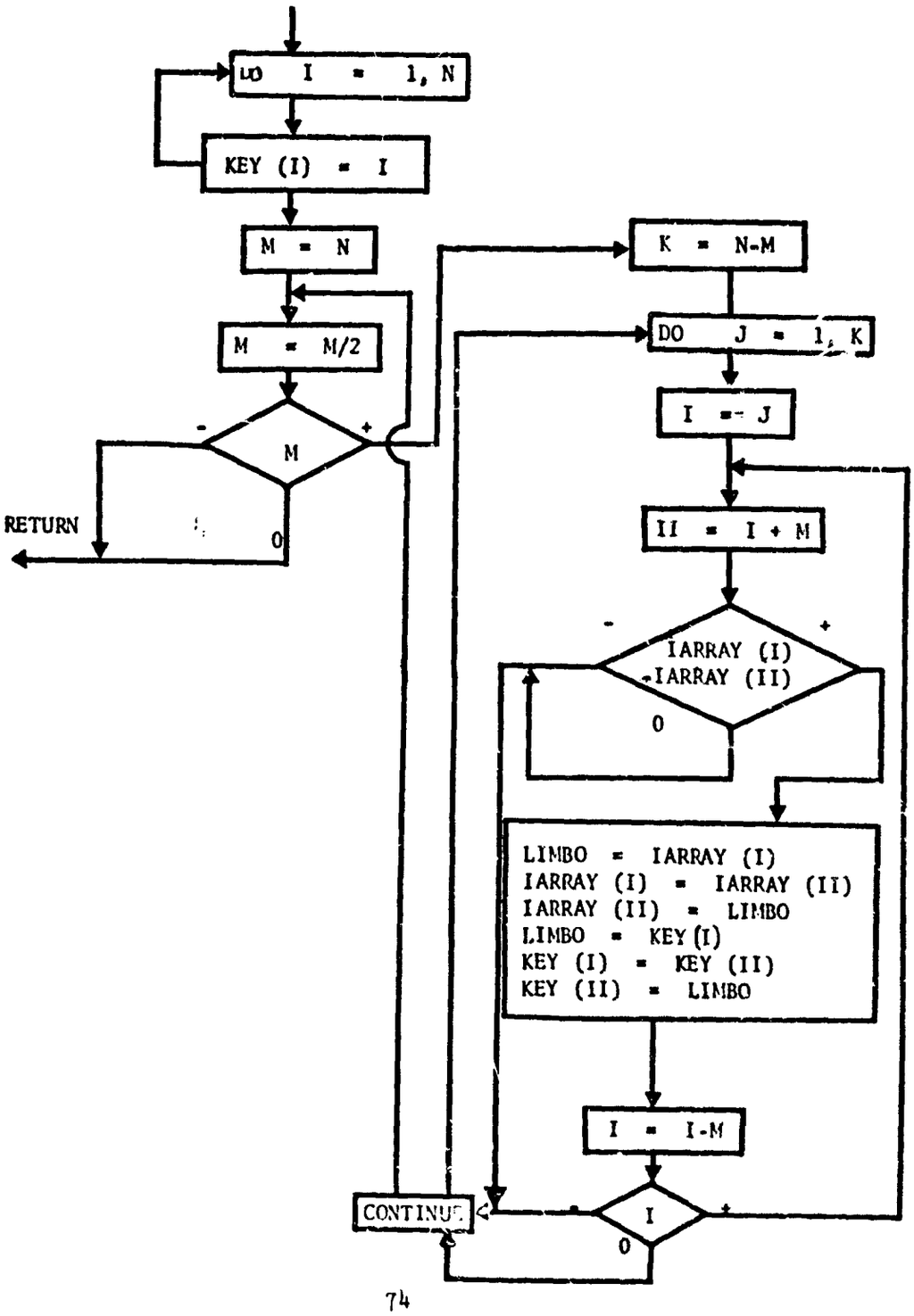
BIBLOCK


IID < 0
40
K = |TABLE| + 1
K
DO 11 I = 1, JJ
DO 21 I = 1, JJ
DO 31 I = 1, JJ
IXA = MAXA
IXB = MAXB
IXC = MAXC
IXA = IXA - 1
IXB = IXB - 1
IXC = IXC - 1
IXA ≠ 0
IXB ≠ 0
IXC ≠ 0
WRITE
IID = -10
REWIND
IID < 0
K
RETURN

19. SHELL - Numeric Sorting Subroutine

Purpose:

To store an array in ascending order.



SECTION VI

PROGRAM EXE AND SUBROUTINE EXE2

EXE is the executive program that drives both vehicle forward trajectories. EXE controls the major logical decisions that must be made. The subprograms of EXE are responsible for accomplishing the calculations which fall into their respective domains of specialization. Since, during any one call, it is not feasible for a subprogram to do all types (e.g. initialization, printing, function calculations, etc.) of computations delegated to it, the subprograms are segmented into functional units; access to a particular functional unit of a subprogram is accomplished by calling that subprogram with an argument called the entry point. For most of the subprograms of EXE there is the following correspondence between entry points and functional units:

Entry Point	Functional Unit
1	pre-data initialization (to be done at the beginning of each trajectory before data is read)
2	post-data initialization and/or initial transformation (to be done at the beginning of each stage after data is read)
3	main calculation (e.g. calculate derivatives)
4	initial print; generally this is for print of that which is calculated at entry point 2.
5	code print; to identify the values printed at entry point 6.
6	value print; for printing time history of variables calculated during trajectory.
7	for specifying variables that are to be integrated. Each variable that is to be integrated should have its derivative computed at entry point 3.
8	h-transformation (or miscellaneous chores)

In addition to the so called "standard subprograms" of EXE there are a few which are of such a special nature that the standard entry points no longer apply, or else perhaps an additional entry point is included for some

special purpose. Those subprograms which have multiple entry points which deviate from the standard are MIMINF, INTGRT, EXTRAN, PLTS, MANTGT, and EXE2.

The structure of EXE is integral with the structure for accomplishing the numerical integration. This is necessitated by the premise that EXE should have control. As a result, the numerical integration routine, MIMINF, is a slave to EXE. The relationship between MIMINF and EXE differs distinctly from that of the standard subprograms and EXE. The MIMINF writeup contains a complete description of the entry points and their respective functions. EXTRAN is a special subprogram which is in reality a subdriver for accomplishing the initial transformation and the h-transformation.

One great benefit of this type of organization of EXE is that it is general; any trajectory program may be set up this way. In particular, this fact was used for the programming of the reverse trajectory. REV is the executive program that drives the reverse trajectory and hence the structure of REV is virtually isomorphic to EXE.

Communication between EXE and its subprograms is accomplished by setting indicators. However, of all the indicators used in EXE proper, EXE sets most of them itself. The following is a list of all those indicators that are set by subprograms of EXE for the purpose of communication with EXE proper.

Indicator	Subprogram
MIMPAS and MIMPAS2	MIMINF and MIMINF2
INDPAR	MANTGT
INGHOM and INGHOM2	STGTST and STGTST2
INDSTE and INDSTE2	any subprogram (denotes error)
INDSTG and INDSTG2	STGTST and STGTST2
LOOK and LOOK2	STGTST and STGTST2

One of the most critical functions of EXE is to control the integration step size. EXE itself actually does very little of the calculation for the step size. Yet, it governs the sequence of calls that are made to subprograms which determine a step size to meet their respective requirements. Exe must then weigh each factor against the other and select that step size which is "most reasonable". A summary of the step size control procedure follows:

The logic in EXE that controls the integration step size, HO, during the forward trajectory is necessarily somewhat complicated because of all the factors that have to be considered. "Staging" and/or "time points" may require the step size to be either cut back or increased. Depending on the situation the integrated variables may have to be backed up one step.

MIMINF(5) and MIMINF2(5) are the entries to the integration routines which check the truncation error; STGTST(3) and STGTST2(3) are the entries to the stage test routines which check the staging variable (s) to see when they pass from one side to the

other of their respective staging values. The MIMINF and STGTST writeups should be referred to if it is desired to know how these routines determine the step size when they are granted this responsibility.

The following points summarize the logic involved in controlling the step size:

(1) An integration step is a "trial step" until it has been accepted as valid by MIMINF(5), MIMINF2(5), STGTST(3), and STGTST2(3)

(2) If MIMINF(5) or MIMINF2(5) rejects the trial step, then the integrated variables are backed up one step,and a new trial step is made with the step size (HO) which MIMINF(5) and MIMINF2(5) have determined.

(3) Only after MIMINF(5) and MIMINF2(5) accept a trial step does STGTST(3) and STGTST2(3) have the opportunity to reject it. If STGTST(3) or STGTST2(3) reject a trial step, then the integration is backed up one step, and a new trial step is made with the step size (HO) which STGTST(3) or STGTST2(3) have determined.

(4) It is assumed that the HO determined by (2) or (3) when a trial step is rejected, is *less than* the HO which produced the bad step.

(5) After a valid integration step has been taken, the next time point, TIMPT, is picked up. TIMPT will be the first time point greater than TIMES + AMINER.

(6) The next trial step *must* have an HO $\leq$ TIMPT - TIMES.

(7) After a valid step, the HO for the next trial step will have been determined or at least sanctioned by MIMINF(5) and MIMINF2(5). Paragraph (6) merely imposes a further condition in addition to MIMINF(5) and MIMINF2(5).

(8) Once staging has started; i.e., once STGTST(3) or STGTST2(3) has rejected a trial step, MIMINF(5) and MIMINF2(5) will be *passive*. This means that MIMINF(5) and MIMINF2(5) accept every trial step and compute no new HO.

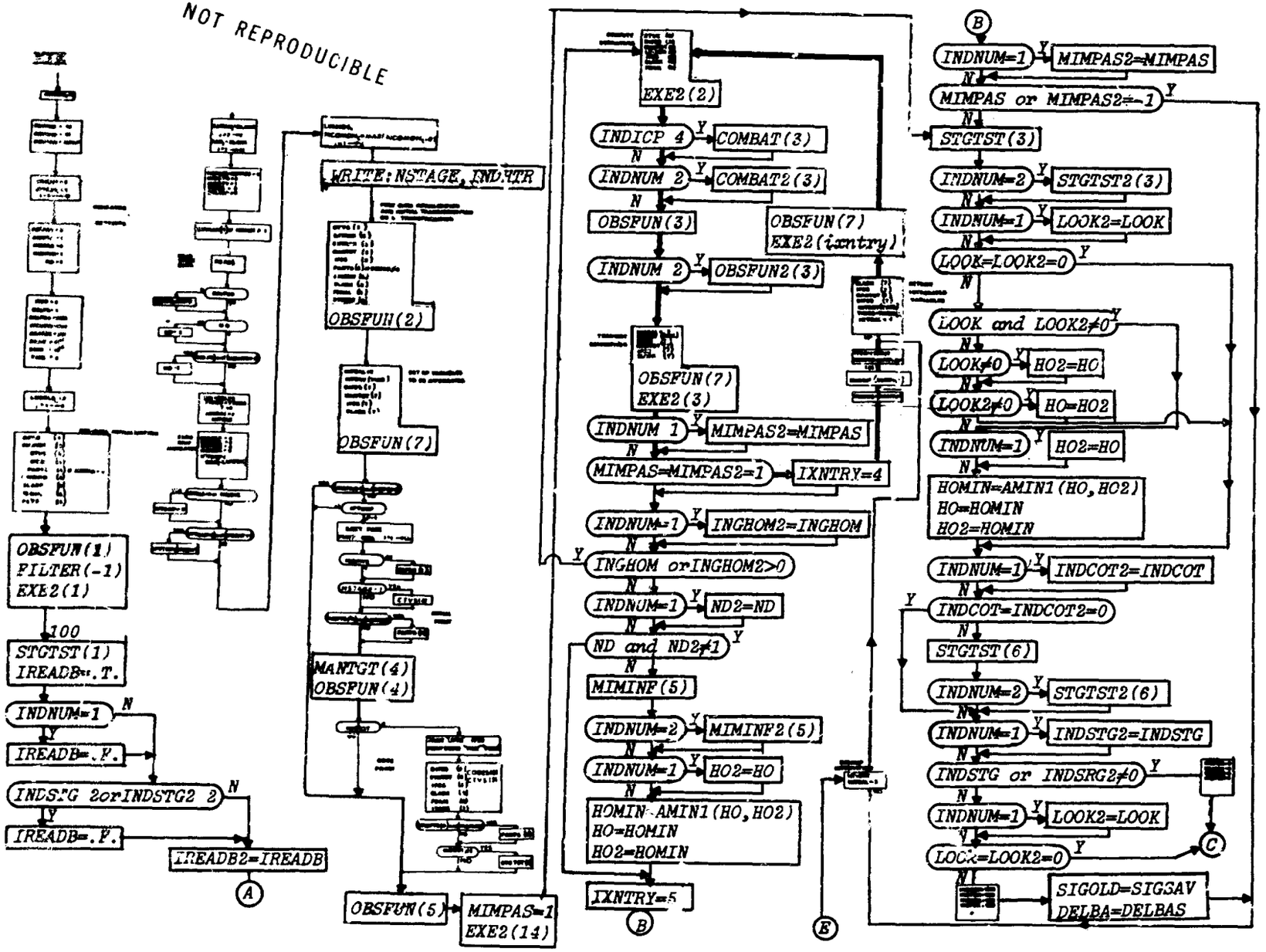
(9) MIMINF(5) and MIMINF2(5) have *sole control over increasing* the step size HO; MIMINF(5) or MIMINF2(5) may increase HO only if they accept the trial step and are not passive.

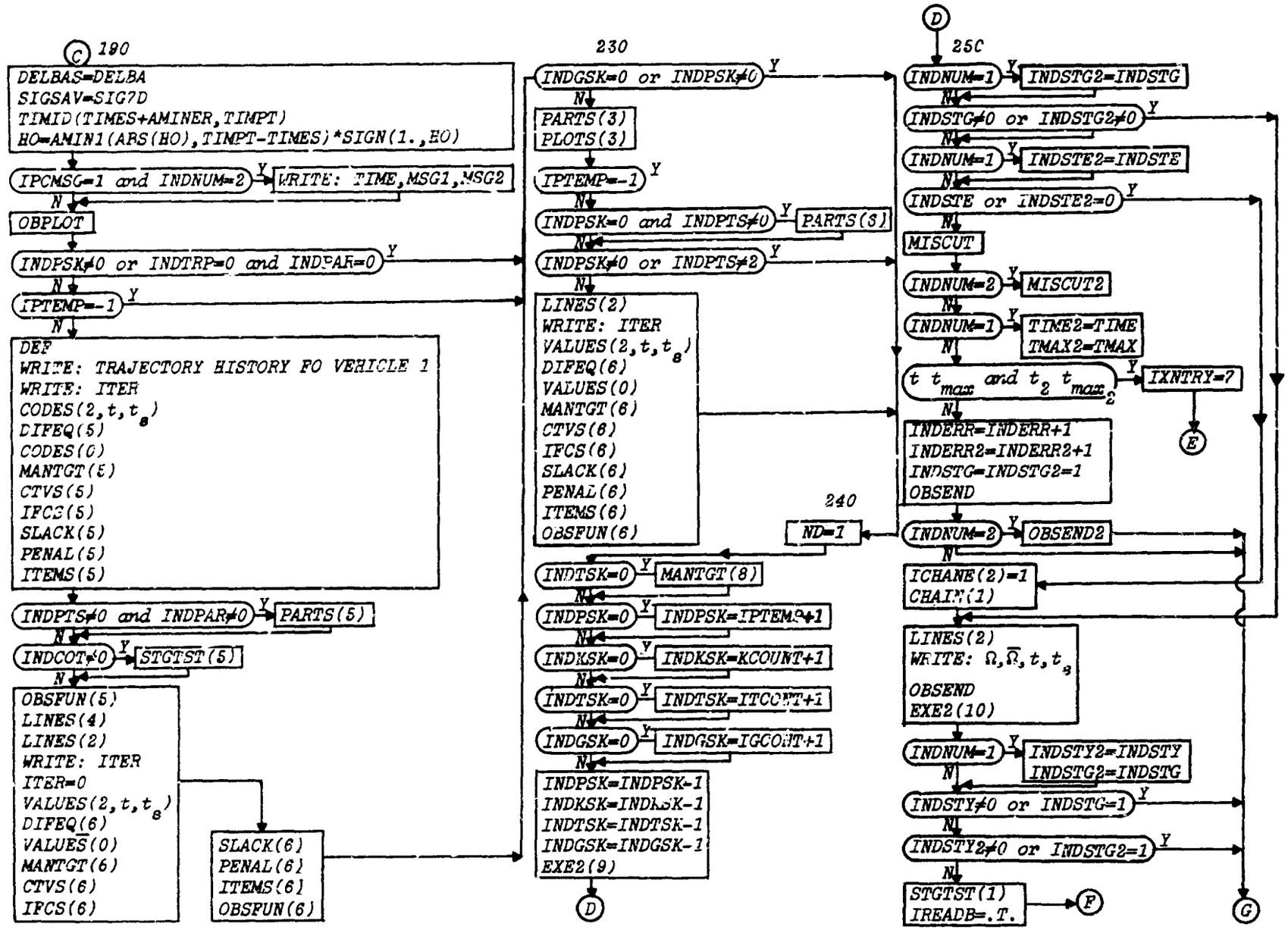
(10) The above procedure insures that all time points are hit, and that variable step integration does not interfere with the staging process.

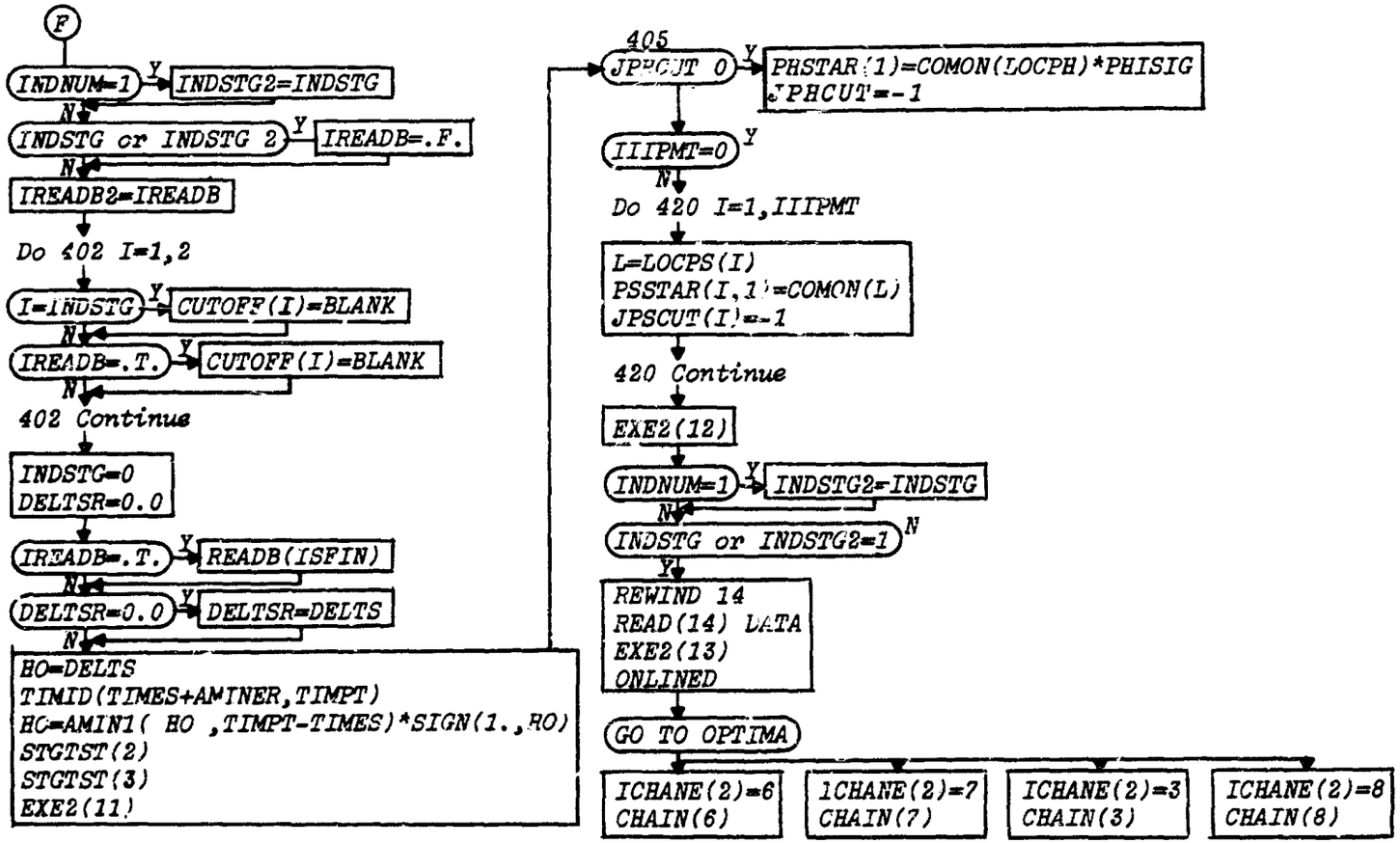
It should be noted that EXE drives the second vehicle trajectory through EXE2. This routine is virtually a copy of EXE. However, correct interfacing of the two vehicle trajectory integration procedures has required EXE2 to be split into a sequence of entry points. Each entry point carries out a well defined function for Vehicle 2's trajectory control. Control of integration stepsize for both trajectories is carried out by EXE itself.

A flow chart for program EXE is presented. Subroutine EXE2 follows the general computational flow of program EXE. However, the COMMON blocks, tapes, and subroutines employed are differentiated from those of EXE by the numeral 2. For example, COMMON/1/ becomes COMMON/12/; DIFEQ becomes DIFEQ2; and tape 12 becomes tape 22.

NOT REPRODUCIBLE




C
190
DELBAS=DELBA
SIGSAV=SIG7D
TIMID(TIMES+AMINER,TIMPT)
HO=AMIN1(ABS(HO),TIMPT-TIMES)*SIGN(1.,HO)
IPCMSG=1 and INDNUM=2
Y
WRITE: TIME,MSG1,MSG2
N
OBPLOT
INDPSK≠0 or INDTRP=0 and INDPAR=0
Y
N
IPTEMP=-1
Y
N
DEF
WRITE: TRAJECTORY HISTORY FO VEHICLE 1
WRITE: ITER
CODES(2,t,t_s)
DIFEQ(5)
CODES(0)
MANTGT(5)
CTVS(5)
IFCS(5)
SLACK(5)
PENAL(5)
ITEMS(5)
INDPTS≠0 and INDPAR≠0
Y
PARTS(5)
N
INDCOT≠0
Y
STGTST(5)
N
OBSFUN(5)
LINES(4)
LINES(2)
WRITE: ITER
ITER=0
VALUES(2,t,t_s)
DIFEQ(6)
VALUES(0)
MANTGT(6)
CTVS(6)
IFCS(6)
SLACK(6)
PENAL(6)
ITEMS(6)
OBSFUN(6)
230
INDGSK=0 or INDPSK≠0
Y
N
PARTS(3)
PLOTS(3)
IPTEMP=-1
Y
N
INDPSK=0 and INDPTS≠0
Y
PARTS(3)
N
INDPSK≠0 or INDPTS≠2
Y
N
LINES(2)
WRITE: ITER
VALUES(2,t,t_s)
DIFEQ(6)
VALUES(0)
MANTGT(6)
CTVS(6)
IFCS(6)
SLACK(6)
PENAL(6)
ITEMS(6)
OBSFUN(6)
240
ND=1
INDTSK=0
Y
MANTGT(8)
N
INDPSK=0
Y
INDPSK=IPTEMP+1
N
INDKSK=0
Y
INDKSK=KCOUNT+1
N
INDTSK=0
Y
INDTSK=ITCOUNT+1
N
INDGSK=0
Y
INDGSK=IGCOUNT+1
N
INDPSK=INDPSK-1
INDKSK=INDKSK-1
INDTSK=INDTSK-1
INDGSK=INDGSK-1
EXE2(9)
D
D
250
INDNUM=1
Y
INDSTG2=INDSTG
N
INDSTG≠0 or INDSTG2≠0
Y
N
INDNUM=1
Y
INDSTE2=INDSTE
N
INDSTE or INDSTE2=0
Y
N
MISCUT
INDNUM=2
Y
MISCUT2
N
INDNUM=1
Y
TIME2=TIME
TMAX2=TMAX
N
t t_max and t_2 t_max_2
Y
IXNTRY=7
N
INDERR=INDERR+1
INDERR2=INDERR2+1
INDSTG=INDSTG2=1
OBSEND
INDNUM=2
Y
OBSEND2
N
ICHANE(2)=1
CHAIN(1)
LINES(2)
WRITE: Ω,Ω̄,t,t_s
OBSEND
EXE2(10)
INDNUM=1
Y
INDSTY2=INDSTY
INDSTG2=INDSTG
N
INDSTY≠0 or INDSTG=1
Y
N
INDSTY2≠0 or INDSTG2=1
Y
N
STGTST(1)
IREADB=.T.
E
F
G


F
INDNUM=1
Y
INDSTG2=INDSTG
N
INDSTG or INDSTG 2
Y
IREADB=.F.
N
IREADB2=IREADB
Do 402 I=1,2
I=INDSTG
Y
CUTOFF(I)=BLANK
N
IREADB=.T.
Y
CUTOFF(I)=BLANK
N
402 Continue
INDSTG=0
DELTSR=0.0
IREADB=.T.
Y
READB(ISFIN)
N
DELTSR=0.0
Y
DELTSR=DELTS
N
HO=DELTS
TIMID(TIMES+AMINER,TIMPT)
HO=AMIN1(HO ,TIMPT-TIMES)*SIGN(1.,HO)
STGTST(2)
STGTST(3)
EXE2(11)
405
JPHCUT 0
Y
PHSTAR(1)=COMON(LOCPH)*PHISIG
JPHCUT=-1
IIIPMT=0
Y
N
Do 420 I=1,IIIPMT
L=LOCPS(I)
PSSTAR(I,1)=COMON(L)
JPSCUT(I)=-1
420 Continue
EXE2(12)
INDNUM=1
Y
INDSTG2=INDSTG
N
INDSTG or INDSTG2=1
N
Y
REWIND 14
READ(14) DATA
EXE2(13)
ONLINED
GO TO OPTIMA
ICHANE(2)=6
CHAIN(6)
ICHANE(2)=7
CHAIN(7)
ICHANE(2)=3
CHAIN(3)
ICHANE(2)=8
CHAIN(8)

1. DIFEQ-Differential Equation Selector

Purpose:

To enable the user to select which set of differential equations will be used by the program.

Usage:

CALL DIFEQ(IENTRY)

IENTRY = indicator which determines what task is to be performed by the differential equations subprogram which will be called by DIFEQ.

INDSEL is of the form i0j. j determines which differential equations subprogram will be used. INDSEL is nominally 101, so for this INDSEL, j = 1 and DIFEQ1 will be used.

Remarks:

DIFEQ is called by EXE, EXTRAN, and PTBEQN

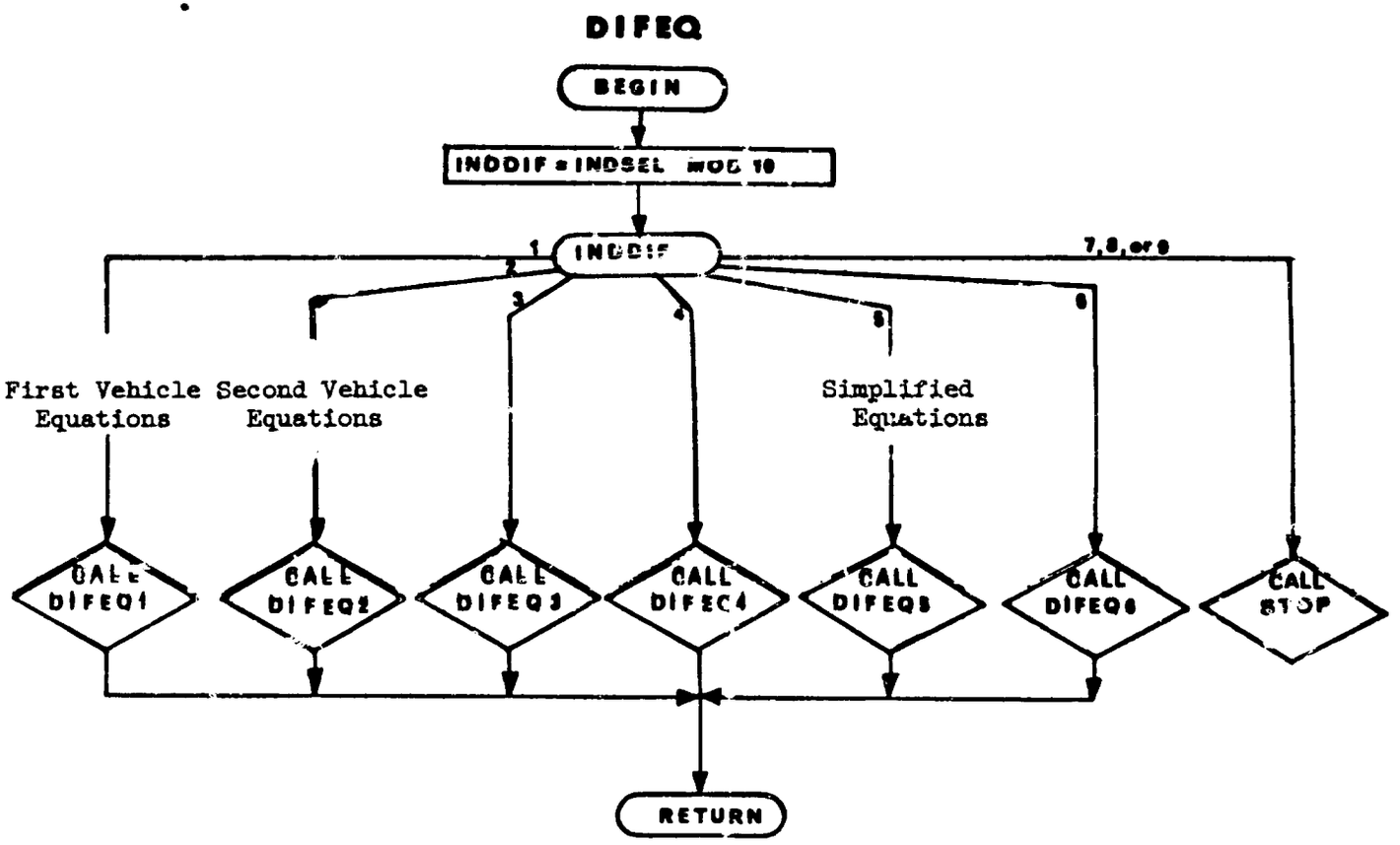
DIFEQ calls DIFEQ1, DIFEQ2, DIFEQ3, DIFEQ4, DIFEQ5, DIFEQ6, and STOP.

i determines which control system will be used. It should be noted that in the combat simulation, the first vehicle equations of motion are contained in DIFEQ1,and the second vehicle equations of motion are in DIFEQ2.

DIFEQ3, DIFEQ4, DIFEQ6

These are dummy routines included so that the user may write his own differential equations for use with the optimization program.

In writing a new differential equation subprogram, the entry point pattern established by EXE for such a subprogram must be adhered to. DIFEQ1 and DIFEQ5 are examples of such subprograms.


DIFEQ
BEGIN
INDDIF = INDSEL MOD 10
INDDIF
1
2
3
4
5
6
7,8,or 9
First Vehicle Equations
Second Vehicle Equations
Simplified Equations
CALL DIFEQ1
CALL DIFEQ2
CALL DIFEQ3
CALL DIFEQ4
CALL DIFEQ5
CALL DIFEQ6
CALL STOP
RETURN

2. MANTGT and MANTGT2 - Maneuvering Target

Purpose:

To prepare a target tape containing the time history of a "target trajectory."

To read the target tape as a function of time during a forward trajectory and to compute functions of the variables read from tape or in COMMON together with the variables of the current trajectory.

Method:

This subprogram has all the standard entry points of EXE, except that entry point 8 is unique, in that it prepares a target tape.

INDICP = 0: No computations made.

INDICP = 1: A target tape will be prepared; the target tape is TAPE 13. The information put on this tape is the values of the following variables:

TIME, XE77F, YE77F, ZE77F, UE77F, VE77F, WE77F, UE77F1, VE77F1, WE77F1.

At entry point 1 the tape is rewound and INDPAR is set to 0 (this deletes computation of partials). At entry point 8 the tape is written. None of the other entry points will be effective. EXE proper calls MANTGT (8) at every (ITCONT+1)th valid integration step. ITCONT is input. It is assumed that only a forward trajectory is to be run for a case with INDICP = 1 in the data. This trajectory should be set up to terminate on TMAX.

INDICP = 2: It is assumed that a target tape (TAPE 13) has been prepared by a previous case run with INDICP = 1 (or 3) in the data. During each forward trajectory of the present case, the target tape will be read so that the value of trajectory time of the present case is bracketed between the trajectory time of two adjacent records on the target tape. The instantaneous value of the variables on the target tape are estimated by linear interpolation. Then the necessary functions are computed from these interpolated values and the information available from the present trajectory. Code and value printing of the computed quantities are accomplished at entry points 5 and 6 respectively.

Remarks:

It must be known that the values of TIME on the target tape exceed the largest value of TIME that will be encountered on all trajectories of a case with INDICP = 2.

TIME should be a state variable when INDICP = 2.

INDICP = 3:

This option is designed for the purpose of running a standard case, without maneuvering target, for INDCYC cycles and then defining the last cycle as a target trajectory to be put on the target tape exactly as if INDICP were input as 1. On the last cycle, at MANTGT (1), TMAX is set to TOMAX and NCYCLE is set to 0 and INDICP is set to 1 so that the case will terminate just as if one trajectory with INDICP = 1 were run; however, this will not be done until INDPAR has been set ≠0 on the last cycle.

The following basic functions are computed at MANTGT (3): (see formulation manual for detail)

XAMTF	ΔX_E	$\vec{R}$	Coordinate differences of the two vehicles
YAMTF	ΔY_E		
ZAMTF	ΔZ_E		
XAMTF1	$\Delta \dot{X}_E$	$\vec{V}$	Velocity differences of the two vehicles
YAMTF1	$\Delta \dot{Y}_E$		
ZAMTF1	$\Delta \dot{Z}_E$		
VAMTF	$\lvert \vec{V} \rvert$		
RAMTF	$\lvert \vec{R} \rvert$		
RAMTF1 (= VCRAF)	$-\vec{V} \cdot \vec{R} / \lvert \vec{R} \rvert$		Rate of change of $\lvert R \rvert$
THASD	θ_{ASP}		Aspect angle
PHLED	ϕ_{LA}		Lead angle
SHEDD	σ_{T_A}		Heading of lead angle vector
SLOSD	σ_{LOS}		Heading of line-of-sight vector
SGERD	$\Delta\sigma_{LOS}$		Heading error from line-of-sight path
GGERD	$\Delta\gamma_{LOS}$		Inclination error from line-of-sight path

SMERD	$\Delta\sigma_{LA}$	Heading error from lead angle path
GMERD	$\Delta\gamma_{LA}$	Inclination error from lead angle path
TC		Time to kill

In addition, the following missile calculations are made:

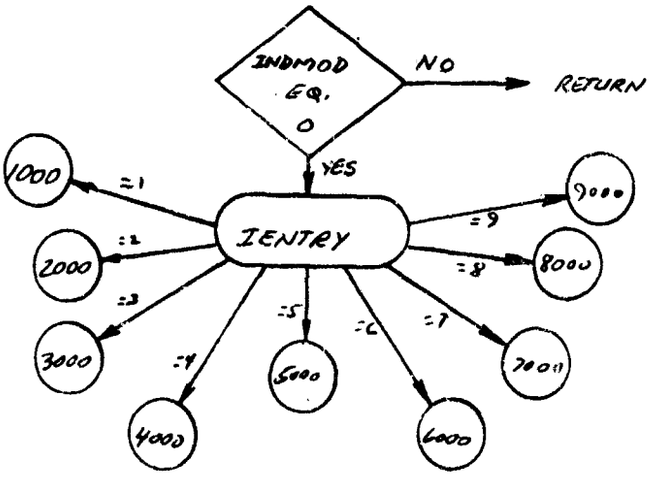
RMINF	R_{min}	Minimum missile range
RC77F	R_c	Range to kill zone
ESE7D	ϵ_{SE}	Allowable steering error

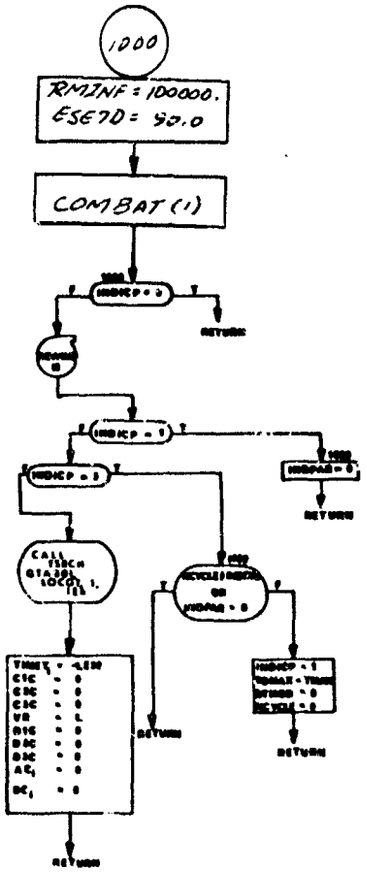
The input data for MANTGT calculations may consist of:

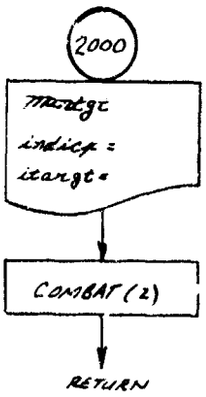
	Nominally	
INDICP	0	Option indicator
ITCONT	0	Skip indicator
AC	0	Arrays of missile range constants
BC	0	
C1C	0	Allowable steering error constants
C2C	0	
C3C	0	
D1C	0	Missile minimum range constants
D2C	0	
D3C	0	
RMAXF R_{MAX}	0	Missile maximum range constants
DLVMF ΔV_M	0	Missile ΔV
GTABO1 $f(\frac{\lvert\Delta R\rvert}{R_a})$		Two dimensional table for multiplier in steering error calculation.

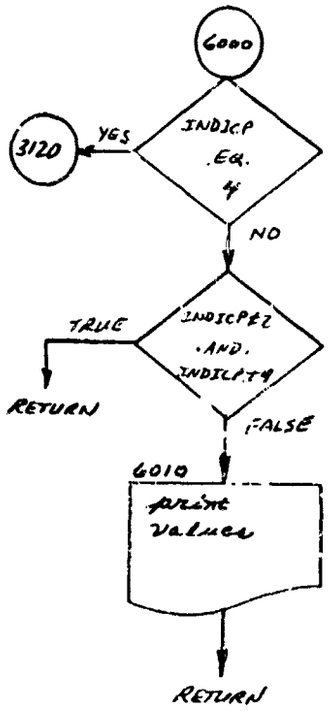
A flow chart for MANTGT is presented. MANTGT2 is identical except for use of vehicle 2 COMMON blocks, tapes, and auxiliary subroutines.

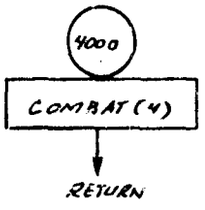
MANTGT

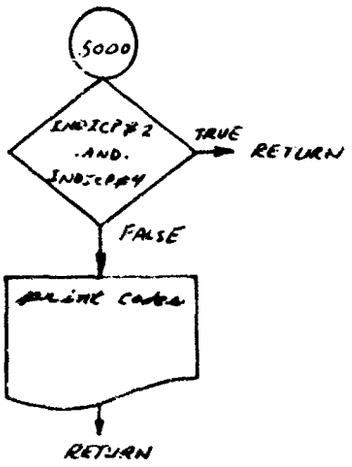

INDMOD .EQ. 0
NO
RETURN
YES
IENTRY
1000
=1
2000
=2
3000
=3
4000
=4
5000
=5
6000
=6
7000
=7
8000
=8
9000
=9

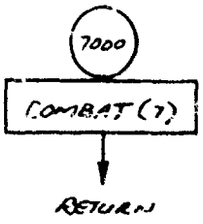

1000
RMINF = 100000.
ESETD = 90.0
COMBAT (1)
RETURN

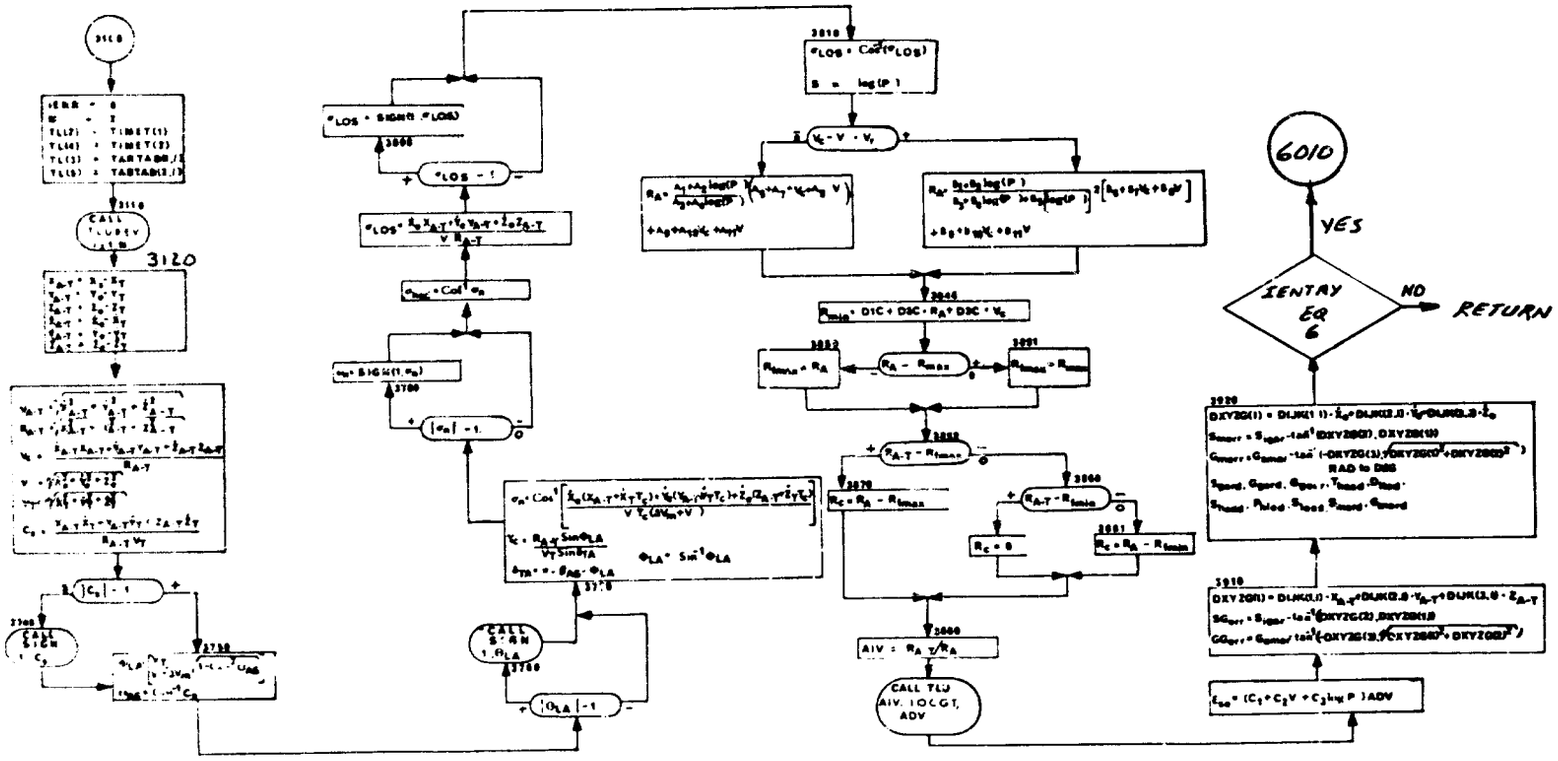

2000
indicp = itargt
COMBAT (2)
RETURN

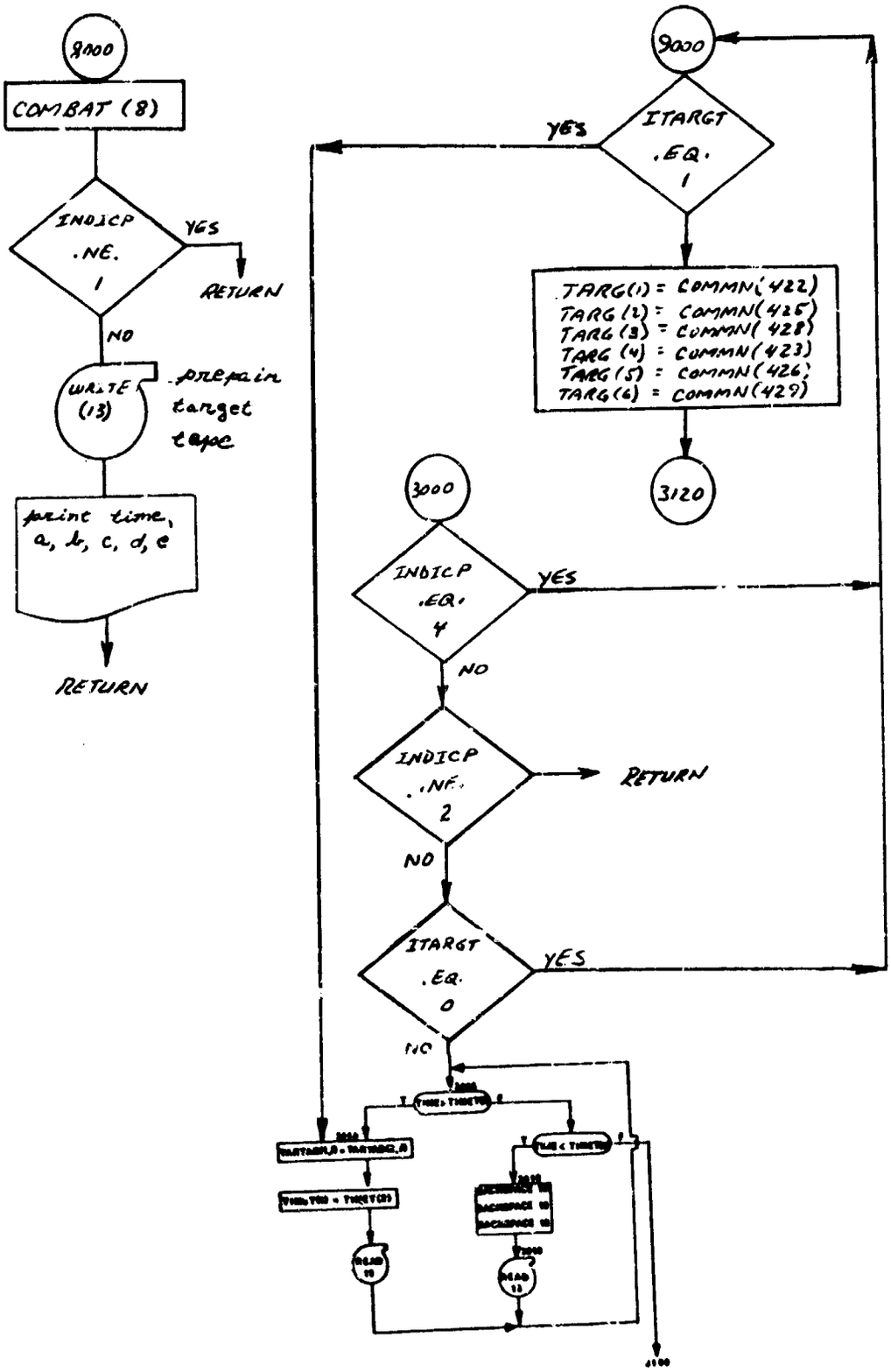

6000
INDICP .EQ. 4
YES
3120
NO
INDICP#2 .AND. INDICP#4
TRUE
RETURN
FALSE
6010
print values
RETURN


4000
COMBAT (4)
RETURN


5000
INDICP#2 .AND. INDICP#4
TRUE
RETURN
FALSE
print codes
RETURN


7000
COMBAT (7)
RETURN


3120
6010
YES
NO
RETURN
IENTRY EQ 6
CALL TLU AIV, IOCGT, ADV


9100
COMBAT (8)
INDICP .NE. 1
YES
RETURN
NO
WRITE (13)
prepair target tape
print time, a, b, c, d, e
RETURN
9000
ITARGT .EQ. 1
YES
TARG(1) = COMMN(422)
TARG(2) = COMMN(425)
TARG(3) = COMMN(428)
TARG(4) = COMMN(423)
TARG(5) = COMMN(426)
TARG(6) = COMMN(429)
3120
3000
INDICP .EQ. 4
YES
NO
INDICP .NE. 2
RETURN
NO
ITARGT .EQ. 0
YES
NO

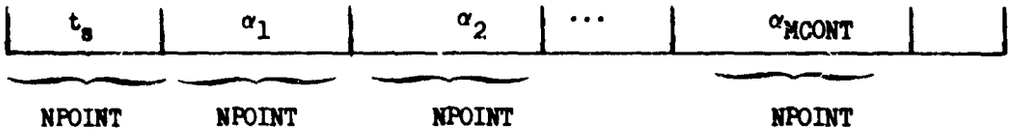
3. CTVS and CTVS2 - Control Variable Routine for EXE

Purpose:

During the forward trajectory, to obtain the instantaneous values of the control variables as a function of stage time; to obtain values of the initial conditions at the start of stage 1; to define points in stage time which must be hit in the corresponding stage of the reverse trajectory.

Control Variables

CTABLE is an array of NPOINT stage time values and the corresponding values of the MCONT control variables. The format of the CTABLE array is:



At CTVS(3), the task of this routine is to interpolate in the CTABLE to obtain the current values of the control variables. This is accomplished trivially by calling the interpolation routine TLU1. CTVS(3) does not allow values of the control variables that exceed the upper or lower boundaries (CTABUB, CTABLB) which may be input for the respective control variables.

CTABLE will be considered to contain stage time values unless INDCTA is 0. When a new CTABLE is prepared (in DALCAL) INDCTA is set to 1; hence on all trajectories after the nominal, CTABLE will contain stage time values. For the nominal trajectory CTABLE may be defined as a function of trajectory time if INDCTA is input 0 in stage 1 data. If the range of the CTABLE for a stage is exceeded, the pertinent end points will be used as the values of the control variables.

Tape Usage

In order that the CTABLE for a particular stage be available for CTVS(3), it must be read from tape before CTVS(3) is first called. This is the task of CTVS(2). For the nominal trajectory, the CTABLE exists in the data; hence after the stage data has been read the CTABLE for that stage may be assumed to be available for CTVS(3). On all trajectories after the nominal, the CTABLE for each major stage resides on tape IATAP. Furthermore, the stages exist in inverse order on IATAP; hence to get the proper stage from IATAP, CTVS(2) does a BACKSPACE, READ, BACKSPACE at the beginning of each major stage. It is assumed that IATAP has not been rewound since it was last written (in DALCAL). The information on IATAP is not for the exclusive use of CTVS; in fact, each logical record on IATAP contains much more information than just the CTABLE for that stage.

The complete format of each logical record of IATAP is described in the write-up of the routine (DALCAL) which prepares IATAP.

Initial Conditions

The values of the INDIOP initial conditions (which are not specified categorically in the stage 1 data) may be perturbed from trajectory to trajectory in a manner analagous to the way the control variable values in the CTABLE are perturbed. VALINS is the array of these initial conditions. VALINS is read from IATAP at the same time the CTABLE for stage 1 is read. Hence, at CTVS(2), after IATAP has been read for stage 1, the values in the VALINS array are given to the respective initial conditions. On the other hand, for the nominal, the VALINS array is initialized with the nominal values of the initial conditions specified in the stage 1 data. Just as with the control variables, the values of the initial conditions are not allowed to violate their respective upper or lower boundaries (VALIUB, VALILB).

Time Points

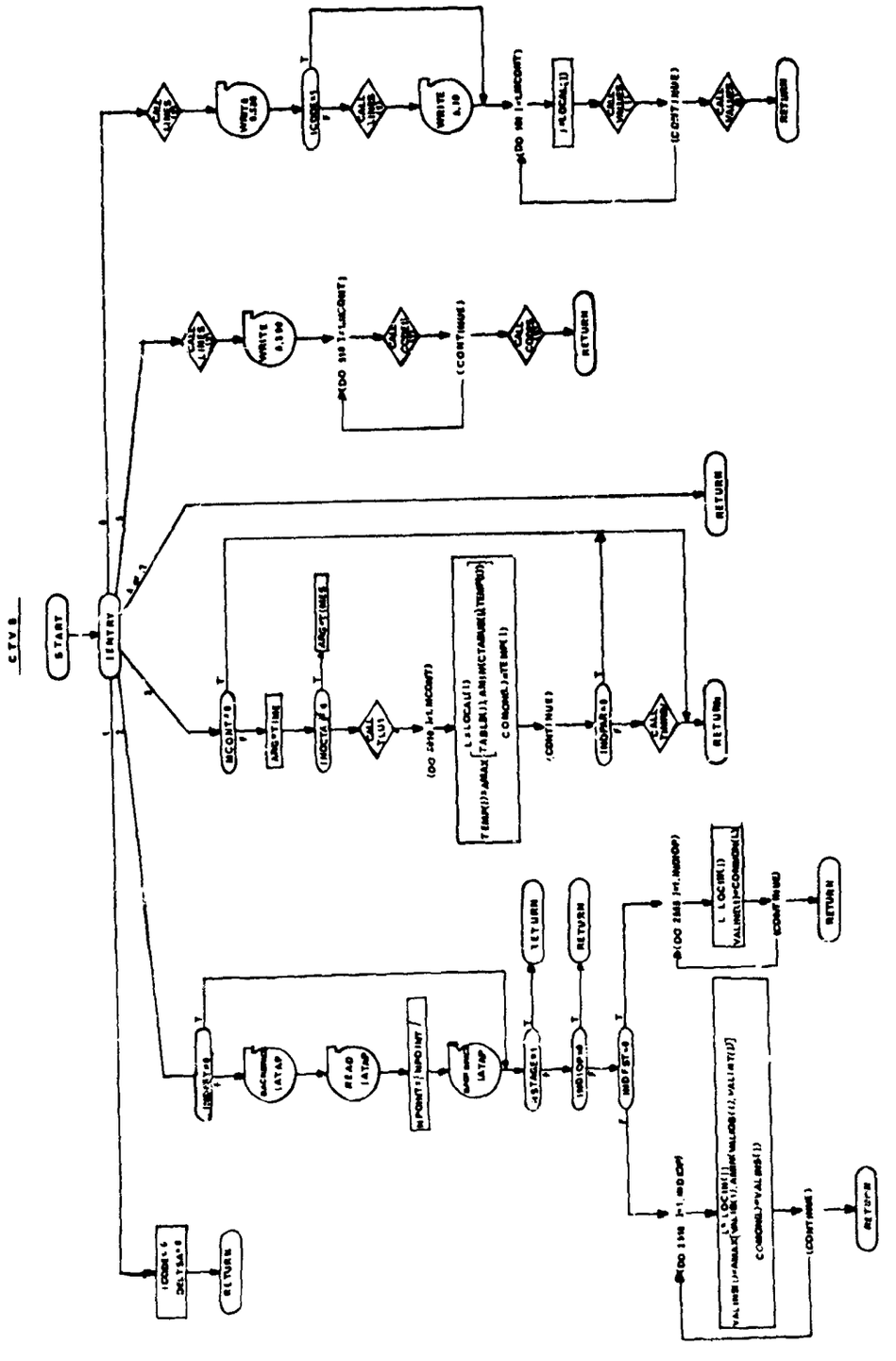
When partials are being computed (INDPAR $\neq$ 0), CTVS(3) makes a call to TIMREV in order to attempt to insert the current stage time value into the array of points which must be hit in the corresponding stage of the reverse trajectory.

However, the tolerance DELTSA is used in the call to TIMREV. This means the points stipulated by CTVS(3) may be no closer than DELTSA from any other point which has already been stipulated as one which the reverse trajectory must hit. The motivation for this feature is to enhance the performance of the variable step integration option by preserving the CTABLE at every point used to generate a trajectory (using DELTSA = 0). DELTSA may be input. It is nominally 1000.

CTVS has the standard entry points. Entry points 2 and 3 have been described in detail above. Code and value printing of the control variables is done at entry points 5 and 6 respectively.

Remarks:

CTVS2 obtains instantaneous values for the second vehicle control variables as a function of stage time, obtains initial second vehicle conditions at the start of stage 1, and defines points in time which must be put in the corresponding points in time of the reverse trajectory. CTVS2 is identical to CTVS except for the use of Vehicle 2's COMMON blocks, auxiliary routines, and tapes. A flow chart of CTVS is presented.


START
ENTRY
RETURN
RETURN
RETURN
RETURN
RETURN
RETURN
RETURN
RETURN

4. IFCS-In-flight Constraints (Boundary Violation Routine)

Purpose:

To calculate the instantaneous boundary violations of given functions and to integrate each of these violations.

IFCS is a subprogram of EXE; it has all the standard entry points. Up to six functions may be specified for IFCS; this is done in the data. If no functions are specified then IFCS is not active.

Method:

The instantaneous violation of the ith IFCS function is computed at entry point 3 as

$$\dot{A}_{IFCS_i} = \left\{ \max \left[0, (y_i - P_i (\xi_i)) \right] \right\}^{INDBND_i} \text{ if } INDBND_i > 0$$

$$= \left\{ \min \left[0, (P_i (\xi_i) - y_i) \right] \right\}^{|INDBND_i|} \text{ if } INDBND_i < 0$$

where y_i is the value of the i'th variable listed in the BFCDV array

ξ_i is the value of the i'th variable listed in the BFCDV array

P_i is the function tabulated as a two dimensional table on the i'th PSTAR data card (s).

$INDBND_i$ is an indicator which is input > 0 to define an upper boundary; and which is input < 0 to define a lower boundary.

At entry point 7, the violation $\dot{A}_{IFCS_i}$ is integrated.

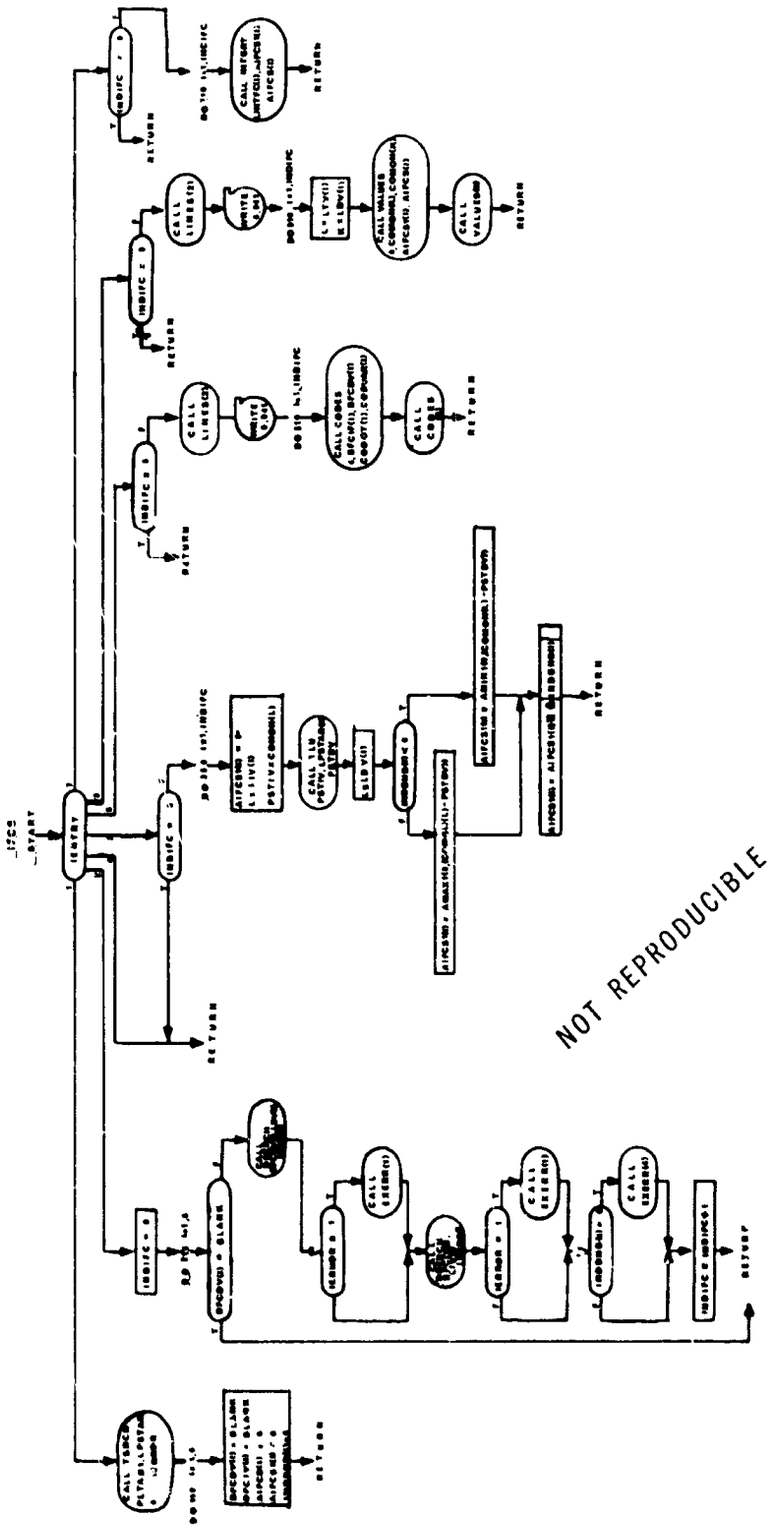
The names ascribed to the respective violations and their integrals are, in order:

Violations	Integrated Violation
AIFCS1	AIFCS
BIFCS1	BIFCS
CIFCS1	CIFCS
DIFCS1	DIFCS
EIFCS1	EIFCS
FIFCS1	FIFCS

The first IFCS function for which no computation is requested terminates the list of those IFCS functions for which computations will be made.

Remarks:

IFCS2 calculates instantaneous boundary violations of given functions and their integrals for vehicle 2. A program modification is required to permit specification of second vehicle in-flight constraints in cooperative variational optimization. IFCS2 is identical to IFCS except for the use of the second vehicle's COMMON blocks and auxiliary routines. A flow chart for IFCS is presented.



NOT REPRODUCIBLE

5. PARTS-Partial Derivatives

Purpose:

To obtain all necessary partial derivatives and put them on the partial tape (TAPE12); also, to transmit information from the forward trajectory to the reverse trajectory by putting it on the partial tape.

Methods:

This subprogram has the standard entry points of all the other subprograms of EXE. The calculation of a particular matrix of partials is accomplished by a call to PSUBR; the particular matrix desired is completely specified in the arguments to PSUBR.

The partials to be computed are:

RMATX	$= R = \left[\frac{\partial x_i^o}{\partial \xi_j}\right]$	initial conditions	PARTS (2) beginning of a stage	
PMATX	$= P = \left[\frac{\partial x_i^+}{\partial x_j^-}\right]$	h-transformation		
FMATX	$= F = \left[\frac{\partial \dot{x}_i}{\partial x_i}\right]$		PARTS (3) during a stage	
GMATX	$= G = \left[\frac{\partial \dot{x}_i}{\partial \alpha_j}\right]$			
PARTPH	$= \left\{\frac{\partial \Phi}{\partial x_i}\right\}$	terminal partials	PARTS (3) at the end of every major stage	
PARTPS	$= \left[\frac{\partial \Psi_k}{\partial x_i}\right]$			
PARTOM	$= \left\{\frac{\partial \Omega}{\partial x_i}\right\}$			
PHI771	$= \dot{\Phi}$			
OMEGA1	$= \dot{\Omega}$			
PSI771	$= \left\{\dot{\Psi}_k\right\}$			

$\dot{\Phi}$, $\dot{\Omega}$, and $\left\{\dot{\Psi}_k\right\}$ are computed by the chain rule:

e.g., $$\dot{\Phi} = \left[\frac{\partial \Phi}{\partial x_i}\right]\left\{\dot{x}_i\right\} + \frac{\partial \Phi}{\partial t_s}$$

this is valid since $\Phi = \Phi(x, t_s) + C$

It may be necessary to modify the terminal partials at the end of a stage. If the contribution to the payoff at the end of a given stage is 0, then $\left\{\frac{\partial \Phi}{\partial x_i}\right\}$ and $\dot{\Phi}$ are set to 0. Likewise if the contribution to the ith constraint at the end of stage i is 0, then $\left\{\frac{\partial \Psi_i}{\partial x_j}\right\}$ and $\dot{\Psi}_i$ are set to 0.

These conditions are determined by the values of JPHCUT and JPSCUT respectively. For the payoff, the contribution is 0, if JPHCUT $\leq$ 0. For the kth constraint the contribution is 0, if $JPSCUT_k \leq 0$. the analyst has control of JPHCUT and $JPSCUT_k$ in the input data.

At PARTS (3) a call to TIMREV is made (with 0 tolerance) to define the current stage time as a point which must be hit in the corresponding stage of the reverse trajectory.

At PARTS (2), the R matrix is calculated at the first stage only (and then only if it is needed). At the beginning of any stage after the first, the P matrix is calculated but only if it is actually needed. The R matrix and the P matrix are put on the partial tape directly within PARTS. The F matrix, the G matrix, all terminal partials and other information for reverse is put on the partial tape via the ad-hoc blocking routine PRPACK.

At PARTS (4), the R matrix or the P matrix is printed (if it has been computed).

At PARTS (5), code printing is done for the F and G matrices.

At PARTS (6), the values of the F and G matrices are printed; at the termination of a stage the terminal partials are printed also.

The information (besides the partials) which is transmitted to reverse is:

TIMES	Stage time
ND	Indicator determining beginning and type of a major stage.
ALPHC	Current values of control variables.
DELTAR	Integration step size for reverse.

INDWMA	Weighting matrix option for reverse.
WAI	Weighting matrix parameters.
WBI	Weighting matrix parameters.

This information is put on the partial tape every time an F and a G matrix are put on the tape.

Saving α - Points

There are several ways of preserving the α history which generates a given trajectory. (there will be no reason to preserve the α history on a trial trajectory). The success of the program will depend to some extent on the accuracy in preserving the α-history of a given cycle.

The (TIMES) history exists in tabular form (CTABLE) as a function of stage time for each major stage. During the numerical integration of the forward trajectory the CTABLE is sampled (by some given interpolation formula) at the various points in stage time dictated by the particular numerical integration method. It is only this sampled version of the α-history which determines the trajectory. Hence to preserve the α-history which generated the trajectory it is only necessary to preserve the corresponding sample of the CTABLE that was used.

There are basically three options available for preserving the α-history.

Option I:

> The α-history is preserved at those points in the forward trajectory where partials are computed, and only at those points.

Option II:

> The α-history is preserved at those points in the forward trajectory where partials are computed (as in Option I), and in addition a subset of the sample points used to effect the numerical integration; the subset is determined as follows:
>
> > An α point is preserved if it is no closer than a distance of DELTSA away from the previous α point which was preserved. (distance means distance in stage time). The α time points are preserved in strictly monotonically increasing order; if for any reason the integration is backed up, all α-time points that might have been accumulated in the back-up interval are annihilated.

Option III:

> Use Option I on the nominal (i.e. first cycle of a run) and Option II on all others.

Every α-time point which is preserved in the forward trajectory will be hit in the reverse trajectory and a corresponding $\delta\alpha$ will be produced at this point.

Observations and Remarks

(1) Option I accomplishes precisely what the program has always done.

(2) Option I is a special case of Option II (with DELTSA = ∞). In fact this is the way Option I is selected.

(3) Option II still can fail to preserve all the α-history that was used to generate a trajectory even when DELTSA = 0. This may happen because the integration step size control logic has a part in determining the trajectory; if we cannot reproduce the exact step size control logic on the next cycle with a $\delta\alpha$ history identically 0, then we cannot reproduce the same trajectory. Now a trial integration step which was rejected on the nominal does not have an α-time point associated with it. Hence this α point was not preserved; hence the next cycle might not obtain the same α value for this point and hence the trial integration step which was rejected on the nominal may be accepted on the next cycle.

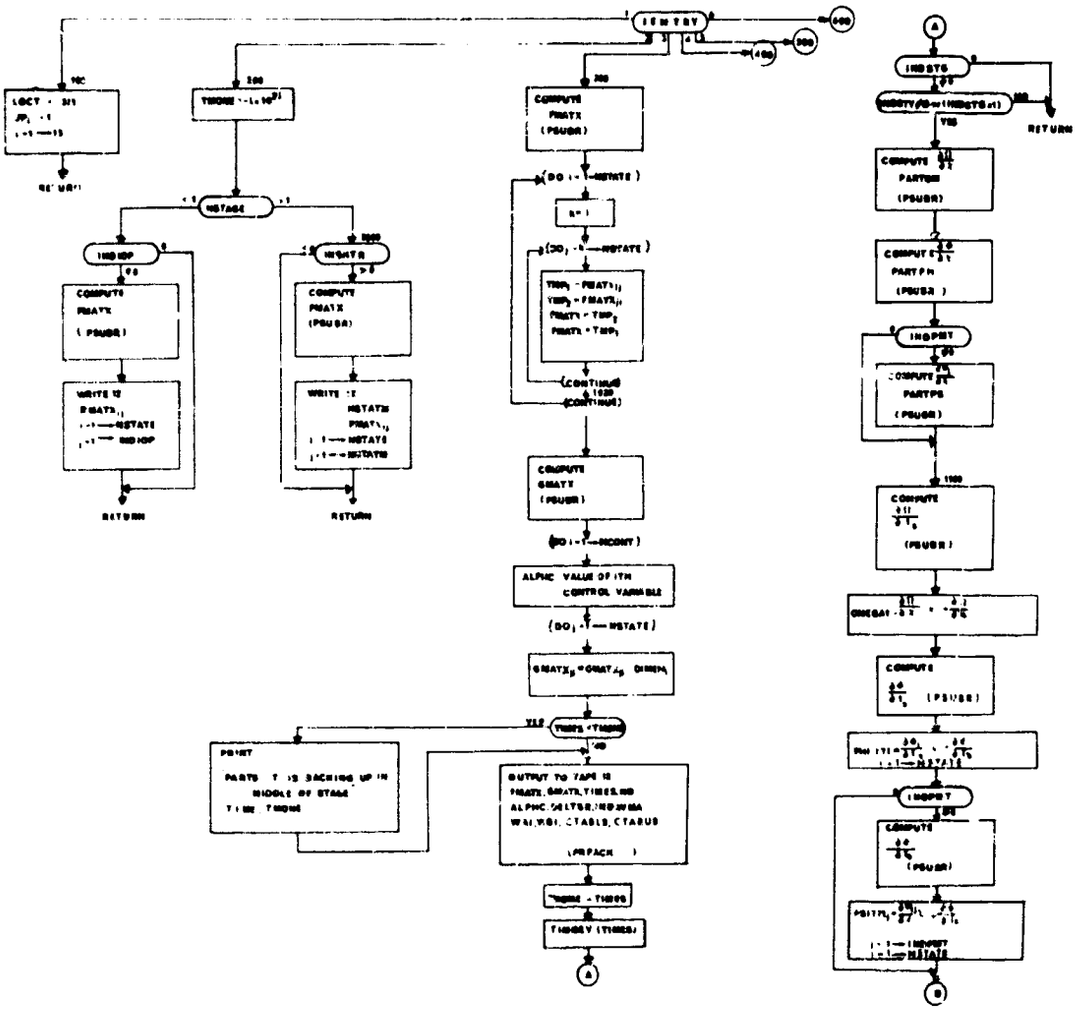
(4) A remedy to the problem in (3) would be to have an Option that would preserve every α-time point even if it were associated with an integration step that was rejected. However, it is questionable whether the problem is of that much importance.

(5) When utilizing vehicle 2 functions in a cooperative variational optimization problem, care must be taken to insure that

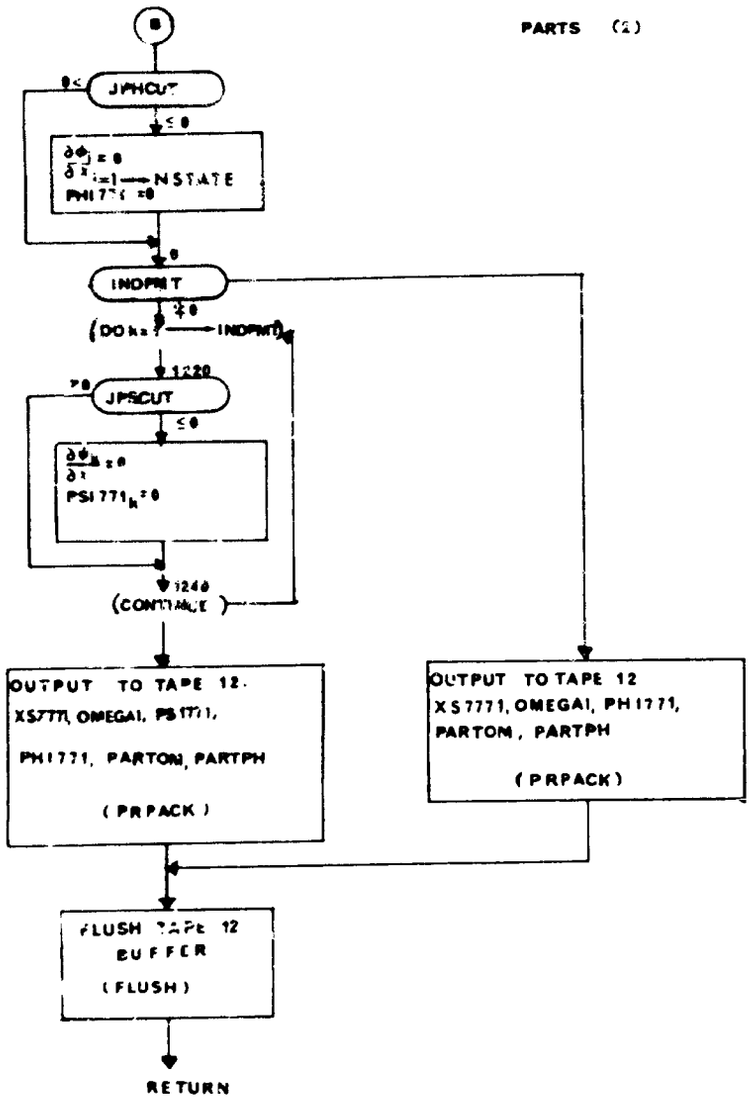
   (a) The second vehicle function has a unique name assigned in the vehicle 1 directory.

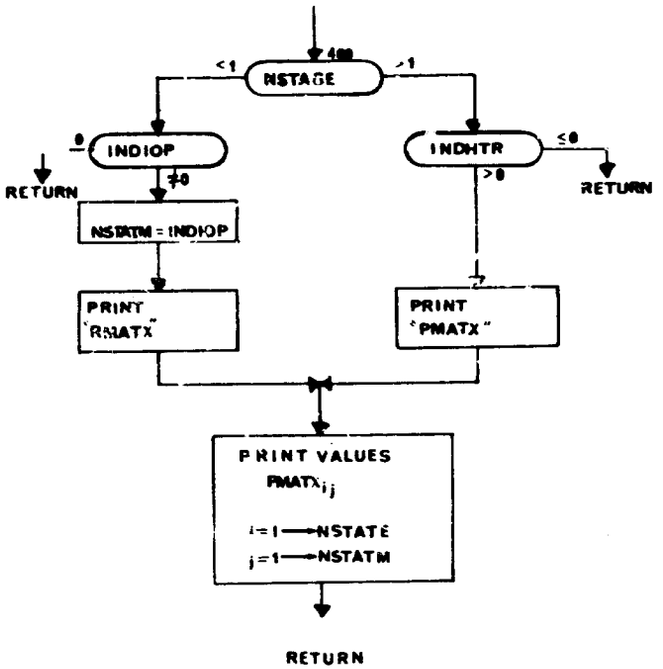
   (b) Appropriate transformations are available in ONETWO and TWOONE.

PARTS (1)

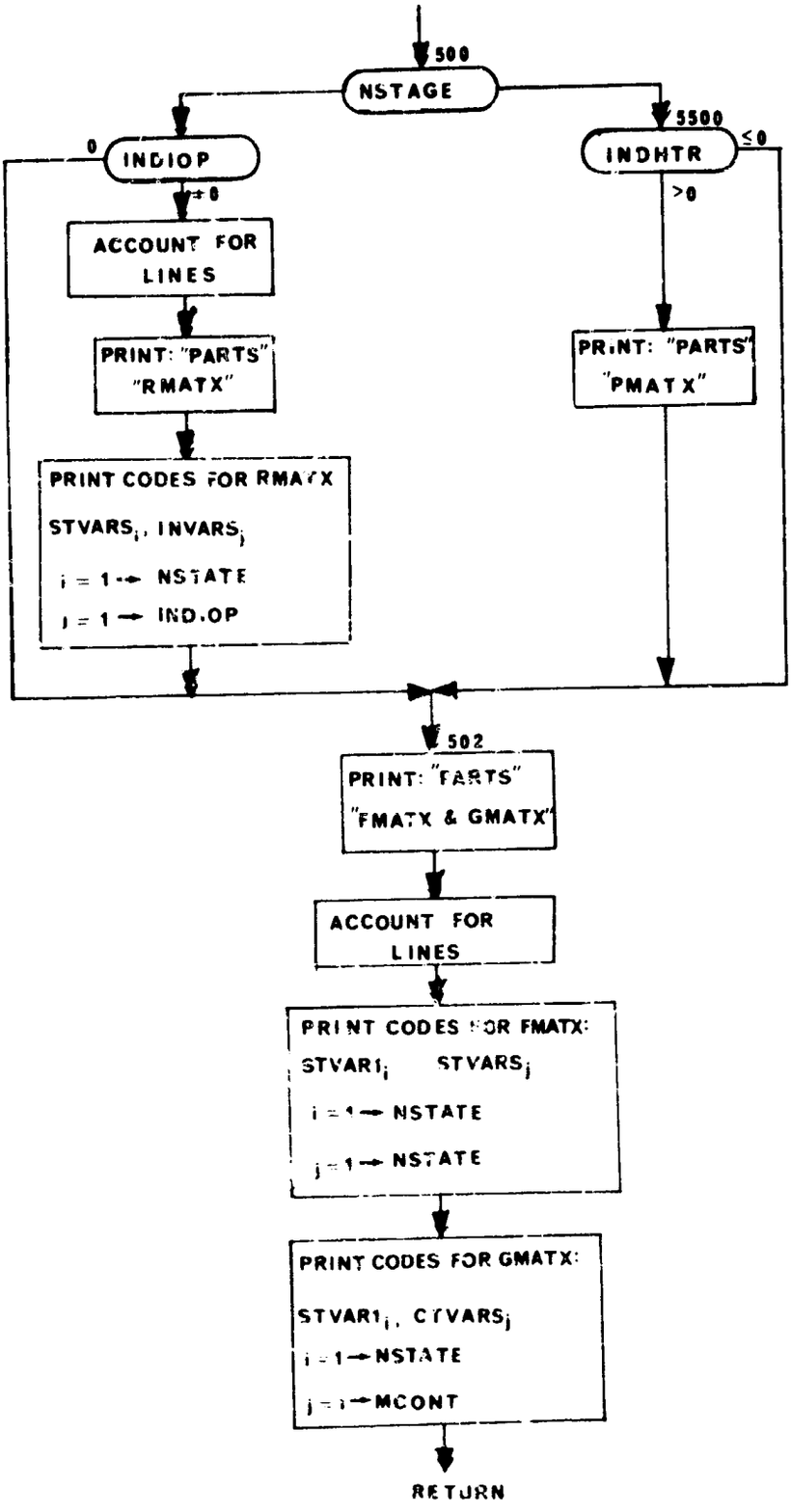


PARTS (2)

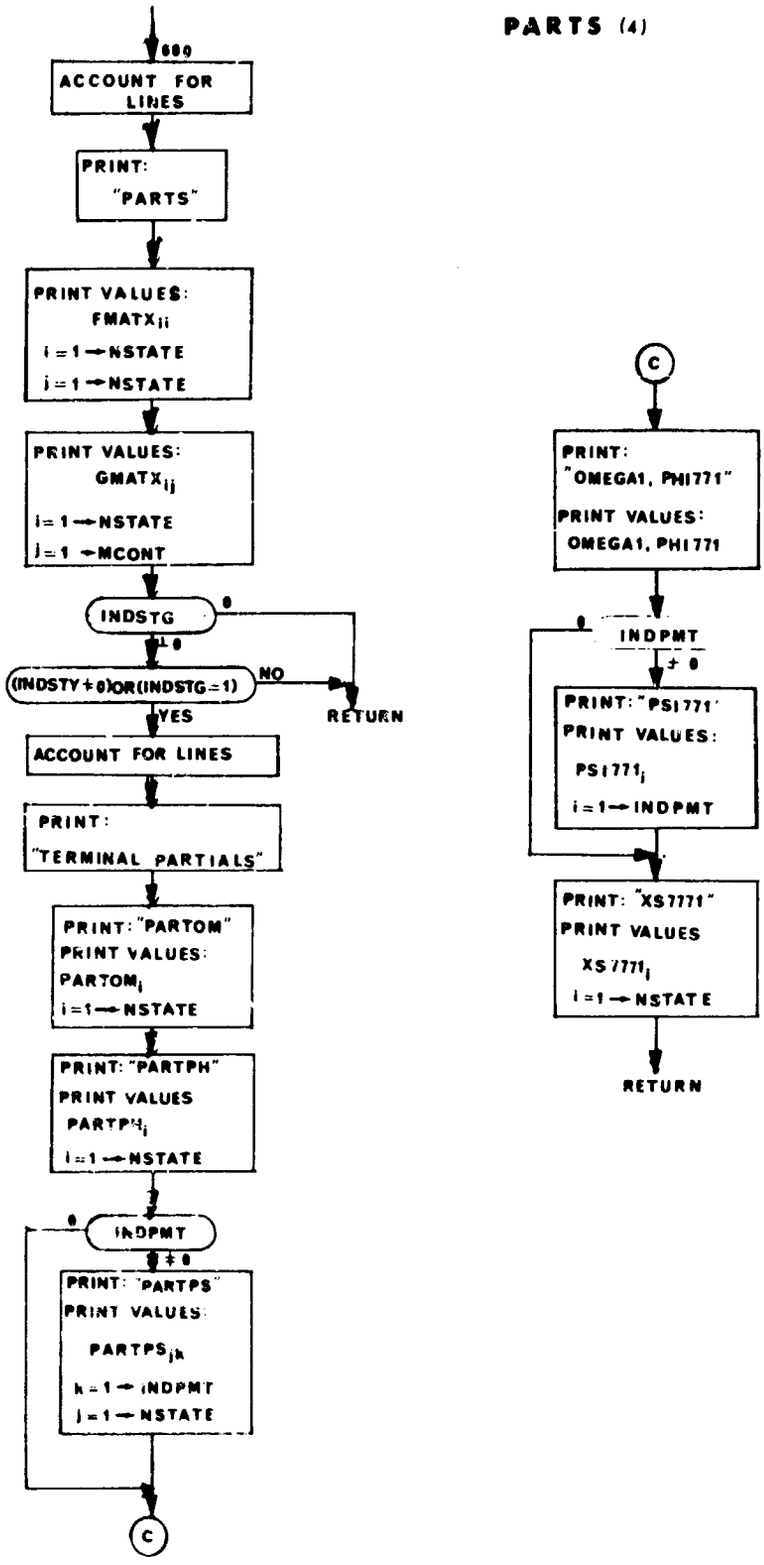

JPHCUT
INDPRT
JPSCUT
OUTPUT TO TAPE 12
XS7771, OMEGAI, PS1771,
PHI771, PARTON, PARTPH
(PRPACK)
OUTPUT TO TAPE 12
XS7771, OMEGAI, PHI771,
PARTON, PARTPH
(PRPACK)
FLUSH TAPE 12 BUFFER
(FLUSH)
RETURN


NSTAGE
INDIOP
INDHTR
RETURN
RETURN
NSTATM = INDIOP
PRINT "RMATX"
PRINT "PMATX"
PRINT VALUES
PMATX
i=1 → NSTATE
j=1 → NSTATM
RETURN

PARTS (3)


500
NSTAGE
0
INDIOP
≠0
ACCOUNT FOR LINES
PRINT: "PARTS" "RMATX"
PRINT CODES FOR RMATX
STVARS$_i$, INVARS$_j$
i = 1 → NSTATE
j = 1 → INDIOP
5500
INDHTR
≤0
>0
PRINT: "PARTS" "PMATX"
502
PRINT: "PARTS" "FMATX & GMATX"
ACCOUNT FOR LINES
PRINT CODES FOR FMATX:
STVAR1$_i$ STVARS$_j$
i = 1 → NSTATE
j = 1 → NSTATE
PRINT CODES FOR GMATX:
STVAR1$_i$, CTVARS$_j$
i = 1 → NSTATE
j = 1 → MCONT
RETURN

PARTS (4)


600
ACCOUNT FOR LINES
PRINT: "PARTS"
PRINT VALUES: FMATX$_{ij}$ i=1→NSTATE j=1→NSTATE
PRINT VALUES: GMATX$_{ij}$ i=1→NSTATE j=1→MCONT
INDSTG
0
≠0
(INDSTY≠0)OR(INDSTG=1)
NO
YES
RETURN
ACCOUNT FOR LINES
PRINT: "TERMINAL PARTIALS"
PRINT: "PARTOM" PRINT VALUES: PARTOM$_i$ i=1→NSTATE
PRINT: "PARTPH" PRINT VALUES PARTPH$_i$ i=1→NSTATE
INDPMT
0
≠0
PRINT: "PARTPS" PRINT VALUES: PARTPS$_{jk}$ k=1→INDPMT j=1→NSTATE
C
C
PRINT: "OMEGA1, PHI771" PRINT VALUES: OMEGA1, PHI771
INDPMT
0
≠0
PRINT: "PSI771" PRINT VALUES: PSI771$_i$ i=1→INDPMT
PRINT: "XS7771" PRINT VALUES XS7771$_i$ i=1→NSTATE
RETURN

6. LINCOM and LINCOM2 - Linear Combination Routines

Purpose:

To compute new functions as linear combinations of existing functions.

i.e. $\xi^* = \sum_{i=1}^{n} c_i \xi_i$

where the ξ_i are existing functions, the c_i are fixed coefficients, and ξ^* is the new function.

Method:

The c_i and the ξ_i for a particular ξ^* are specified in the data -- the c_i are input directly; the ξ_i are input as BCD names, and hence the names for the ξ_i must be in the directory.

Up to five different linear combination functions, ξ^*, may be defined this way. For each ξ^*, n must be ≤ 3. If a linear combination is desired with more than three component functions ξ_i, then it must be defined as a linear combination of already computed linear combinations (as will be illustrated in the examples below).

The names of the five available linear combination functions are ALINK, BLINK, CLINK, DLINK, and ELINK. The names of the cost vector for the above linear combination functions are, respectively, ACOST, BCOST, CCOST, DCOST, ECOST. ALINK through ELINK are computed in alphabetical order.

Example 1:

To define a linear combination $\xi^* = (1)h + (2)V_g$, the following input data is sufficient:

```
BLINC BCD 2HGC7F⁰VG77F

BCOST     1., 2.
```

Example 2:

To define a linear combination $\xi^* + (5.2)M_N + (6.5)h + (2.0)AIFCS + (10.)V_g$ the following input data in sufficient:

```
ALINC BCD 3AMACH⁰HGC7F⁰AIFCS

ACOST     5.2, 6.5, 2.0

BLINC BCD 2ALINK⁰VG77I

BCOST     1., 10.
```

Example 3:

To define a linear combination $\xi^* = (1/2)m^1 + (1/2)m^2 + h^2$ where m^1 denotes the mass in the first stage, m^2 denotes the mass in the second stage and h^2 the altitude in the second stage, the following data is sufficient:

```
          ALINC BCD 1AMASS
Stage 1
          ACOST     .5

          ALINC BCD 1b
Stage 2   BLINC BCD 3ALINKbAMASSbHGC7F
          BCOST     1.,.5,1.
```

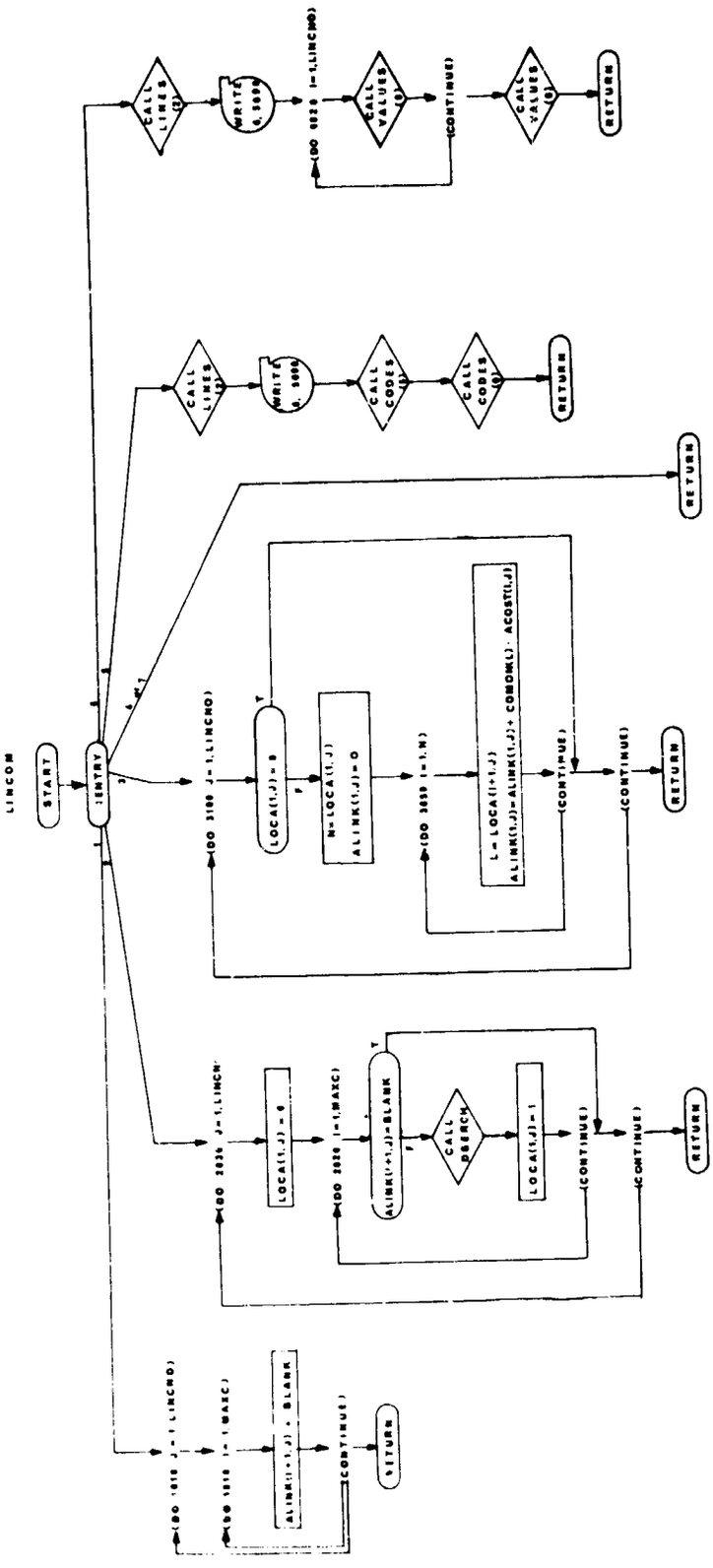
Hence, at the end of the last stage, BLINC will be the average of the mass at the termination of the first stage and the last stage plus the altitude at the end of the last stage. Note that when a linear combination function is "blanked out" in the data, the value of that function remains constant.

This subprogram has all the standard entry points. However, entry points 4, 5, 6, and 7 are vacuous. That is, no printing of the values of the linear combinations is done; also, nothing is integrated.

It is well to note that it is possible to take linear combinations involving in-flight constraints and also to take in-flight constraints of linear combinations. The reason is that LINCOM(3) is called before IFCS(3) from EXE.

Remarks:

LINCOM2 computes linear combination functions for the second vehicle. The use of second vehicle linear combination functions in cooperative variational optimization problems will require a program modification. LINCOM2 is identical to LINCOM except for the use of the second vehicle's COMMON blocks and auxiliary subroutines. A flow chart for LINCOM is presented.


LINCOM
START
ENTRY
CALL LINES
WRITE
CALL VALUES
CALL CODES
RETURN
CONTINUE
CALL DSERCH
LOCA(1,J) = 0
LOCA(1,J) = 1
N = LOCA(1,J)
ALINK(1,J) = 0
L = LOCA(I+1,J)
ALINK(1,J) = ALINK(1,J) + COMON(L) · ACOST(1,J)
ALINK(I+1,J) = BLANK

7. SLACK and SLACK2 - Slack Variable Routines

Purpose:

To make available "slack" variables that may be used for various effects including optimal staging and optimizing or constraining the initial value of a variable.

Method:

The four slack variables, FLUXA, FLUXB, FLUXC, FLUXD, and their respective derivatives, FLUXA1, FLUXB1, FLUXC1, and FLUXD1, are available. SLACK has all the standard entry points, however, some of the entry points are vacuous.

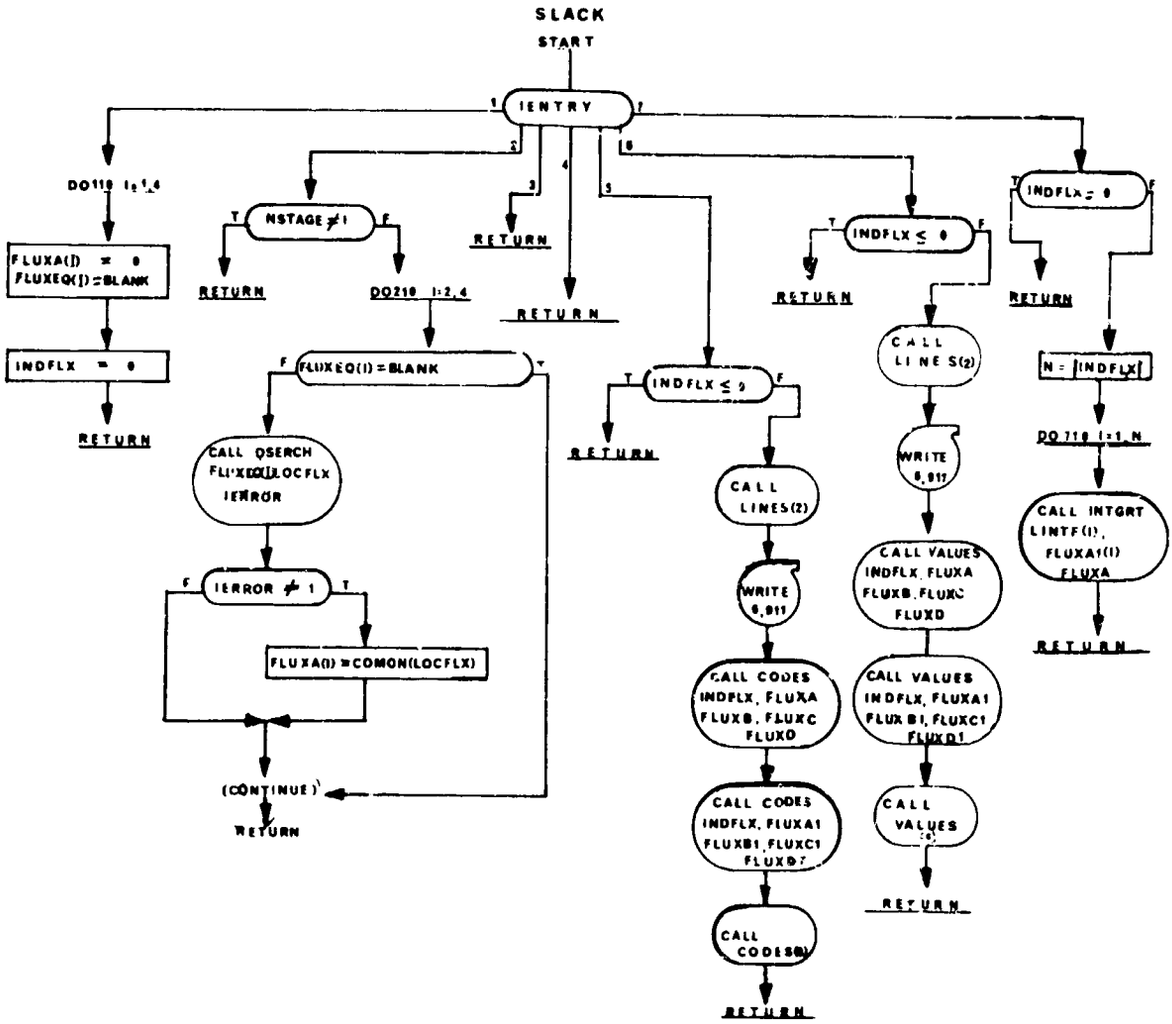
The first |INDFLX| slack variables are integrated; INDFLX is input. If INDFLX is 0, no slack variables are integrated. FLUXEQ is an array of BCD names; the BCD name in $FLUXEQ_i$ equates the ith slack variable with the initial value of that variable. (i.e., the value at the beginning of stage 1).

Code and value printing of the first INDFLX slack variables and their derivatives is done if and only if INDFLX > 0.

All variables are nominally 0; FLUXEQ is nominally blank.

Remarks:

SLACK2 defines slack variables for the second vehicle. Before use of the second vehicle's slack variables in a cooperative variational optimization problem, a program modification equating the second vehicle's slack variables to names in the first vehicle's directory is required. SLACK2 is identical to SLACK except for the use of the second vehicle's COMMON blocks and auxiliary subroutines. A flow chart for SLACK is presented.


SLACK
START
IENTRY
DO110 I=1,4
FLUXA(I) = 0
FLUXEQ(I)=BLANK
INDFLX = 0
RETURN
NSTAGE≠1
RETURN
DO210 I=2,4
FLUXEQ(I)=BLANK
CALL QSERCH
FLUXEQ(I),LOCFLX
IERROR
IERROR ≠ 1
FLUXA(I)=COMON(LOCFLX)
(CONTINUE)
RETURN
RETURN
INDFLX ≤ 0
RETURN
CALL
LINES(2)
WRITE
6,911
CALL CODES
INDFLX, FLUXA
FLUXB, FLUXC
FLUXD
CALL CODES
INDFLX, FLUXA1
FLUXB1, FLUXC1
FLUXD1
CALL
CODES(8)
RETURN
INDFLX ≤ 0
RETURN
CALL
LINES(2)
WRITE
6,911
CALL VALUES
INDFLX, FLUXA
FLUXB, FLUXC
FLUXD
CALL VALUES
INDFLX, FLUXA1
FLUXB1, FLUXC1
FLUXD1
CALL
VALUES
(8)
RETURN
INDFLX ≠ 0
RETURN
N = INDFLX
DO 710 I=1,N
CALL INTGRT
LINTF(I),
FLUXA1(I)
FLUXA
RETURN

8. PENAL and PENAL2 - Penalty Function Routine

Purpose:

To compute a penalty function using the values of the variables, Ψ_i, whose names are listed in the ENDCON array and/or values Ψ_i^* that have been accumulated at previous stage points for these variables in the PSSTAR table.

Method:

PENAL is a standard subprogram of EXE. PENAL will be effective if and only if MAXIM or MINIM contains the name "PNALTY". PNALTY is the name of the function that will be computed at entry point 3:

$$PF = W_1 \hat{\Psi}_1 \mp \sum_{i=2}^{IIIPMT} W_i (\hat{\Psi}_i - \bar{\Psi}_i)^2$$

where W_i are weights (WGTSI) and

$$\hat{\Psi}_i - \bar{\Psi}_i = \Psi_i - \bar{\Psi}_i \quad \text{if } JPSCUT_i = 1$$

$$= \Psi_i^* - \bar{\Psi}_i \quad \text{if } JPSCUT_i < 0$$

$$= 0 \quad \text{if } JPSCUT_i = 0$$

The - sign is used if the name "PNALTY" is in MAXIM. The + sign is used if the name "PNALTY" is in MINIM.

The weights W_i may be input or may be controlled by some logic in the control system. $JPSCUT_i$ is the array of indicators that indicate the stage point to which the corresponding Ψ_i function applies. Ψ_i^* is an element of the PSSTAR table and is the value of Ψ_i at the most recent stage point for which $JPSCUT_i = 1$.

Code and value printing are done at entry points 5 and 6 respectively. The value of PNALTY will be printed.

Remarks:

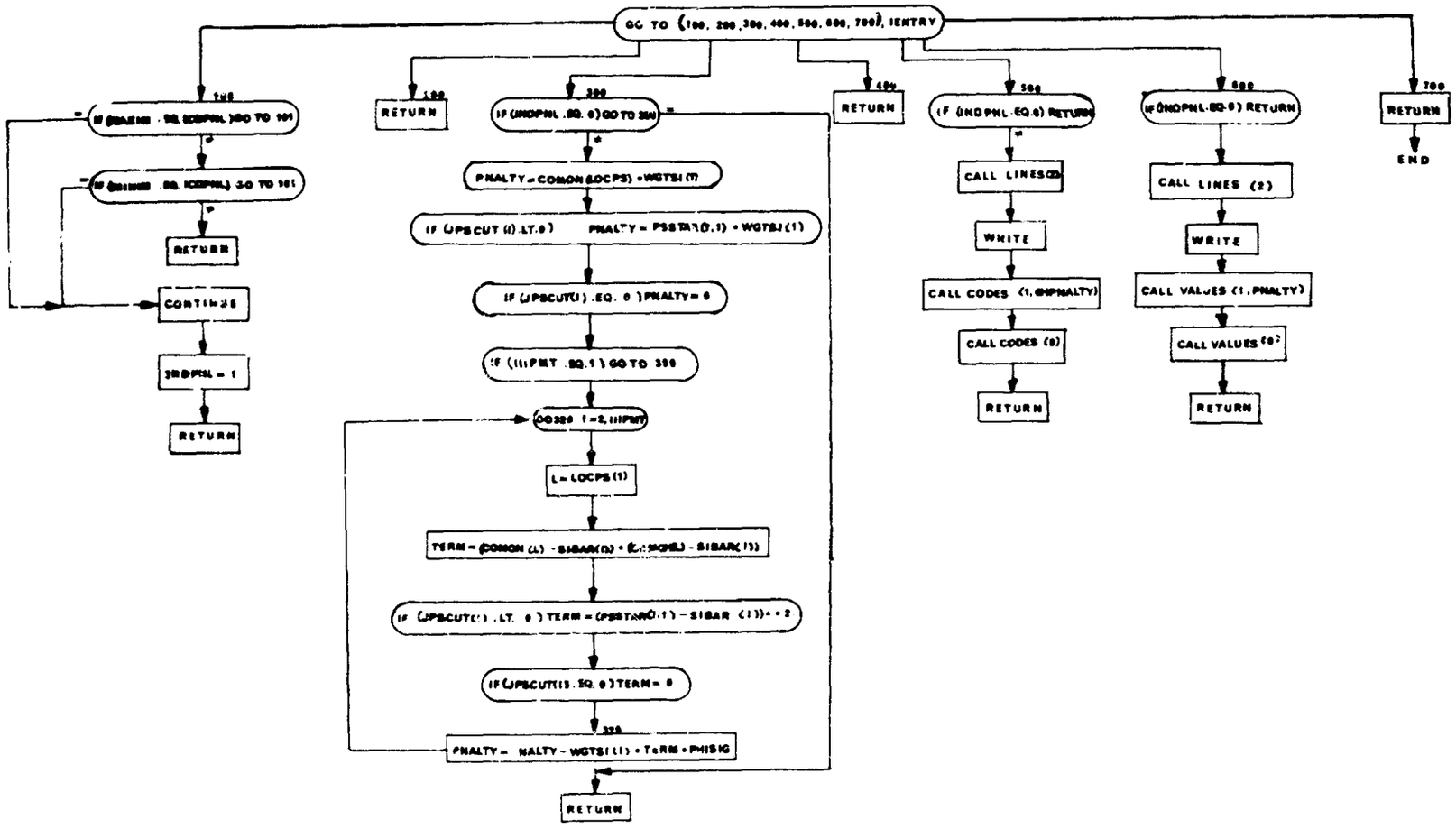
1. WGTSI and DSINL are two names for the same array.

2. Note that the number of functions is taken to be IIIPMT. (This is set in MAIN1.).

3. The penalty function must be computed when using control system CTLS2. Thus, when using CTLS2, MAXIM or MINIM must contain the name "PNALTY", the variable being maximized or minimized must appear as the first name of the ENDCON array and the remaining names of the ENDCON array must be those variables that are being constrained.

4. PENAL2 creates a penalty function from the second vehicle variables, ψ_j. Use of the second vehicle's penalty function in cooperative variational optimization will require a program modification. PENAL2 is identical to PENAL except for the use of the second vehicle's COMMON blocks and auxiliary subroutines. A flow chart for PENAL is presented.

PENAL


GO TO (100, 200, 300, 400, 500, 600, 700), IENTRY
RETURN
CONTINUE
RETURN
RETURN
PNALTY = COMON (LOCPS) + WGTSI (1)
IF (JPSCUT (I) .LT. 0) PNALTY = PSSTAR(I,1) + WGTSI (1)
IF (JPSCUT(I) .EQ. 0) PNALTY = 0
IF (IIIPMT .EQ. 1) GO TO 350
L = LOCPS (1)
RETURN
RETURN
CALL LINES (2)
WRITE
CALL CODES (1, IHPNALTY)
CALL CODES (0)
RETURN
CALL LINES (2)
WRITE
CALL VALUES (1, PNALTY)
CALL VALUES (0)
RETURN
RETURN
END

9. PLTS and PLTS2 - Data-Gathering Routine

PURPOSE:

To record the time histories of a set of prescribed variables on a tape to be used later (GRAPH segment) for plotting.

Method:

The following assumptions are made

1. The variable whose time histories are to be saved are input (PLOTS card(s)).

2. The time histories are to be saved for each pass up to the sixth without requiring the tape.

3. The F and G matrix time histories are to be saved on the last forward trajectory of a cycle on a different tape than the other variables.

4. Saving of the F and/or G should be done as an option based on the value of the indicator INDFAG.

Tape 17 is used for the regular variables.
Tape ILTAP is used for the F and/or G.

Entry Point 1: Number of points for a trajectory is zeroed.

Entry Point 2: Tape 17 is written. NPASS, INDSTG and the point number for the pass, MPT (NPASS), are put on tape 17 along with the values of the variables that have been specified.

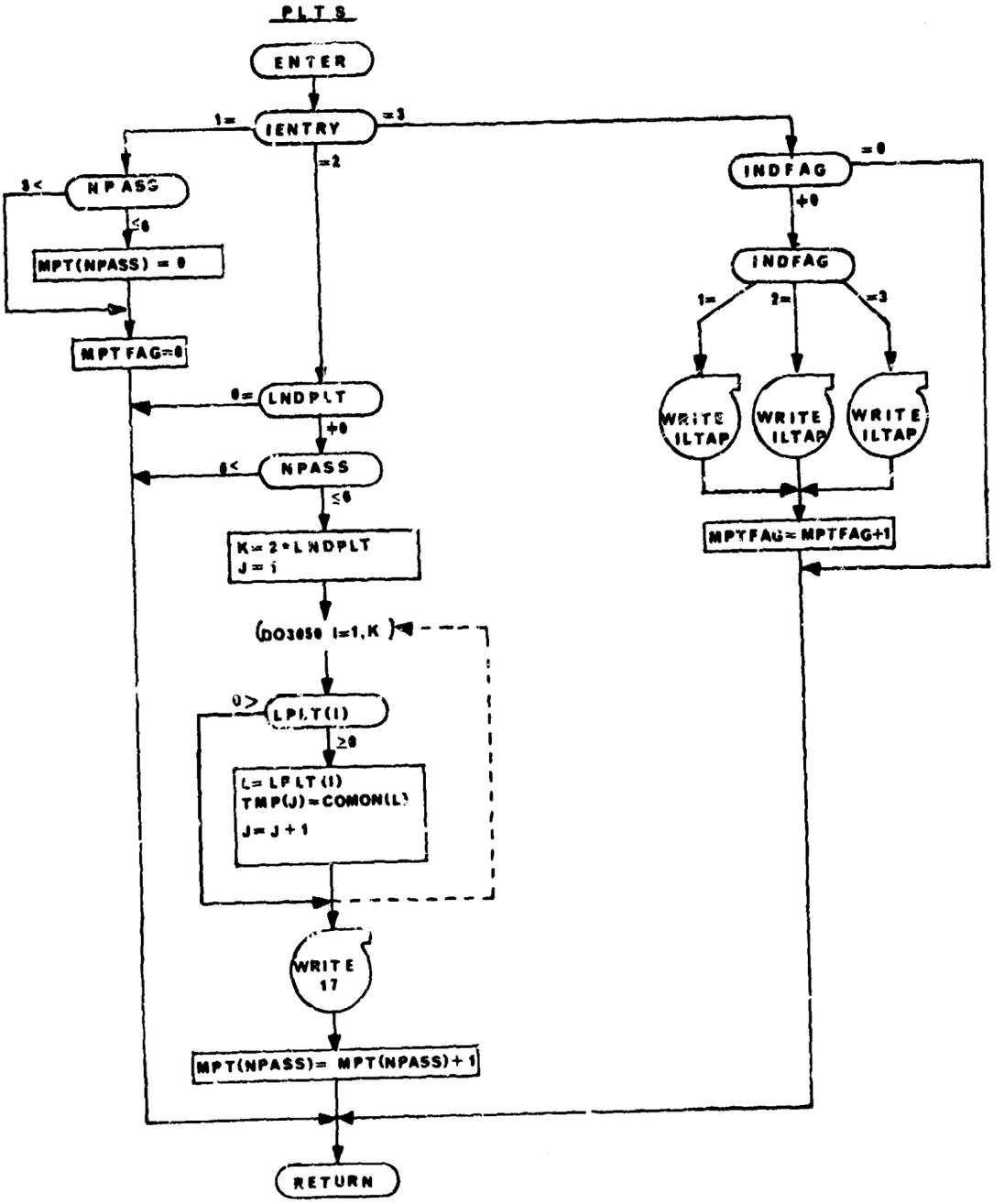
Entry Point 3: Tape ILTAP is written only if INDFAG $\neq$ 0. NPASS, INDSTG, the point number (MPTFAG) and TIME are put on ILTAP. In addition the following is put on ILTAP depending on the value of INDFAG:

INDFAG = 1 FMATX
= 2 GMATX
= 3 FMATX and GMATX

Entry point 2 is called by EXE after every (IGCONT+1) valid integration steps. Entry point 3 is called by EXE after the F and G are computed. IGCONT may be input.

Up to 40 variables may be specified in the PLOTS array by BCD names. Only distinct variables are written on tape. This means that if a variable is specified twice in the PLOTS array, its value will be put on tape only once per record.

The LPLT array is set up in MAIN1. $LPLT_i$ is either the COMMON subscript for the name in $PLOTS_i$ or is a negative integer j where j points to the cell in the LPLT array which contains the COMMON subscripts for the name in $PLOTS_i$. PLTS2 performs the PLTS function for vehicle 2.


PLTS
ENTER
IENTRY
1=
=2
=3
NPASS
≤0
MPT(NPASS) = 0
MPTFAG=0
LNDPLT
+0
NPASS
K= 2*LNDPLT
J = 1
DO3050 I=1,K
LPLT(I)
≥0
L= LPLT(I)
TMP(J)=COMON(L)
J=J+1
WRITE 17
MPT(NPASS)= MPT(NPASS)+1
RETURN
INDFAG
=0
+0
INDFAG
1=
2=
=3
WRITE ILTAP
WRITE ILTAP
WRITE ILTAP
MPTFAG= MPTFAG+1

10. OBSFUN and OBSFUN2 - Observation Function Routine

Purpose:

To provide an organized time history output of selected trajectory variables, the observation functions.

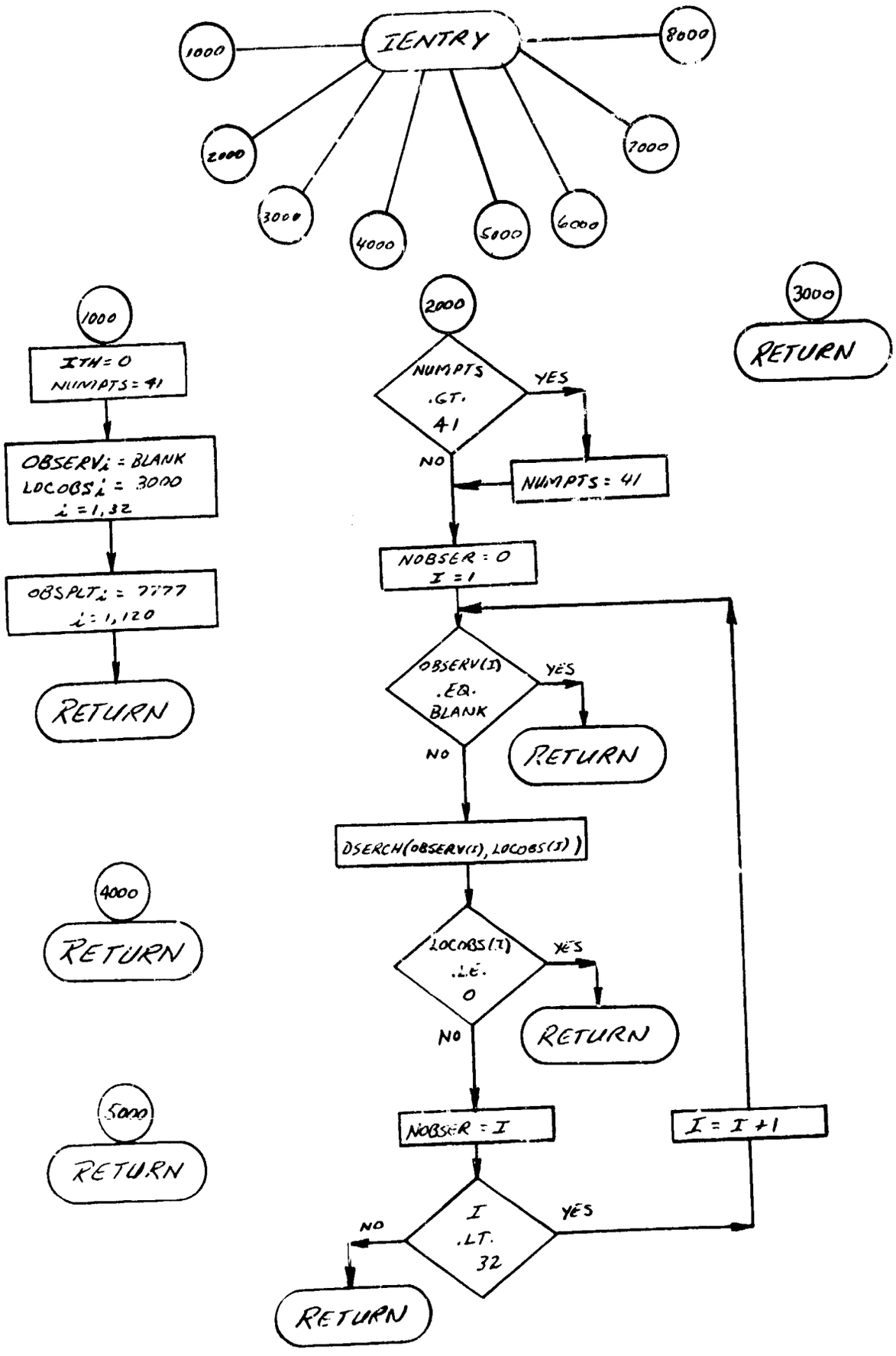
Method:

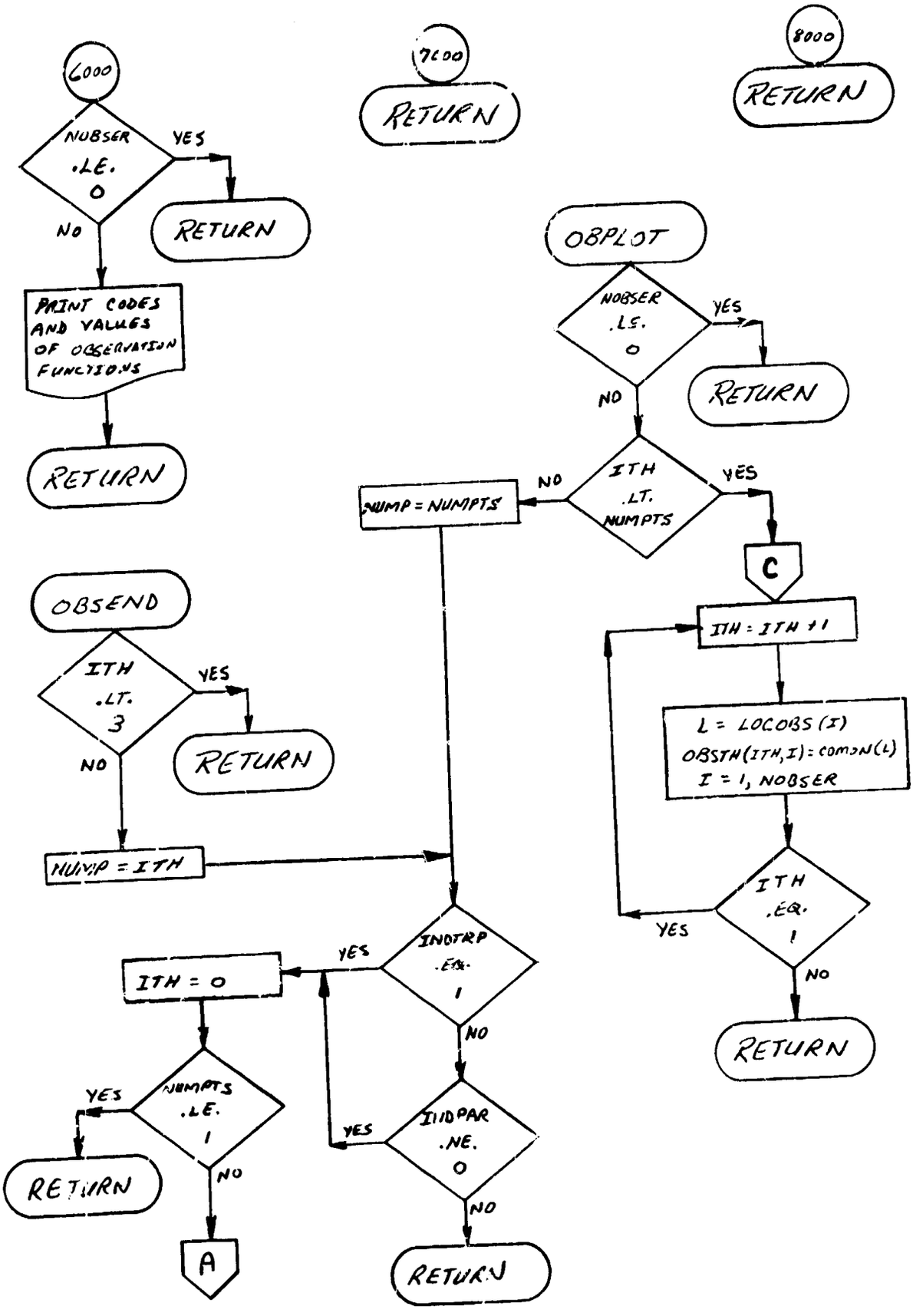
Input data defines a set of observation functions for each vehicle. Time histories of these variables are organized and printed in a compact form at set intervals of time. Selected observation functions may also be plotted on the printer using the printer-plotter routine, PAPERP.

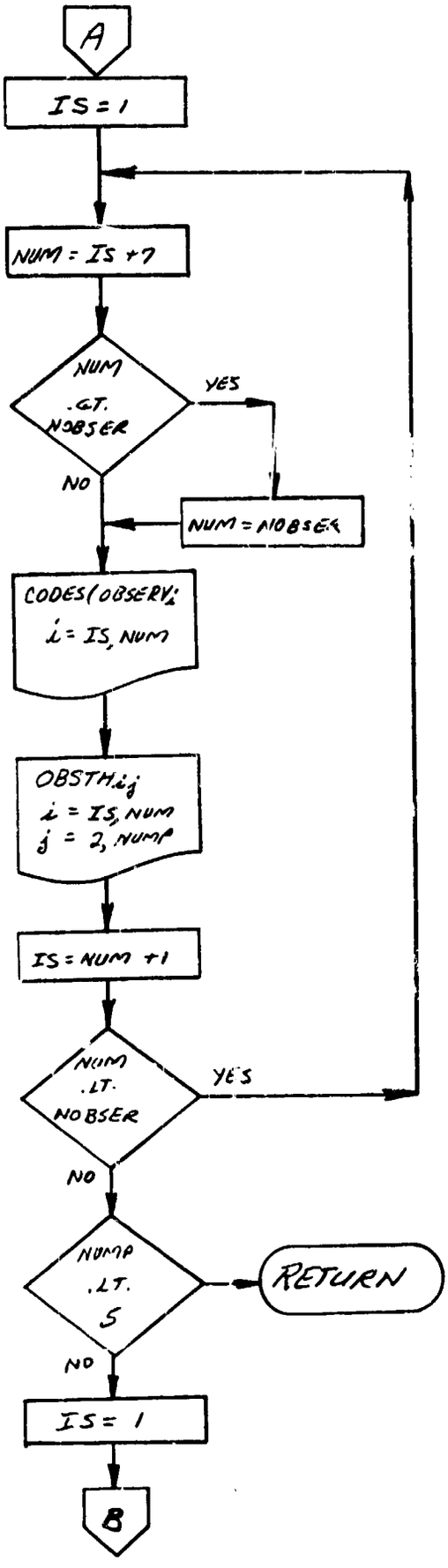
Remarks:

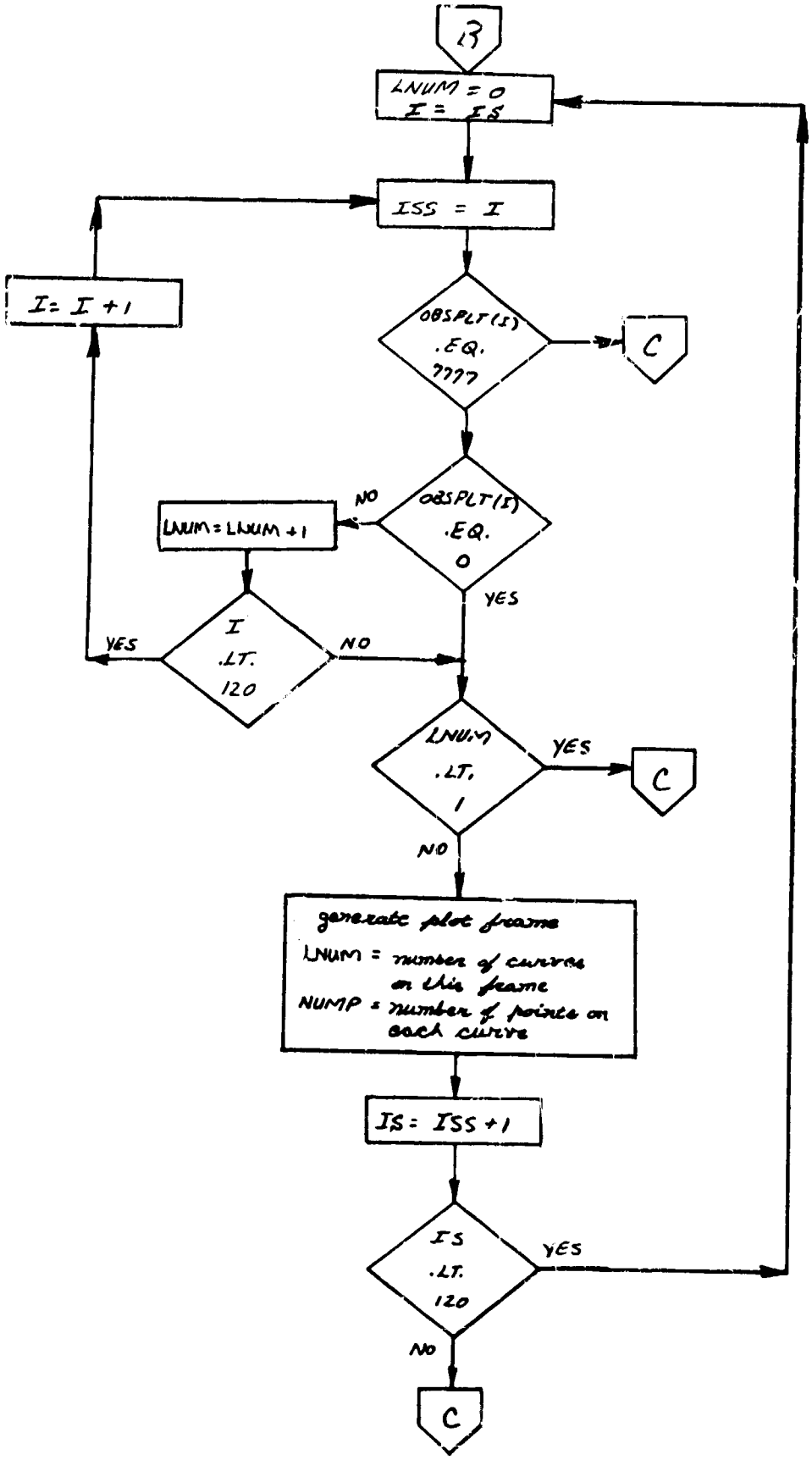
A flow chart for OBSFUN is presented. OBSFUN2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary routines.

OBSFUN


IENTRY
1000
2000
3000
4000
5000
6000
7000
8000
1000
ITH = 0
NUMPTS = 41
OBSERV$_i$ = BLANK
LOCOBS$_i$ = 3000
i = 1,32
OBSPLT$_i$ = 7777
i = 1,120
RETURN
2000
NUMPTS .GT. 41
YES
NO
NUMPTS = 41
NOBSER = 0
I = 1
OBSERV(I) .EQ. BLANK
YES
RETURN
NO
DSERCH(OBSERV(I), LOCOBS(I))
LOCOBS(I) .LE. 0
YES
RETURN
NO
NOBSER = I
I = I + 1
I .LT. 32
YES
NO
RETURN
3000
RETURN
4000
RETURN
5000
RETURN


6000
NUBSER .LE. 0
YES
RETURN
NO
PRINT CODES AND VALUES OF OBSERVATION FUNCTIONS
RETURN
7000
RETURN
8000
RETURN
OBPLOT
NOBSER .LE. 0
YES
RETURN
NO
ITH .LT. NUMPTS
NO
NUMP = NUMPTS
YES
C
ITH = ITH + 1
L = LOCOBS(I)
OBSTH(ITH,I) = COMON(L)
I = 1, NOBSER
ITH .EQ. 1
YES
NO
RETURN
OBSEND
ITH .LT. 3
YES
RETURN
NO
NUMP = ITH
INDTRP .EQ. 1
YES
ITH = 0
NO
NUMPTS .LE. 1
YES
RETURN
NO
A
IIIDPAR .NE. 0
YES
NO
RETURN


A
IS = 1
NUM = IS + 7
NUM .GT. NOBSER
YES
NO
NUM = NOBSER
CODES(OBSERV_i
i = IS, NUM
OBSTM_i,j
i = IS, NUM
j = 2, NUMP
IS = NUM + 1
NUM .LT. NOBSER
YES
NO
NUMP .LT. 5
RETURN
NO
IS = 1
B


B
LNUM = 0
I = IS
ISS = I
I = I + 1
OBSPLT(I)
.EQ.
7777
C
OBSPLT(I)
.EQ.
0
NO
YES
LNUM = LNUM + 1
I
.LT.
120
YES
NO
LNUM
.LT.
1
YES
C
NO
generate plot frame
LNUM = number of curves on this frame
NUMP = number of points on each curve
IS = ISS + 1
IS
.LT.
120
YES
NO
C

11. FILTER and FILTER2 - Repeater Routines for h-Transformation

Purpose:

To detect if a change has been made in the value of a prescribed COMMON variable and if a change has been made to produce either the original or the altered value when asked to do so.

Method:

There are three entry points to FILTER.

FILTER (1): The current values are saved; a filter is laid over each variable.

FILTER (2): Altered values of variables that are trapped in the filter are saved; the original values of the variables are restored.

FILTER (3): A variable for which there is an altered value is set equal to the altered value.

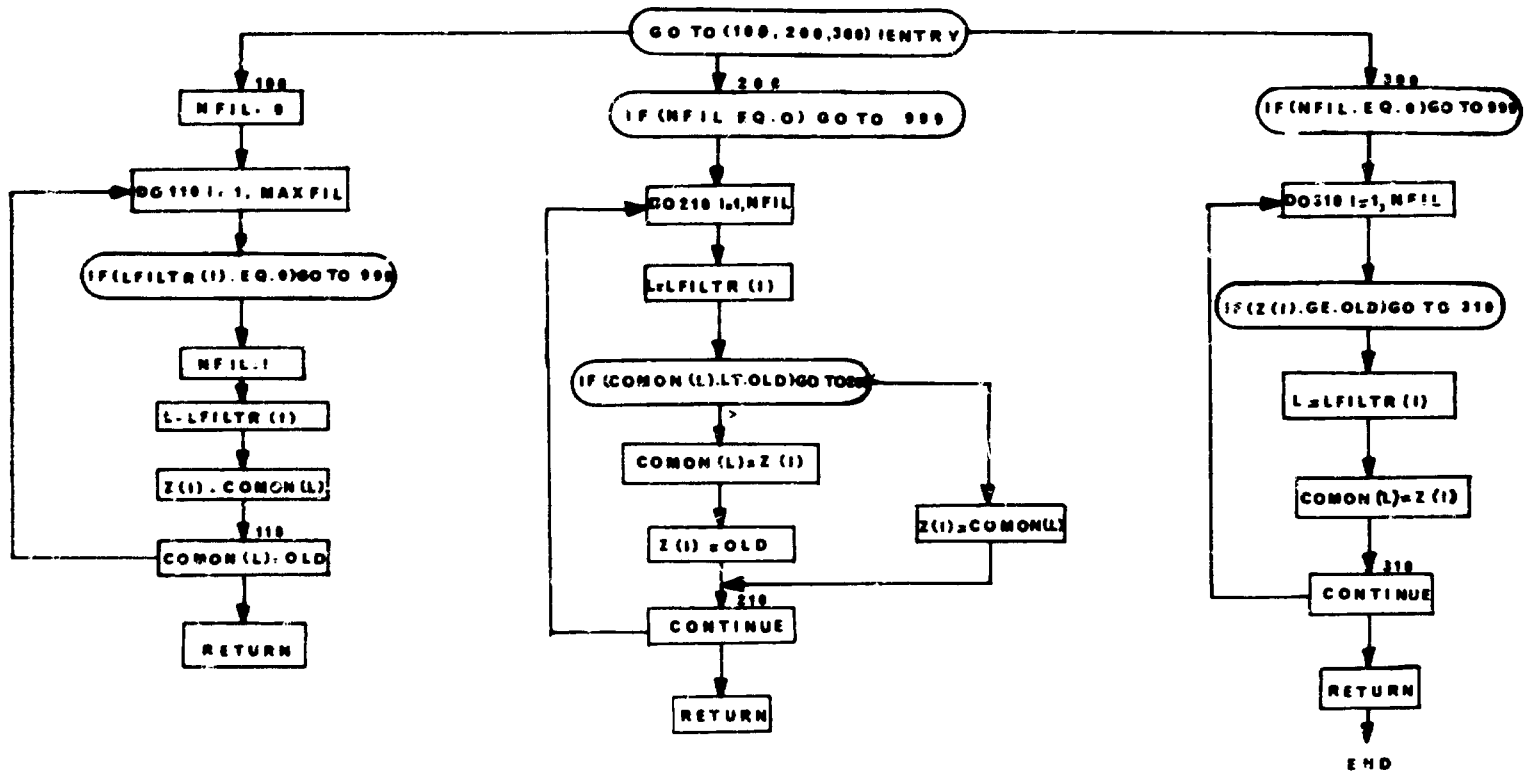
If FILTER(1) is ever called, FILTER (2) must be called before the values being filtered are ever used. FILTER(3) may be called any number of times after FILTER(1) and FILTER(2) have been called. FILTER(3) is the entry point which reproduces the detected change. FILTER(1) and FILTER(2), in conjunction, detect the change.

The variables to be filtered are prescribed by putting their COMMON locations successively into the cells of the name-common block/FILTER/. At most 14 variables may be filtered.

Remarks:

FILTER2 detects changes in the value of specified second vehicle COMMON variables. FILTER2 is identical to FILTER except for the use of vehicle 2's COMMON blocks. A flow chart of FILTER is provided.

FILTER


GO TO (100, 200,300) IENTRY
100
NFIL. 0
DO 110 I. 1, MAXFIL
IF(LFILTR (I). EQ.0)GO TO 999
NFIL.I
L-LFILTR (I)
Z(I) . COMON(L)
110
COMON (L): OLD
RETURN
200
IF (NFIL EQ.0) GO TO 999
DO 210 I=1,NFIL
L=LFILTR (I)
IF (COMON (L).LT.OLD)GO TO
COMON (L)=Z (I)
Z (I) =OLD
Z(I)=COMON(L)
210
CONTINUE
RETURN
300
IF(NFIL. EQ. 0)GO TO 999
DO 310 I=1, NFIL
IF(Z (I).GE.OLD)GO TO 310
L=LFILTR (I)
COMON (L)=Z (I)
310
CONTINUE
RETURN
END

12. STGTST and STGTST2 - Stage Testing Routine

Purpose:

(1) To determine if the trajectory should be terminated (i.e., $\Omega_1 = \bar{\Omega}_1$).

(2) To determine if a stage should be terminated (i.e., $\Omega_i = \bar{\Omega}_i$ $i = 2, \ldots, 4$)

(3) To determine an integration step size to use to achieve condition (1) or (2).

(4) To communicate the information that is gleaned to EXE proper for the appropriate action.

This subprogram has the standard entry points. Code and value printing are used only for diagnostic purposes and are called if and only if INDCOT is input $\neq 0$.

Up to four variables may be prescribed for terminating the trajectory or the stage by the BCD names in the array CUTOFF. The respective values of these variables at which termination is desired appear in the array OMBAR.

Termination Criteria

(1) $\left|\Omega_i - \bar{\Omega}_i\right| \leq 10^{-6}\left(\left|\bar{\Omega}_i\right| + 1\right)$

(2) it is not the first point of a major stage.

(3) if $i = 1$ then TIME $\geq$ TOMAX - DTMOD

If the above three conditions are satisfied for the i'th variable listed in the CUTOFF array, then INDSTG is set equal to i at STGTST (3). This informs EXE to terminate the stage if $i = 2$, 3, or 4 or to terminate the trajectory if INDSTG = 1.

Termination Algorithm

(1) t_s^p is the value of stage time, t_s, at the last valid integration step. $\Omega_i{}^p$ is the value of Ω_i on the last valid integration step. STGTST(3) updated $\Omega_i{}^p$ and $t_s{}^p$ only if it does not reject the integration step (or if it is the first point of a major stage).

(2) STGTST(3) rejects an integration step (sets LOOK $\neq 0$) only if the termination criteria is not satisfied and $(\Omega_i - \bar{\Omega}_i)(\Omega_i^p - \bar{\Omega}_i) < 0$ for some i and TIME > TOMAX - DTMOD if $i = 1$.

(3) For each i which dictates that the integration step be rejected an integration step size h_i is computed.

$$h_i = \frac{\bar{\Omega}_i - \Omega_i^p}{(\Omega_i - \Omega_i^p) / (t_s - t_s^p)}$$

This is the step size which the linear prediction estimates should be taken from the last valid integration step in order to exactly satisfy $\Omega_i = \bar{\Omega}_i$.

(4) The minimum of all the h_i's is determined; call it h*. The step size HO, to use on the next integration step is set equal to h* sign (HO). LOOK has been set nonzero; this informs EXE that the integration is to be backed up to the last valid step.

(5) INGHOM is set to 1 whenever STGTST(3) rejects an integration step. Within a given stage, STGTST(3) may reject an integration step on account of the behavior of Ω_i at most 20 times. If this limit is reached the staging criteria is assumed to be satisfied (even if it is not in fact) for the variable in question.

(6) Within a given stage, each time STGTST(3) accepts an integration step after INGHOM has once been set to 1, INGHOM is incremented by 1. The iteration is terminated if INGHOM ever reaches 100; if this happens, INDSTE is set to 0 to indicate an error in the trajectory.

Remarks

(1) Specifying TOMAX and DTMOD permit dealing with a multiple valued cutoff function; cutoff cannot be achieved until TIME > TOMAX - DTMOD. TOMAX and DTMOD may be input; TOMAX is set to the termination time of the trajectory of the previous cycle in CTLS. Care should be exercised when specifying TOMAX and DTMOD.

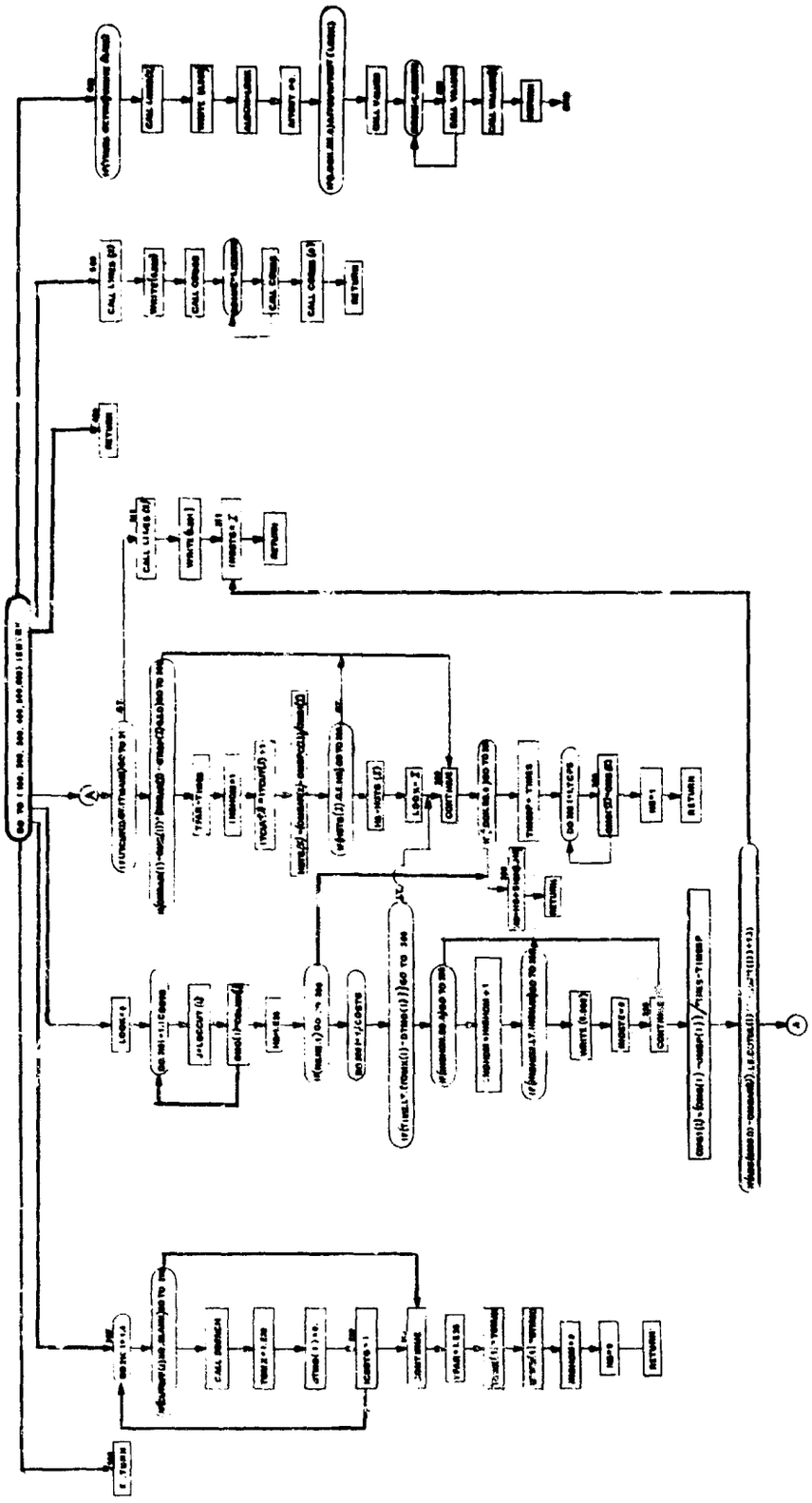
(2) BCDSTG is equivalenced to the second location of the CUTOFF array (in the directory). Likewise STEST is equivalenced to the second location of the OMBAR array. (Each of these may be input in stage data to define the staging variables and their desired staging values).

(3) Before stage data is read for a given stage, EXE proper blanks out CUTOFF(2), CUTOFF(3), CUTOFF(4), (i.e. the BCDSTG array).

(4) There is no restriction on the number of stages.

(5) No provision is made for specifying "increasing" or "decreasing" as an additional requirement of the termination criteria, but such a modification should not be difficult.

(6) STGTST2 extends the stage testing function to the second vehicle. STGTST2 is identical to STGTST except for the use of the second vehicle's COMMON blocks and auxiliary routines. EXE directly controls STGTST2. A flow chart for STGTST is presented.



13. EXTRAN and EXTRAN2-Driver for h-Transformation

Purpose:

To drive various combinations of the initial transformation and the h-transformation determined by the value of the indicator INDHTR.

Methods:

EXTRAN is a *special* subprogram of EXE. Its entry points are not standard.

Entry Point

1 INDHTR is set to 0 and the number of the location of, and the values of the state variables are saved. Values of variables that are to be filtered are saved by a call to FILTER(1).

2 The new values of the filtered variables are picked up by a call to FILTER(2).

3 The initial transformation or h-transformation is driven according to the value of INDHTR:

INDHTR

0 No calculations

-1 or 1 Initial Transformation: $\xi^{+}\xrightarrow[\text{DIFEQ(2)}]{}x^{+}$

2 h-transformation:

$$x^{-}\xrightarrow[\text{DIFEQ(3)}]{}\xi^{-}\xrightarrow[\text{FILTER(3)}]{}\xi^{+}\xrightarrow[\text{DIFEQ(3)}]{}x^{+}$$

3 h-transformation:

$$x^{-}\xrightarrow[\text{DIFEQ(3)}]{}\xi^{-}\xrightarrow[\text{DIFEQ(8)}]{}\xi^{+}\xrightarrow[\text{FILTER(3)}]{}\xi^{+}\xrightarrow[\text{DIFEQ(2)}]{}x^{+}$$

4 h-transformation:

$$x^{-}\xrightarrow[\text{DIFEQ(8)}]{}x^{+}\xrightarrow[\text{FILTER(3)}]{}x^{+}$$

where x^{-} denotes values of state variables at end of last stage.

x^{+} denotes values of state variables at beginning of current stage.

ξ^{-} denotes values of auxiliary variables (i.e. functions of the state variables) at end of last stage.

ξ^{+} denotes values of auxiliary variables at beginning of current stage.

Usage:

At the beginning of the trajectory EXE proper sets INDHTR to 1 in order to accomplish the initial transformation.

EXTRAN(1) is called by EXE at the beginning of every stage after the first, before the stage data for that stage is read. EXTRAN(2) is called after the stage data is read. And, of course, EXTRAN(3) is called to do the initial or h-transformation in order to initialize the values of the variables that are to be integrated.

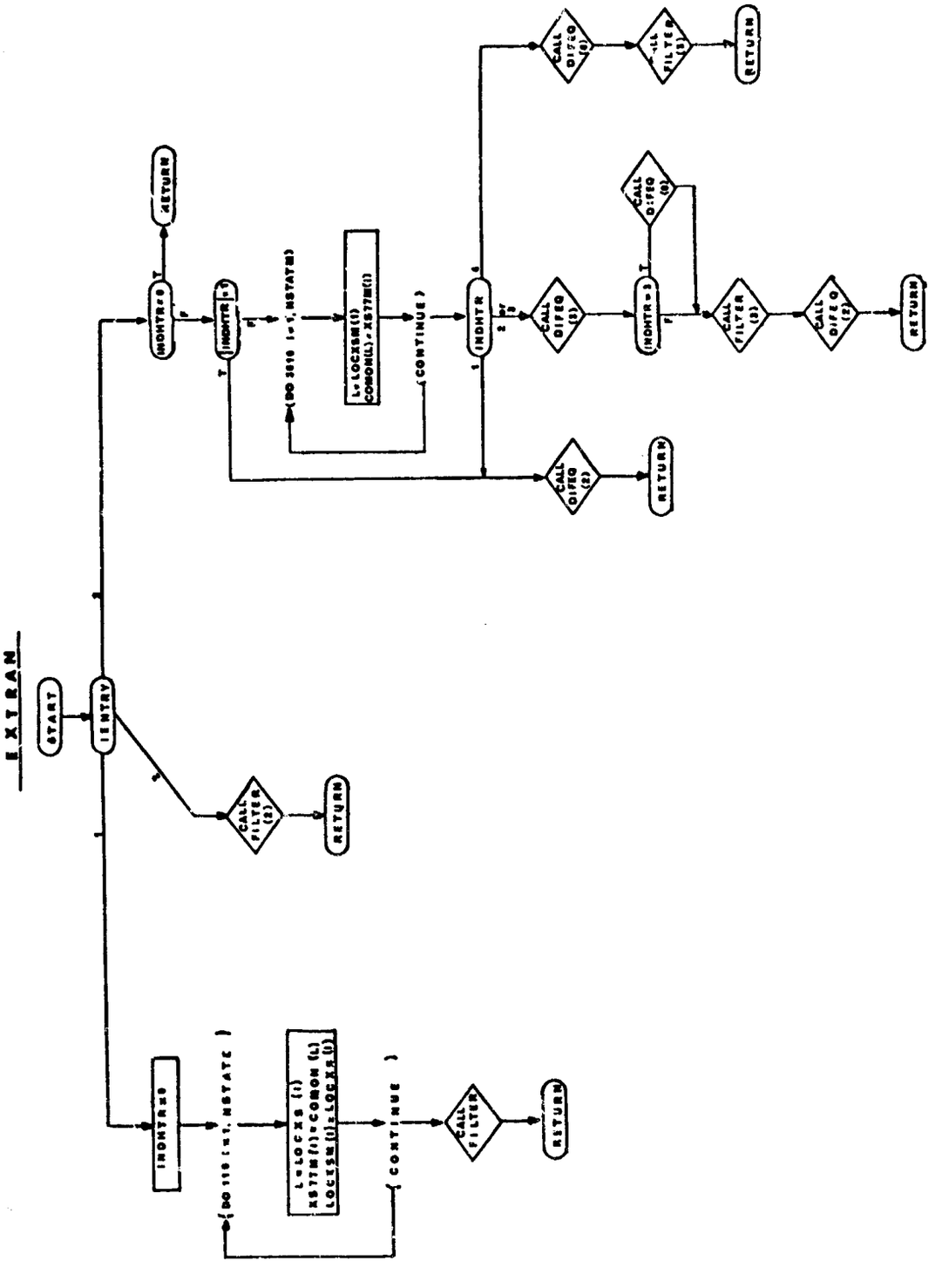
Remarks:

The reason for having several basic options for the h-transformation is in order to achieve flexibility. It is assumed that DIFEQ(2) is the inverse of the DIFEQ(3) transformation (and vice-versa) with respect to the values of the state variables and the auxiliary variables. DIFEQ(8) is either a transformation from the state variables to the state variables directly (INDHTR = 4) or from the auxiliary variables to the auxiliary variables (INDHTR = 3). FILTER (3) is the transformation equivalent to the reading of data for those variables that are set up to be filtered. This permits some simple types of h-transformations (INDHTR = 2) to be accomplished within the data without the need for introducing new code at DIFEQ(8), and also permits the other types of h-transformations to be modified to a limited extent right within the data.

On a trajectory for which partials are being computed EXTRAN(3) will be entered in the process of computing the P matrix and/or the R matrix. This action is initiated by PARTS. Negative values of INDHTR are to indicate that the P-matrix for that particular h-transformation is the identity and hence need not be computed.

The indicator INDHTR plays an important role in EXE proper too. ND is an indicator which is normally 2 to denote the beginning of a major stage after stage 1. If a P matrix is computed at a stage point (i.e. INDHTR > 0) then ND is set to 3 to inform R&V to expect a P matrix on the partial tape.

EXTRAN2 performs the identical and h-transformation for Vehicle 2. It is called by EXE2 and is identical to EXTRAN other than for use of Vehicle 2 COMMON blocks and auxiliary subroutines. A flow chart for EXTRAN is provided.


EXTRAN
START
IENTRY
CALL FILTER (0)
RETURN
INDNTR=0
(DO 110 I=1, NSTATE)
L=LOCXS (I)
XSTTM(I)=COMON (L)
LOCSM(I)=LOCXS(I)
(CONTINUE)
CALL FILTER
RETURN
INDNTR=0
RETURN
INDNTR=1
(DO 3010 I=1, NSTATE)
L=LOCSM(I)
COMON(L)=XSTTM(I)
(CONTINUE)
(INDNTR
CALL DIFEQ (2)
RETURN
CALL DIFEQ (1)
INDNTR=2
CALL FILTER (2)
CALL DIFEQ (2)
RETURN
CALL DIFEQ (0)
CALL FILTER (1)
RETURN
CALL DIFEQ (0)

14. INTGRT , INTGRT2, and INTGRTR - Interface for Integration Routine

Purpose:

To serve as an interface between the integration routine proper (MIMINF or MIMINR) and any routine requesting a variable to be integrated; also to backup the integration.

There are five logical functions which this routine performs depending on the status of the indicator INTCAL. For a particular call to INTGRT one of these functions will be enacted. The P array is the array of current derivative values of the variables that are being integrated. The Y array is the array of the current integrated variable values.

Usage:

To integrate XDOT and have the resulting integrated value stored in X use the statement

CALL INTGRT (INTNUM, XDOT, X)

INTNUM must be a distinct call used in, and only in, the calls for the integration of XDOT.

A call of the above form must be made at three different times (when the value of INTCAL is 2, 3, and 4). However, all of this may be accomplished with only one statement if the statement is inserted at the entry point 7 of the calling program.

INTNUM is the subscript in the P and Y arrays for the values XDOT and X respectively.

Method:

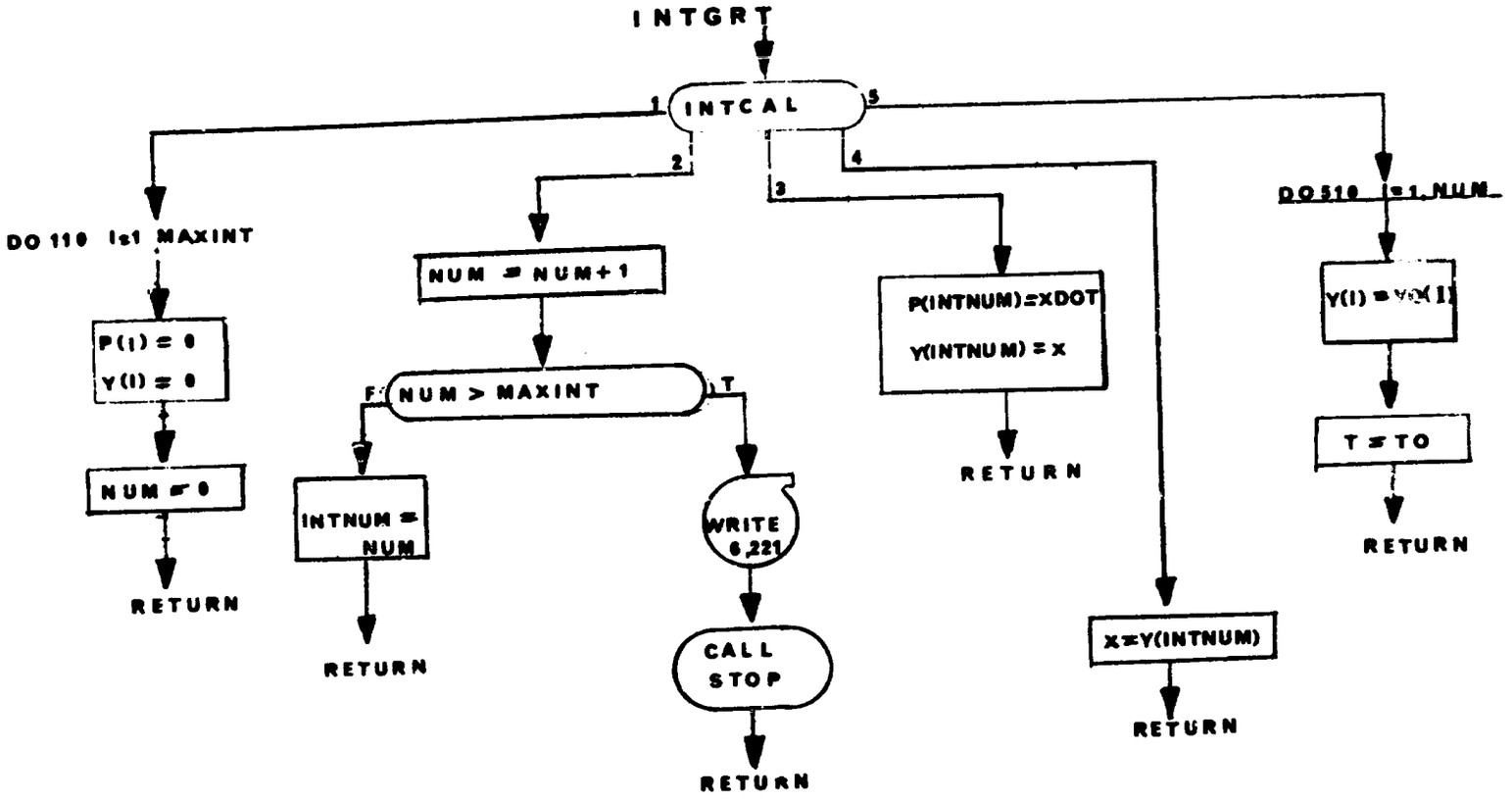
INTCAL = 1	the P and Y arrays are set to 0; the number of integrated variable is set to 0.
INTCAL = 2	the subscript INTNUM is computed for the arguments XDOT and X; the number of integrated variables is updated by 1.
INTCAL = 3	the value of XDOT is put into its proper place in the P array.
INTCAL = 4	the value of the integrated variable is picked up from the proper place in the Y array and put in X.
INTCAL = 5	the integration is backed up one step by resetting the Y array to the previous Y array (YO) and by resetting T to TO.

INTGRT will terminate the case if more variables are requested to be integrated than there is room for in the integration arrays; at present, this upper limit is 25 variables in the forward trajectory, and 300 in the reverse trajectory.

EXE proper is the only routine which calls INTGRT when INTCAL = 1 or when INTCAL = 5. EXE has sole control over setting the indicator INTCAL.

INTGRT2 serves as an interface between the integration routine MIMINF2 for the second vehicle and any second vehicle subroutine requesting a variable to be integrated. It is identical to INTGRT except for use of Vehicle 2's COMMON blocks. INTGRT2 is directly controlled from EXE. A flow chart for INTGRT is provided.

INTGRTR serves as an integration interface routine for the reverse time integration of the adjoint equations through MIMINR and the REV program. It is identical to INTGRT other than for use of the UPDCAR COMMON block and subroutine STOP2.


INTGRT
INTCAL
1
2
3
4
5
DO 110 I=1 MAXINT
P(I) = 0
Y(I) = 0
NUM = 0
RETURN
NUM = NUM+1
F
NUM > MAXINT
T
INTNUM = NUM
RETURN
WRITE 6,221
CALL STOP
RETURN
P(INTNUM)=XDOT
Y(INTNUM) = X
RETURN
X=Y(INTNUM)
RETURN
DO 510 I=1,NUM
Y(I) = Y(I)
T = TO
RETURN

15. CODES and CODES2 - Code Print Routines

Purpose

To print code names of variables to identify the values in the output.

Usage

CALL CODES (L, A1,...,AL)

L — is an integer $0 \leq L \leq 8$ identifying the number of arguments following it.

A1, ., ., ., AL — L cells each containing a Hollerith code word of at most 10 characters.

The above call adds the L Hollerith code words to the list of code words to be output on the next line of print. When 8 code words have been accumulated, the line is printed; any excess code words are added to the list for the next line of print. If L is 0, then the codes in the existing list (the number may be less than eight) are printed immediately.

Lines accounting is taken care of within this routine.

This subroutine is designed to be used in conjunction with the VALUES subroutine.

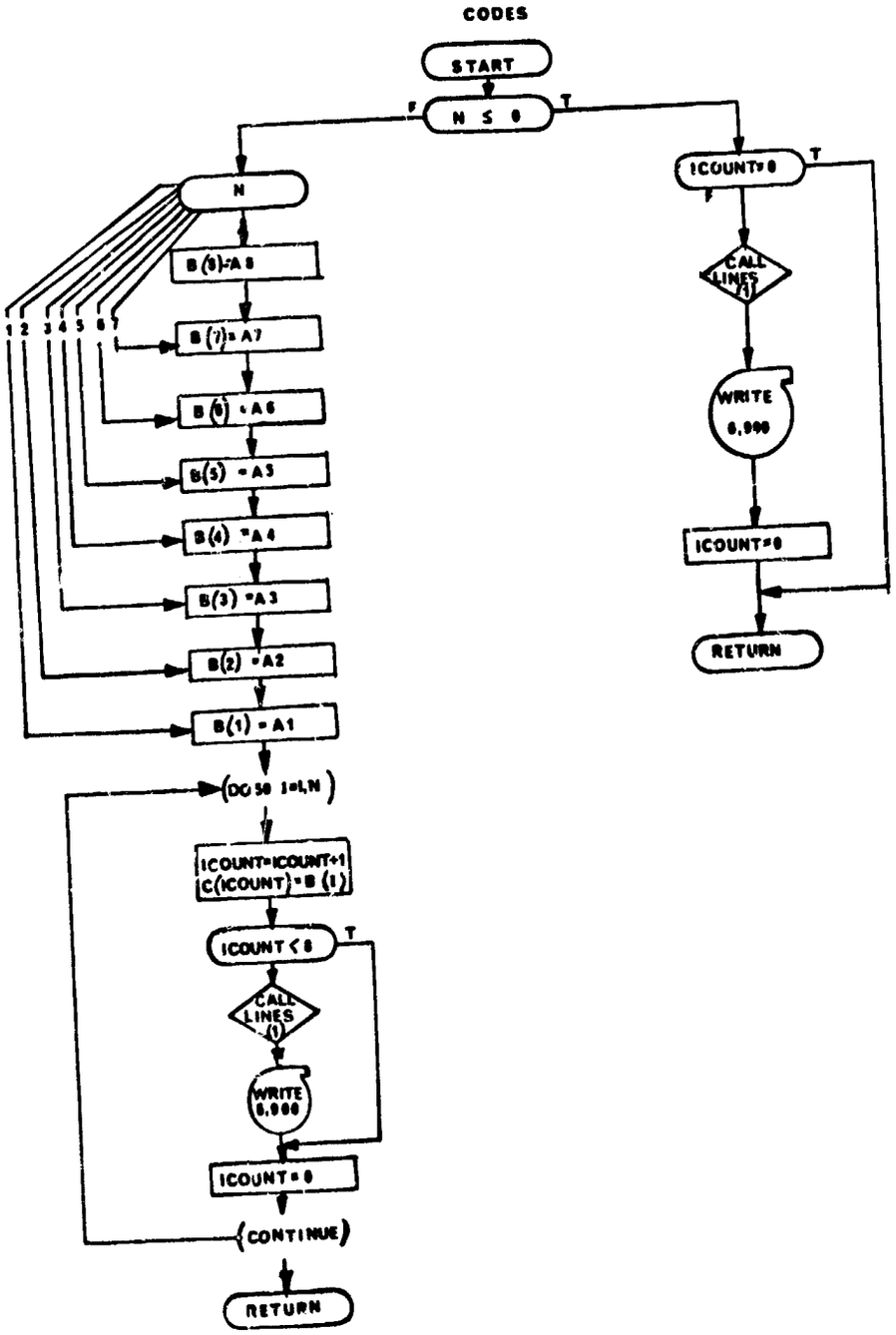
Codes are printed with a "5X,A10" format.

CODES and VALUES control small secondary output buffers within the program itself.

Normally a call to CODES is made before any call to VALUES in order to identify the values. The CALL CODES (0) is necessary to be sure that the codes are printed; the CALL VALUES (0) is necessary to be sure the values are printed. (If this is not done the buffers may never be flushed.)

Remarks:

A flow chart for CODES is presented. CODES2 is identical to CODES but is required to insure correct output function for the second vehicle.


CODES
START
N ≤ 0
F
T
N
1 2 3 4 5 6 7
B(8)=A8
B(7)=A7
B(6)=A6
B(5)=A5
B(4)=A4
B(3)=A3
B(2)=A2
B(1)=A1
(DO 50 I=1,N)
ICOUNT=ICOUNT+1
IC(ICOUNT)=B(I)
ICOUNT<8
T
CALL LINES (1)
WRITE 6,900
ICOUNT=0
(CONTINUE)
RETURN
ICOUNT>0
T
F
CALL LINES (1)
WRITE 6,900
ICOUNT=0
RETURN

16. ITEMS and ITEMS2 - Variable Print Routines

Purpose:

To allow printing of variables in addition to those printed by other subprograms of EXE at entry points 5 and 6.

Usage:

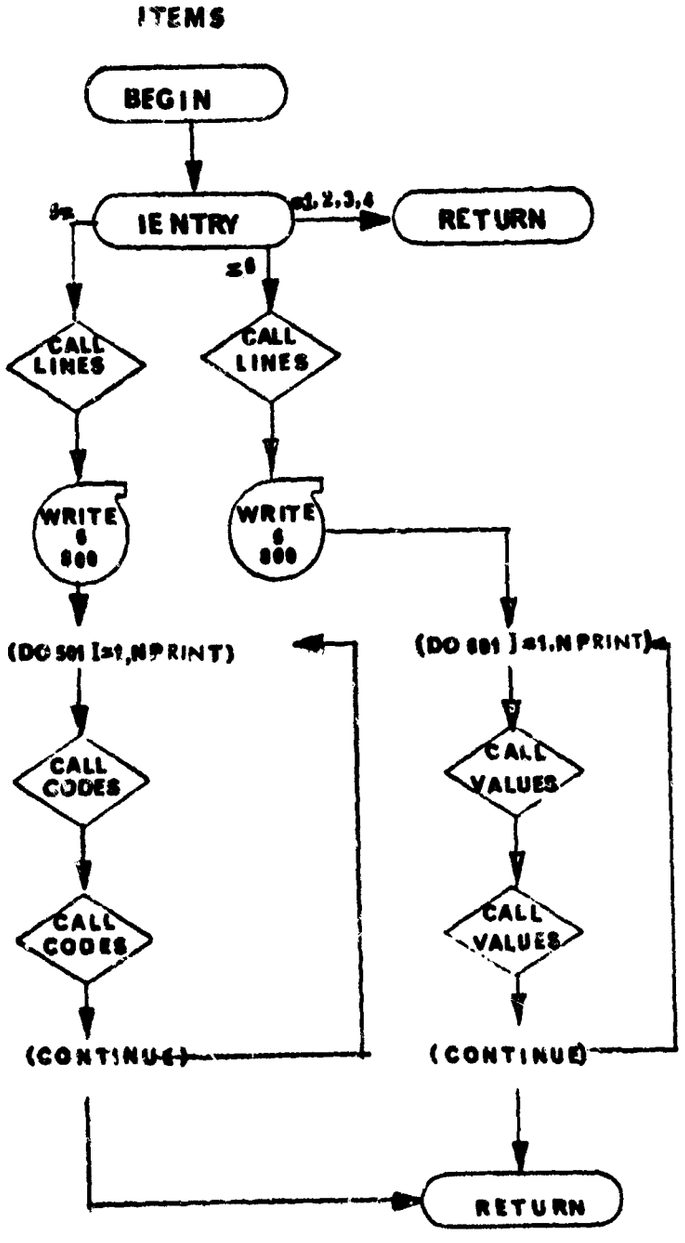
```
CALL ITEMS (IENTRY)
```

IENTRY = 5 prints BCD names of specified variables.
IENTRY = 6 prints values of specified variables.
CODES and VALUES are used for output.

All variables specified on the VPRINT card will be output. Up to twenty variables may be specified.

Remarks:

ITEMS is called by EXE.
ITEMS calls CODES and VALUES.
ITEMS2 is identical to ITEMS except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.
ITEMS2 is called by EXE2.
A flow chart for ITEMS is presented.


ITEMS
BEGIN
IENTRY
=1, 2, 3, 4
RETURN
=0
CALL LINES
CALL LINES
WRITE 6 800
WRITE 6 800
(DO 501 I=1,NPRINT)
(DO 601 I=1,NPRINT)
CALL CODES
CALL CODES
CALL VALUES
CALL VALUES
(CONTINUE)
(CONTINUE)
RETURN

17. COMBAT and COMBAT2 - Combat Control Routine

Purpose:

To set the initial values of the combat parameters and act as the calling program to other subroutines for combat role and tactics selection

Usage:

Entry is made to this routine by the following statement

CALL COMBAT (IENTRY)

where IENTRY is a fixed point variable.

IENTRY = 1

This is the pre-data initialization. At this entry the nominal values are established for the following combat parameters.

MNEMONIC	NOMINAL VALUE	DESCRIPTION
NGSPE	1.0	Load factor to use in the maximum specific energy maneuver.
DLTSPE	2.0	Number of seconds "Look Ahead" used to determine maximum specific energy path.
MSUBE	1.0	Mach number at which to start "NGSPE/2" path during maximum specific energy maneuver
MSUPE	1.2	Mach number at which to end "NGSPE/2" path during maximum specific energy maneuver.
GAMSPE	-15.0	Lower bound on flight path angle during "NGSPE/2" portion of maximum specific energy maneuver.
HSUBE	25000.	Altitude at which to start "NGSPE/2" path during maximum specific energy maneuver.
AFSUBE(1)	1.0	Factor applied to ALPNG to establish lower bound on angle of attack for subsonic portion of maximum specific energy path.
AFSUBE(2)	1.5	Factor applied to ALPNG to establish upper bound on angle of attack for subsonic portion of maximum specific energy path.

MNEMONIC	NOMINAL VALUE	DESCRIPTION
AFSUPE(1)	-1.0	Factor applied to ALPNG to establish lower bound on angle of attack for supersonic portion of maximum specific energy path.
AFSUPE(2)	1.5	Factor applied to ALPNG to establish upper bound on angle of attack for supersonic portion of maximum specific energy path.
INDDIV	0	Switch indicator set by program (Do not input as data).
IPCMSG	1	Print control for combat message 1 = Print combat messages 0 = Do not print combat messages
CPCON(1)	1.0	Factor applied to maximum angle of attack (AMAXD) to establish the upper bound on desired angle of attack (ALPDD)
CPCON(2)		
CPCON(12)	30.0	Role selection logic: attack incoming targets when CONBD $\leq$ CPCON(12)
CPCON(13)	60.0	Role selection logic cone angle for second sector: offensive role when CONBD $\leq$ CPCON(13)
CPCON(14)	1.0	
CPCON(15)	0.0	
CPCON(16)	10.0	
CPCON(17)	10.0	Role selection logic, take evasive action when ESERDT $\leq$ CPCON(17)

MNEMONIC	NOMINAL VALUE	DESCRIPTION
CPCON(18)	1.0	Throttle factor for hard turn defensive maneuver AND = ANMAX * CPCON(18)
CPCON(19)	1.0	Angle of attack factor for hard turn defensive maneuver ALPDD = AMAXD * CPCON(19)
CPCON(20) ↓ CPCON(29)	90.0 90.0	Table of desired band angles for hard turn defensive maneuver.
CPCON(3)		
CPCON(4)		
CPCON(5)	1.0	Finite control rate logic: factor on angle of attack rate ALPHD1 = ALPDOT * CPCON(5)
CPCON(6)	1.0	Finite control rate logic: rate adjustment factor for angle of attack RAFALP = ALPDOT * DELTS * CPCON(6)
CPCON(7)	1.0	Finite control rate logic: factor on bank angle rate BA77D1 = BADOT * CPCON(7)
CPCON(8)	1.0	Finite control rate logic: rage adjustment factor for bank angle RAFBA = BADOT * DELTS * CPCON(8)
CPCON(9)	1.0	Finite control rate logic: factor on throttle rate AN7771 = ANDOT * CPCON(9)
CPCON(10)	1.0	Finite control rate logic: rate adjustment factor for throttle RAFAN = ANDOT * DELTS * CPCON(10)

MNEMONIC	NOMINAL VALUE	DESCRIPTION
CPCON(11)	90.0	Role selection logic, outgoing target when PHOFD < CPCON(11)
CPCON(30)	1.0	Throttle factor for line-of-sight vector rotation defensive maneuver AND = ANMAX * CPCON(30)
CPCON(32)	1.0	
CPCON(33)	1.0	Angle of attack factor for Split S defensive maneuver ALPDD = AMAXD * CPCON(33)
CPCON(34)	1.0	Throttle factor for Split S Defensive maneuver AND = AMAXD * CPCON(34)
CPCON(35)	1.0	
CPCON(36)		
CPCON(37)		
CPCON(38)	1.0	Angle of attack factor for lag-pursuit offensive maneuver ALPDD = AMAXD * CPCON(38)
CPCON(39)		

MNEMONIC	NOMINAL VALUE	DESCRIPTION
CPCON(40)		
CPCON(41)		
CPCON(42)		
CPCON(43)		
CPCON(44)		
CPCON(45)		
CPCON(46)		
CPCON(47)		
CPCON(48)	1.0	Attacking maneuver: TC * CPCON(48) determines the time constant for elimination of pointing error.
CPCON(49)	1.0	Attacking maneuver: throttle control if RP RI77F then AND = ANMAX * CPCON(49)

MNEMONIC	NOMINAL VALUE	DESCRIPTION
CPCON(50)	1.0	Attacking maneuver: throttle control if RP ⩽ RI77F then AND = ANMAX * COCON(50)
$INDBNC_i$	0	Boundary control $INDBNC_i$ = 0, i^{th} boundary violation can not be controlled = 1, i^{th} boundary violation can be controlled by angle of attack
ALPDOT	20.0	Maximum rate at which angle of attack can be changed (DEG/SEC)
BADOT	45.0	Maximum rate at which bank angle can be changed (DEG/SEC)
ANDOT	.2	Throttle rate (Fraction of full throttle/ sec)
AMAXD	15.0	Maximum angle of attack (DEG)
ANMAX	1.0	Maximum throttle setting
VMINF	200.	Minimum vehicle velocity (Ft/Sec)

IENTRY = 2

This is the post-data initialization. At this entry the values of the array CPCON is output on unit 6.

IENTRY = 3

At this entry subroutines ANGLES, DETECT, ROLE1, HLIMIT, and CRATE are called role selection and to determine desired angle of attack, bank angle, and throttle setting.

IENTRY = 4

Entry for initial print.

IENTRY = 5

Entry to print mnemonic headings. The following codes are printed

ALPHD1 BA77D1 AN7771 ALPDD BAD7D AND ALPHD BA77D AN XAIMF

YAIMF ZAIMF

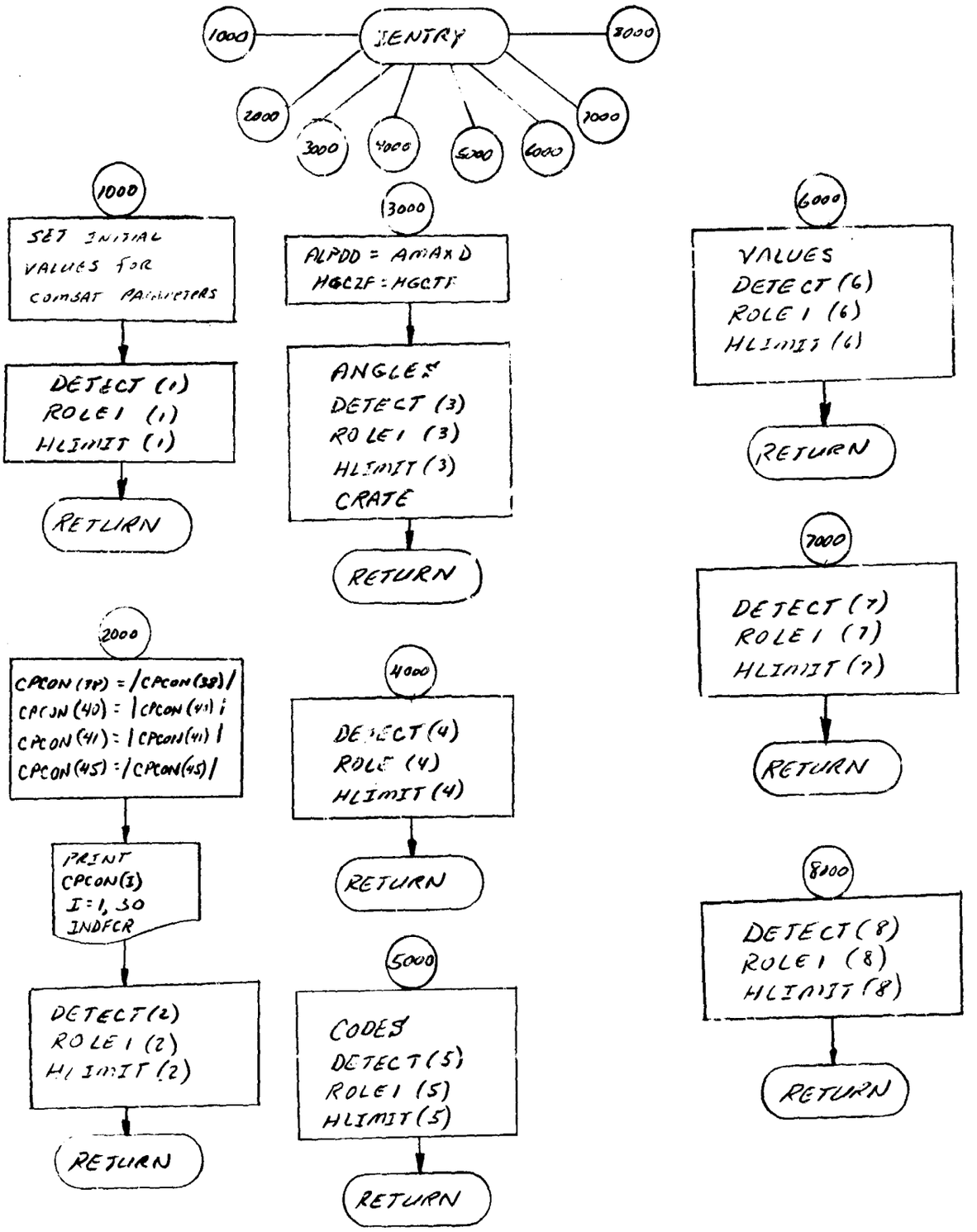
IENTRY = 6

At this entry the corresponding values to the mnemonic codes of Entry 5 are printed.

Remarks:

A flow chart for COMBAT is presented. COMBAT 2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

COMBAT


1000
ENTRY
3000
2000
3000
4000
5000
6000
7000
1000
SET INITIAL VALUES FOR COMBAT PARAMETERS
DETECT (1)
ROLE1 (1)
HLIMIT (1)
RETURN
3000
ALPDD = AMAXD
HGCZF = HGCTF
ANGLES
DETECT (3)
ROLE1 (3)
HLIMIT (3)
CRATE
RETURN
6000
VALUES
DETECT (6)
ROLE1 (6)
HLIMIT (6)
RETURN
2000
CPCON (39) = |CPCON (38)|
CPCON (40) = |CPCON (41)|
CPCON (41) = |CPCON (41)|
CPCON (45) = |CPCON (45)|
PRINT CPCON(I) I=1, 30 INDFCR
DETECT (2)
ROLE1 (2)
HLIMIT (2)
RETURN
4000
DETECT (4)
ROLE (4)
HLIMIT (4)
RETURN
5000
CODES
DETECT (5)
ROLE1 (5)
HLIMIT (5)
RETURN
7000
DETECT (7)
ROLE1 (7)
HLIMIT (7)
RETURN
8100
DETECT (8)
ROLE1 (8)
HLIMIT (8)
RETURN

18. MIMINF and MIMINF2 - Integration Routine

Purpose

To perform the calculations necessary to integrate an array of variables by the Runge-Kutta method; to determine an estimate of the truncation error and, with this information, to decide whether to accept or reject an integration step; to compute a new step size, HO, (based on the truncation error) to be used for the next integration step.

T - the independent variable for the integration

TO - the value of T at the last valid integration step

Y - array of current values of integrated variables

YO - an array of the values in the Y array at the last valid integration step

P - array of current values of derivatives of integrated variables

Usage:

In general, a numerical integration routine must make several intermediate or preliminary calculations of the integrated variables before an integration step is actually taken. This routine is organized so that for each basic entry point to it, one of the several intermediate calculations of the integrated variables is performed. The reason for organizing it this way is so that MIMINF does not have to be shackled with the responsibility of driving other calculations which need to be performed during an integration step; as a consequence, MIMINF itself is virtually void of complex logic.

EXE proper is the only routine which calls MIMINF. Communication between EXE and MIMINF is accomplished for the most part by the indicator MIMPAS. On the call to MIMINF at which the final values of the integrated variables for an integration step are computed, MIMINF sets MIMPAS = 0 in order to inform EXE that the step is tentatively all right. After each return from a basic call to MIMINF, EXE increases MIMPAS by 1. EXE makes each basic call to MIMINF with (the absolute value of) MIMPAS as the argument. Hence at each basic call to MIMINF the next intermediate calculation of the integrated variable is performed.

Now every integration step that is made has to be double checked by MIMINF; it is only when the next integration step is ready to start that there is enough information for MIMINF to get a good estimate of the truncation error that has occurred on the last integration step. Hence at this time, EXE makes the terminal call to MIMINF (entry point 5) for the truncation error check. If the truncation error is too large MIMINF (5) sets MIMPAS = -1; this informs EXE that the integration is to be backed up and the last integration step is to be attempted again with a smaller step size h (i.e. HO) determined by MIMINF (5).

Method:

The method employed here performs three intermediate calculations of the integrated variables, y_A, y_B, y_C, before the final value of the integrated variables, y_{n+1}, is calculated.

Entry point 1: $y_A = y_n + h/2\ \dot{y}_n$

Entry point 2: $y_B = y_n + h/2\ \dot{y}_A$

Entry point 3: $y_C = y_n + h\ \dot{y}_B$

Entry point 4: $y_{n+1} = y_n + h/6\ (\dot{y}_n + 2\dot{y}_A + 2\dot{y}_B + \dot{y}_C)$

(The calculation of the necessary derivatives (i.e. $\dot{y}_n, \dot{y}_A, \dot{y}_B, \dot{y}_C$ will have been done elsewhere by the time that entry point of MIMINF is called. The general form of these derivatives is

$\dot{y}_n = f\ (y_n, t_n)$

$\dot{y}_A = f\ (y_A,\ t_n + h/2)$

$\dot{y}_B = f\ (y_B,\ t_n + h/2)$

$\dot{y}_C = f\ (y_C,\ t_n + h)$

The derivatives are picked up from the P array. The values of the integrated variables are put in the Y array. At present, at most 25 variables may be integrated.

Entry point 5 Truncation error estimation and step size control

At this entry point, the truncation error T_n is estimated as

$$T_n = h \left[\frac{\dot{y}_{n+1} - \dot{y}_C}{y_{n+1}} \right]$$

This is done for each integrated variable. Finally, T_n^* is defined as the maximum in absolute value of all the T_n's. Now there are three possibilities:

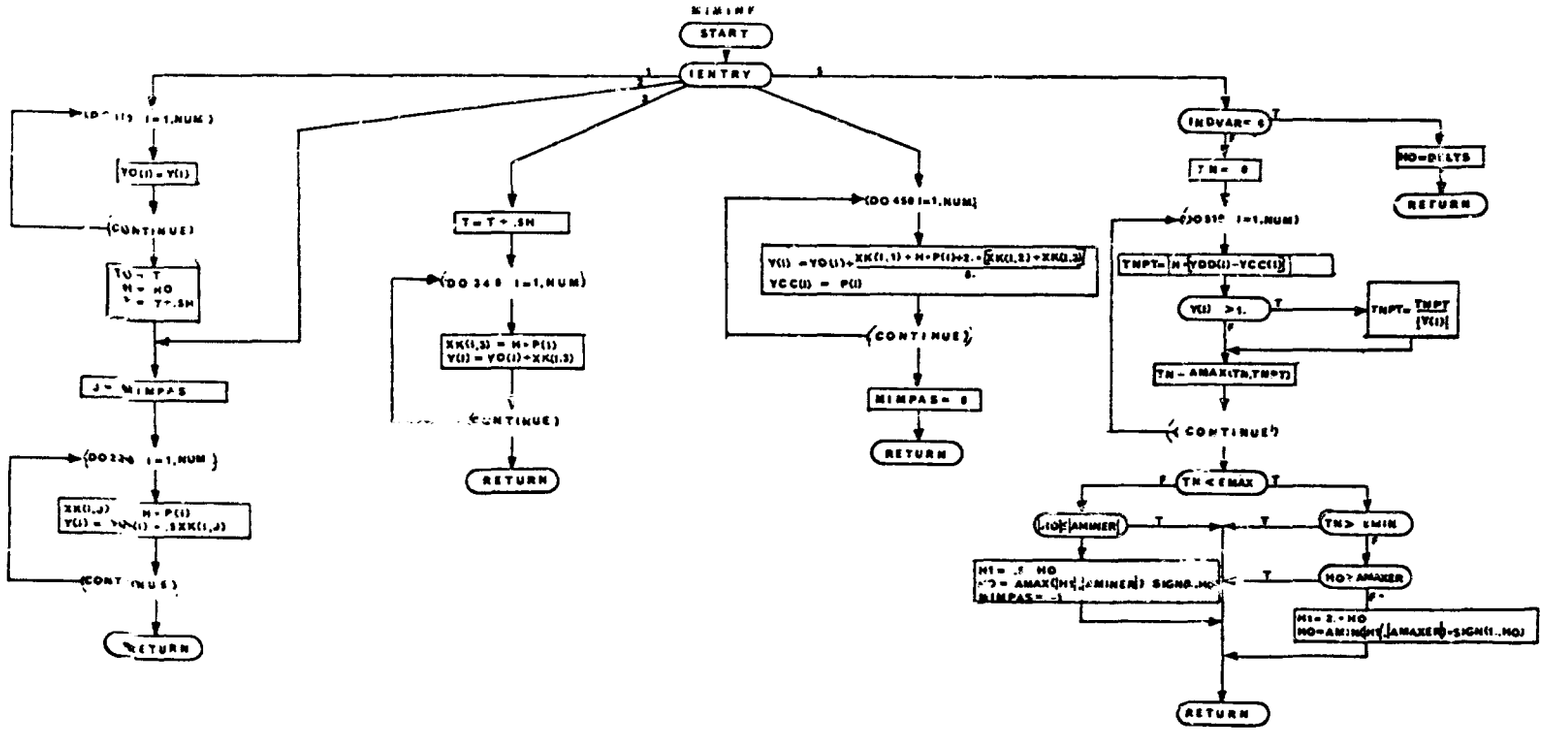
(a) EMIN $\leq T_n^* <$ EMAX : in this case MIMINF (5) accepts the integration step and retains the same value of HO for the next step.

(b) $T_n^* <$ EMIN : in this case MIMINF (5) accepts the integration step and sets HO = sgn (HO) min $\left[2\ |HO|\ ,\ |AMXER| \right]$

(c) $T_n^* \geq$ EMAX : if $|HO|$ = AMINER the integration step is accepted; otherwise MIMINF (5) rejects the integration step (setting MIMPAS = -1) and sets HO = sgn (HO) max $\left[1/2\ |HO|, |AMINER| \right]$

"Variable step" integration is an option (specified in the data by INDVAR = 1). If INDVAR is 0 then the "fixed step" option is used. For the fixed step, the calculations of MIMINF (5) as described above are not used; entry point 5 for fixed step is trivial; HO is set equal to DELTS.

MIMINF2 performs the calculations necessary to integrate the second vehicle array of variables. It is identical to MIMINF except for the use of Vehicle 2 COMMON blocks. It should be noted that MIMINF2 is called directly by EXE. Program EXE thus directly controls both vehicle equation of motion integrations. The indicator INDNUM controls the number of vehicles in EXE; INDNUM = 1 signifies only one vehicle is being employed, and INDNUM = 2 signifies two vehicles are being employed. A flow chart for MIMINF is presented.


START
ENTRY
YO(I) = Y(I)
CONTINUE
T = T + .5H
DO 348 I=1,NUM
XK(I,3) = H*P(I)
Y(I) = YO(I)+XK(I,3)
CONTINUE
RETURN
DO 450 I=1,NUM
Y(I) = YO(I)+XK(I,1) + H*P(I)+2.*(XK(I,2)+XK(I,3))/6.
YCC(I) = P(I)
CONTINUE
MIMPAS = 0
RETURN
J = MIMPAS
DO 220 I=1,NUM
H*P(I)
CONTINUE
RETURN
INDVAR = 0
TN = 0
DO 510 I=1,NUM
TNPT= H*(YDO(I)-YCC(I))
Y(I) >1.
TNPT= TNPT/|Y(I)|
TN = AMAX1(TN,TNPT)
CONTINUE
TN < EMAX
TN > EMIN
HO>AMAXER
RETURN

19. TIMID and TIMID2 - Step Function Routines for Time Points

Purpose:

To compute the step function $y = \tau(x)$ given by

$$y = \min \left\{ y \in Y \;\middle|\; y \geq x \right\}$$

where Y contains the following types of elements

1) all integral multiples of DELTS

2) $y_1 = \tau_1(x)$ where τ_1 is the step function evaluated by the TIM001 subroutine.

Usage:

CALL TIMID (XX,YY)

where

XX is the argument.

YY is the answer.

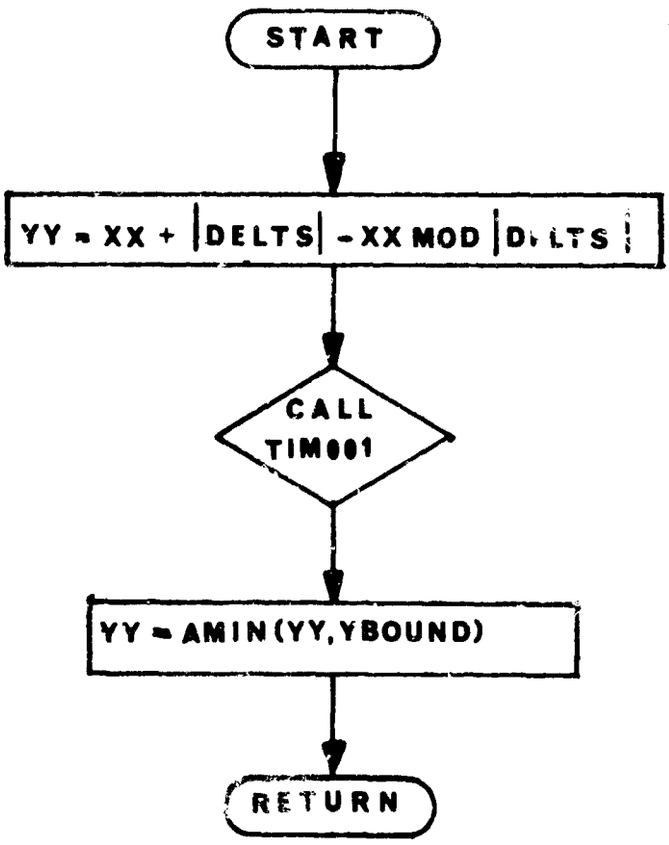
EXE is the only routine which calls TIMID; it does so to compute the next stage time point that must be hit as a function of the current stage time. In order that TIMID not return a stage time point which is "too close" to the current stage time, EXE calls TIMID as follows:

CALL TIMID (TIMES + AMINER, TIMPT)

Remarks:

TIMID2 computes $y = \tau(x)$ for the second vehicle. TIMID2 is identical to TIMID except for the use of vehicle 2's COMMON blocks and auxiliary subroutines. A flow chart for TIMID is presented.

TIMID


START
YY = XX + |DELTS| - XX MOD |DELTS|
CALL TIM001
YY = AMIN(YY, YBOUND)
RETURN

20. VALUES and VALUES2 - Value Print Routines

Purpose

To print the values of floating point variables.

Usage

CALL VALUES (L, X1,...,XL)

L — is an integer $0 \leq L \leq 8$ identifying the number of arguments following it.

X1, ., ., ., XL — L cells each containing the value of some floating point variable to be printed.

The above call adds the L variable values, X1,...,XL, to the list of values to be output on the next line of print. When 8 values have been accumulated the line is printed, any excess values are added to the list for the next line of print. If L is 0, then the values in the existing list (the number may be less than eight) are printed immediately.

Lines accounting is taken care of within this routine.

This subroutine is designed to be used in conjunction with the CODES subroutine.

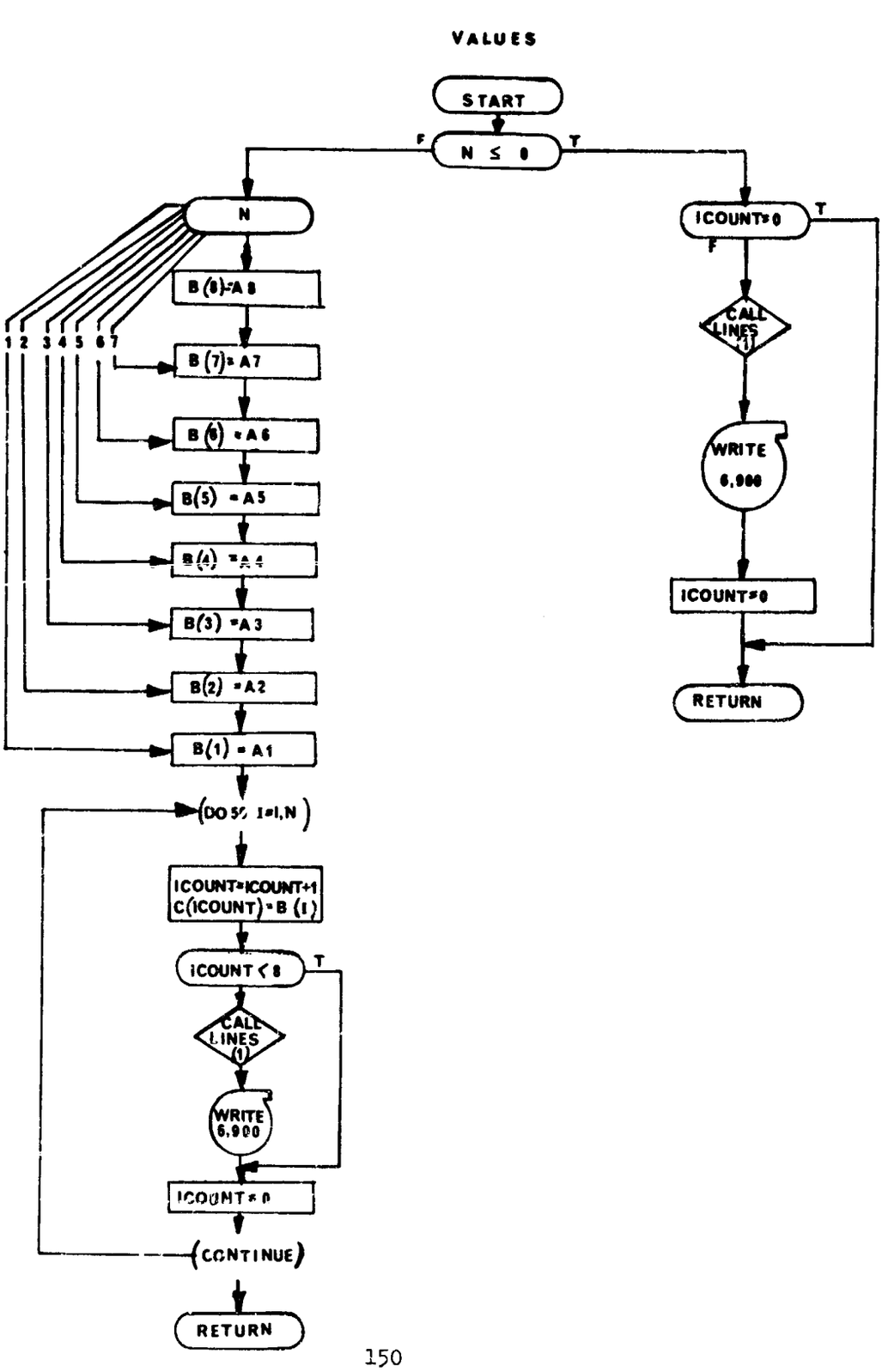
Values are printed with a "1PE15.7" format.

CODES and VALUES control small secondary output buffers within the program itself.

Normally a call to CODES is made before any call to VALUES in order to identify the values. The CALL CODES (0) is necessary to be sure that the codes are printed; the CALL VALUES (0) is necessary to be sure the values are printed. (If this is not done the buffers may never be flushed.)

Remarks:

A flow chart for VALUES is presented. VALUES2 is identical but is required for correct output function for the second vehicle.


VALUES
START
N ≤ 0
F
T
N
1 2 3 4 5 6 7
B(8)=A 8
B(7)=A7
B(6) =A6
B(5) =A5
B(4) =A 4
B(3) =A3
B(2) =A2
B(1) = A1
(DO 5% I=1,N)
ICOUNT=ICOUNT+1
C(ICOUNT)=B (I)
ICOUNT < 8
T
CALL LINES (1)
WRITE 6,900
ICOUNT=0
(CONTINUE)
RETURN
ICOUNT>0
T
F
CALL LINES (1)
WRITE 6,900
ICOUNT=0
RETURN

21. MISCUT and MISCUT2 - Abortion Routine

Purpose:

To terminate the trajectory if any specified variable lies outside a specified upper or lower bound for that variable.

Usage:

CALL MISCUT

If a specified variable lies outside a specified upper or lower bound, TIME is set to 1.E36. TMAX should not be greater than 1.E36 when use of this routine is specified. Thus the trajectory will terminate on MISSED CUTOFF.

Up to nine variables may be specified for bounding on the MISVAR card. Values for bounding the variables appear in the corresponding positions on the BOUNDS card. Testing of the boundaries is determined by the LUBGLB card.

The integers appearing on the LUBGLB card are of the form

ij

j specifies that the jth variable on the MISVAR card will be checked.

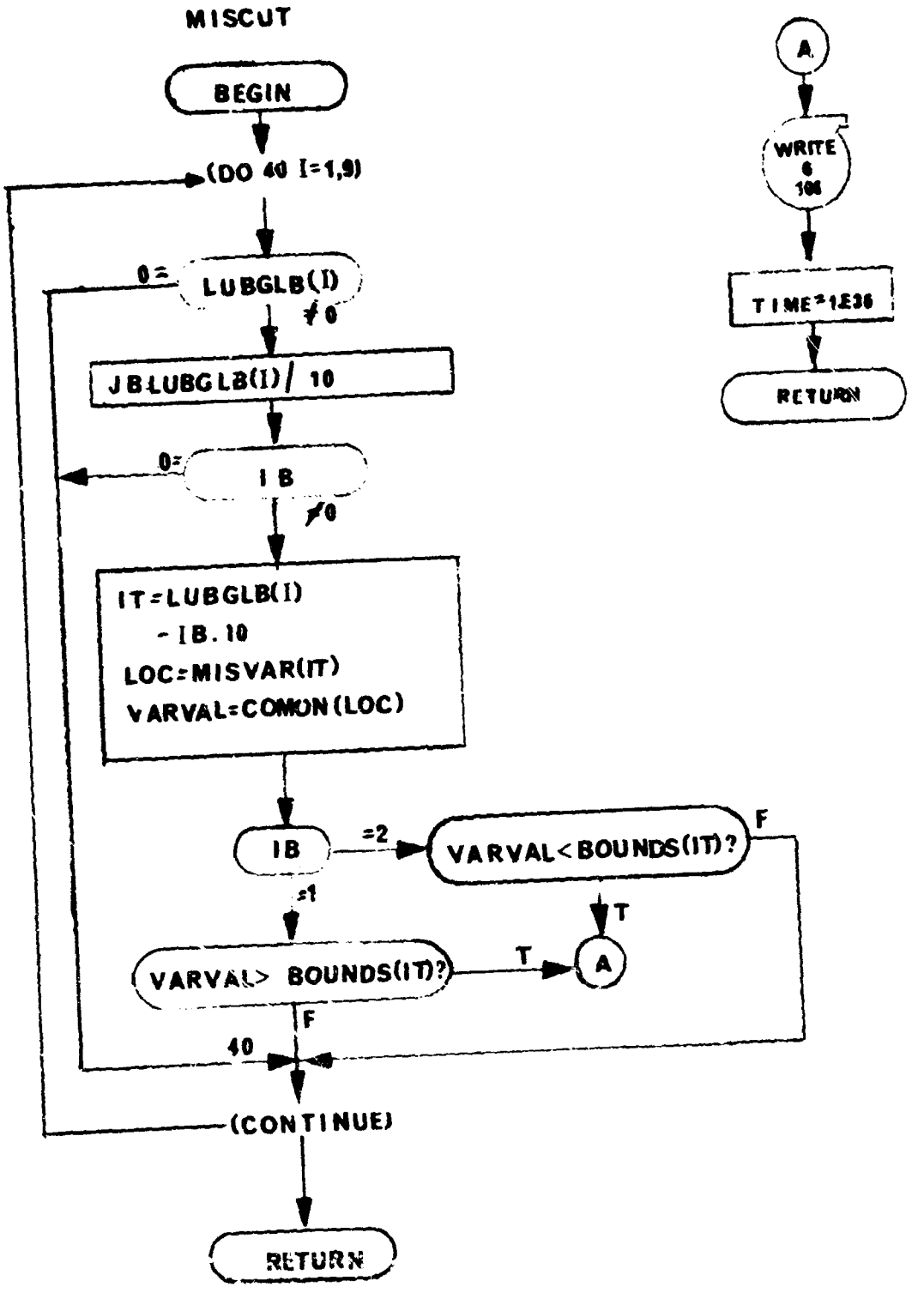
When i = 0 the jth value on the BOUNDS card will be ignored and no check made.

When i = 1 the jth value on the BOUNDS card will be an upper bound for the jth variable on the MISVAR card.

When i = 2 the jth value on the BOUNDS card will be a lower bound for the jth variable on the MISVAR card.

Remarks:

MISCUT is called by EXE. MISCUT2 terminates the trajectory if any specified second vehicle lies outside specified upper and lower bounds. A flow chart for MISCUT is presented; MISCUT2 is identical except for use of second vehicle COMMON blocks.


MISCUT
BEGIN
(DO 40 I=1,9)
0=
LUBGLB(I)
≠0
JB=LUBGLB(I)/ 10
0=
IB
≠0
IT=LUBGLB(I)
-IB.10
LOC=MISVAR(IT)
VARVAL=COMON(LOC)
IB
=2
VARVAL<BOUNDS(IT)?
F
=1
T
VARVAL> BOUNDS(IT)?
T
A
F
40
(CONTINUE)
RETURN
A
WRITE
6
108
TIME=1E36
RETURN

22. ONLINED - On Line Display Routine

Purpose:

To display a thirty-nine character message on the CDC display station of a CDC 200 user terminal.

Method:

The system routine *encode* is used to construct the message line, and the system routine *scope* is used to display the message on the CRT device.

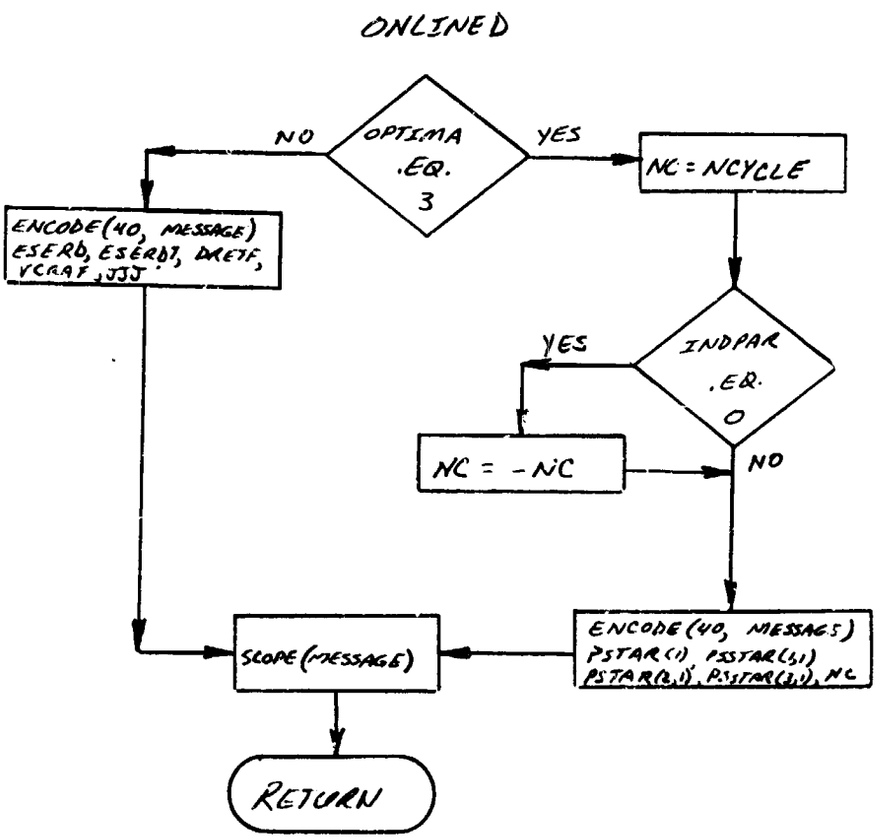
Usage:

Entry is made to this routine with the following statement:

CALL ONLINED

Subroutines Called:

ENCODE
SCOPE



23. DIFEQ1 and DIFEQ2 - Point Mass Equations of Motion

Purpose:

For performing all calculations necessary for the three-degree of freedom trajectory equations.

Method:

DIFEQ1 has several subprograms; in general the subprograms are responsible for performing the calculations made necessary by certain physical attributes of the vehicle, the trajectory or the environment. DIFEQ1 may be called from EXE, EXTRAN, and PTBEQN (by first passing through the interface routine DIFEQ).

Subprogram

HETS	**Temperature and heating parameters**
TFFS	**Thrust force; mass flow**
FPPS **FPPG**	**Control variables as functions of already computed variables**
SACS	**Wind axes aerodynamic forces on vehicle**
ATMS	**Atmospheric parameters**
GVSP	**Local geocentric gravitational force components**
LATS	**Conversion from geodetic to geocentric latitude and vice-versa**

In addition to the vehicle and planetary characteristics subprograms above, DIFEQ1 is broken down into a sequence of control subroutines.

Subroutine:

DEQPRE	Used at Entry 1 from EXE
DEQINI	Used at Entry 2 from EXE
DEQBCI	Used at Entry 3 from EXE
DEQACI	Used at Entry 3 from EXE
DEQSPI	Used at Entry 4 from EXE
DEQVAL	Used at Entry 5 from EXE
DEQCOD	Used at Entry 6 from EXE
DEQIV	Used at Entry 7 from EXE
DEQHT	Used at Entry 8 from EXE
FIRFUN	Control computation of fire control functions

GAM91	Controls flight through the vertical
CTLITR	Controls instantaneous control variable iteration process.

With DIFEQ1 itself there is a considerable amount of code that may be by-passed depending on the type of problem being considered. Many of the calculations done at DIFEQ1 (3) would be purely auxiliary. Therefore for calculations of this type, there are indicators which may be input = 0 if it is desired that a particular calculation be performed. Since DIFEQ1 (3) has to be called so many times during a trajectory, any deletions of calculations which are not essential will help to cut down on the machine time.

A general description will be given of each entry point associated with DIFEQ1. Detailed information can be obtained in the user's manual for each entry point used by DIFEQ1.

DIFEQ1 (1)

At this entry point nominal values of indicators are set and the values of the integrated variables are initialized at 0. Also, nominal values are set for the standard constants used in the equations. (e.g. polar and equatorial radius of the earth). See DEQPRE.

DIFEQ1 (2)

This is the initial transformation. It is always performed at the beginning of a trajectory; it may be performed at the beginning of a major stage (see EXTRAN). Also, it may be used in certain combinations for the h-transformation (see EXTRAN). See DEQINI.

DIFEQ1 (3)

The main body of the calculations in DIFEQ1 appear at this entry point. The primary purpose at this entry point is to compute the derivatives of the variables that are being integrated. See DEQBCI, DEQACI, and CTLITR.

DIFEQ1 (4)

At this entry point the variables computed at either the initial transformation or for the h-transformation are printed. See DEQSPI.

DIFEQ1 (5)

At this entry point, the codes are printed, the codes identify the variables printed at DIFEQ1 (4) and DIFEQ1 (6). See DEQUAL

DIFEQ1 (6)

At this entry point the values of the variables computed at DIFEQ1 (3) are printed. Also, the values of the state variables (and perhaps other integrated variables) are printed. Entry is made to DIFEQ1 (6) only at valid integration steps. In some cases the value of a variable is printed only if the corresponding indicator is set in the data. See DEQCOD.

DIFEQ1 (7)

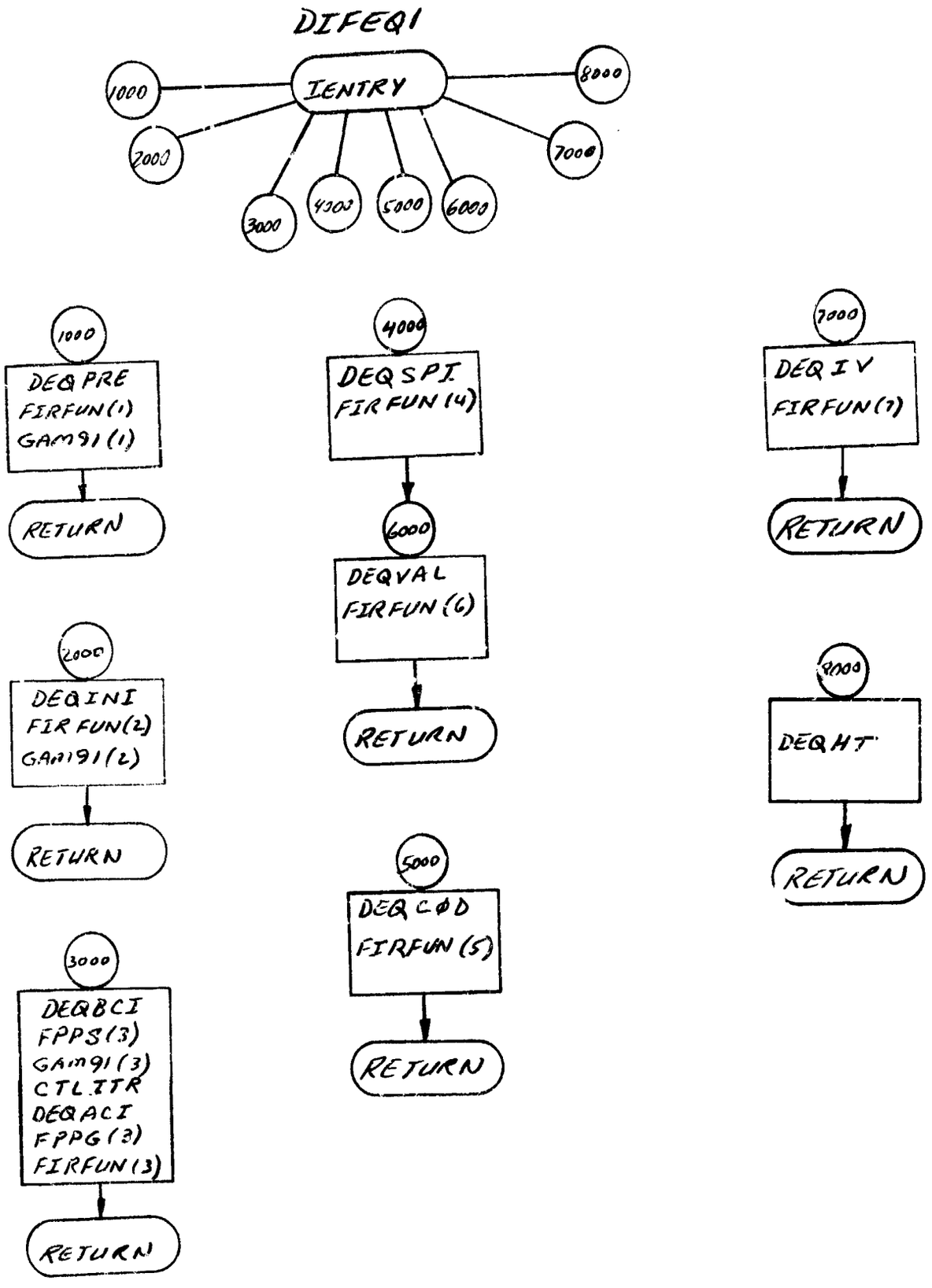
At this entry point the variables to be integrated are defined by making a call to INTGRT for each variable to be integrated. In some instances this will be done only if the appropriate indicator is set. See DEQIV.

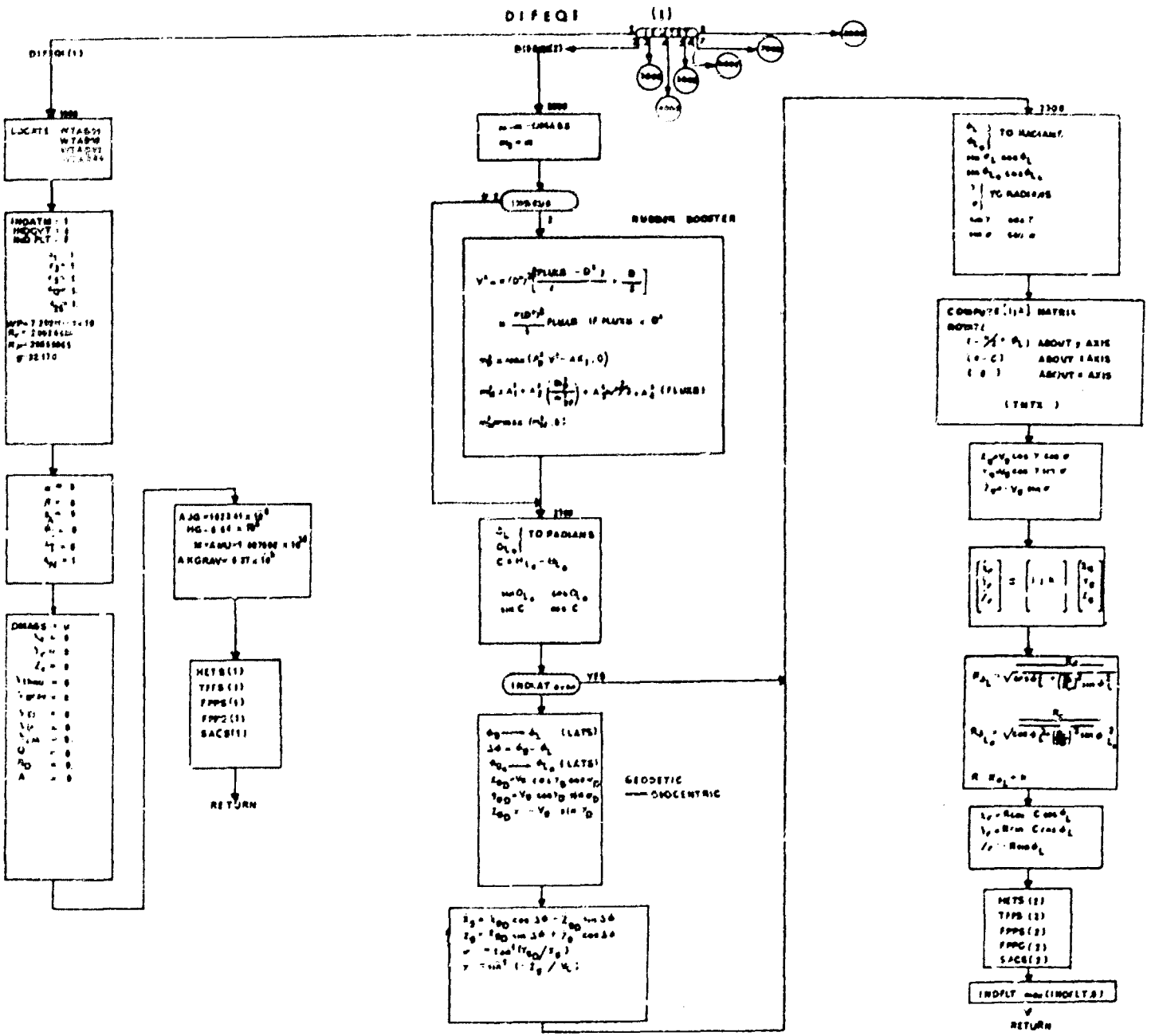
DIFEQ1 (8)

At DIFEQ1 (8) an h-transformation is made. This transformation may be performed at a stage point (see EXTRAN). See DEQHT.

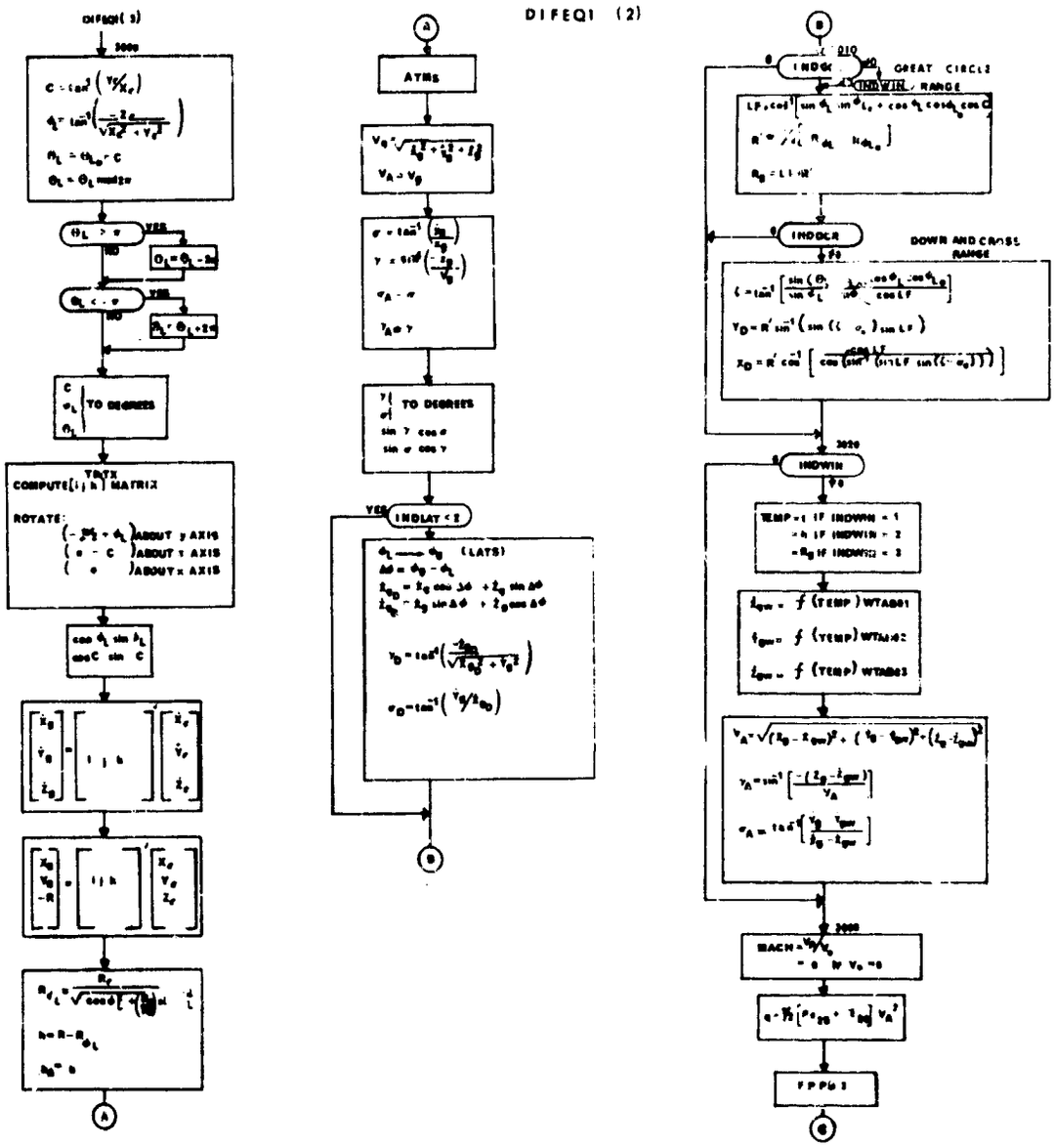
Remarks:

Subroutine DIFEQ2 is identical in form to DIFEQ1. DIFEQ2 controls the vehicle 2 equation of motion calculations using appropriate COMMON blocks and auxiliary subroutines. A flow chart of DIFEQ1 is presented. For user convenience, the original single vehicle equation of motion routine, DIFEQ1, is also presented.

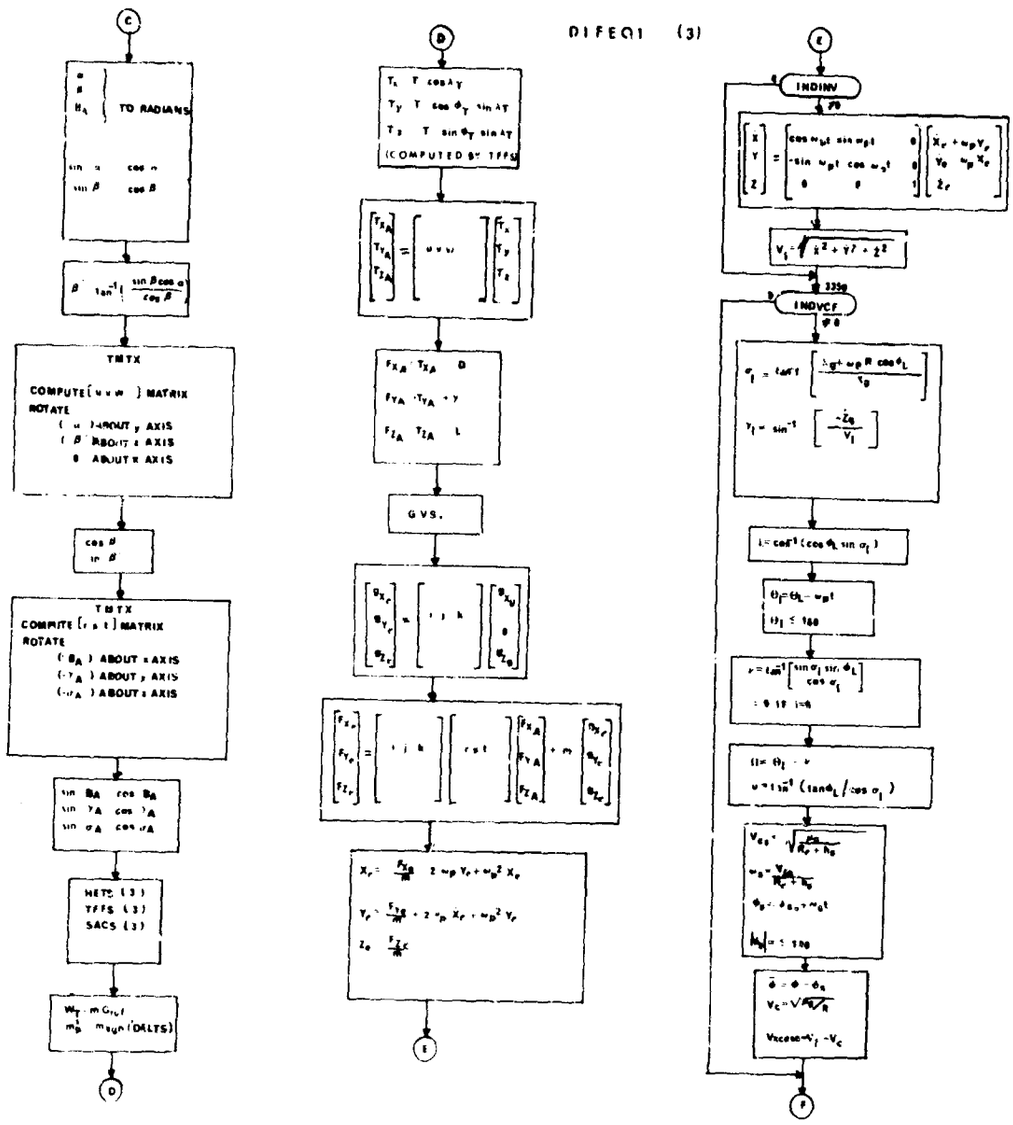

DIFEQI
IENTRY
1000
2000
3000
4000
5000
6000
7000
8000
1000
DEQPRE
FIRFUN(1)
GAM91(1)
RETURN
2000
DEQINI
FIRFUN(2)
GAM91(2)
RETURN
3000
DEQBCI
FPPS(3)
GAM91(3)
CTLITR
DEQACI
FPPG(3)
FIRFUN(3)
RETURN
4000
DEQSPI
FIRFUN(4)
6000
DEQVAL
FIRFUN(6)
RETURN
5000
DEQCOD
FIRFUN(5)
RETURN
7000
DEQIV
FIRFUN(7)
RETURN
8000
DEQHT
RETURN



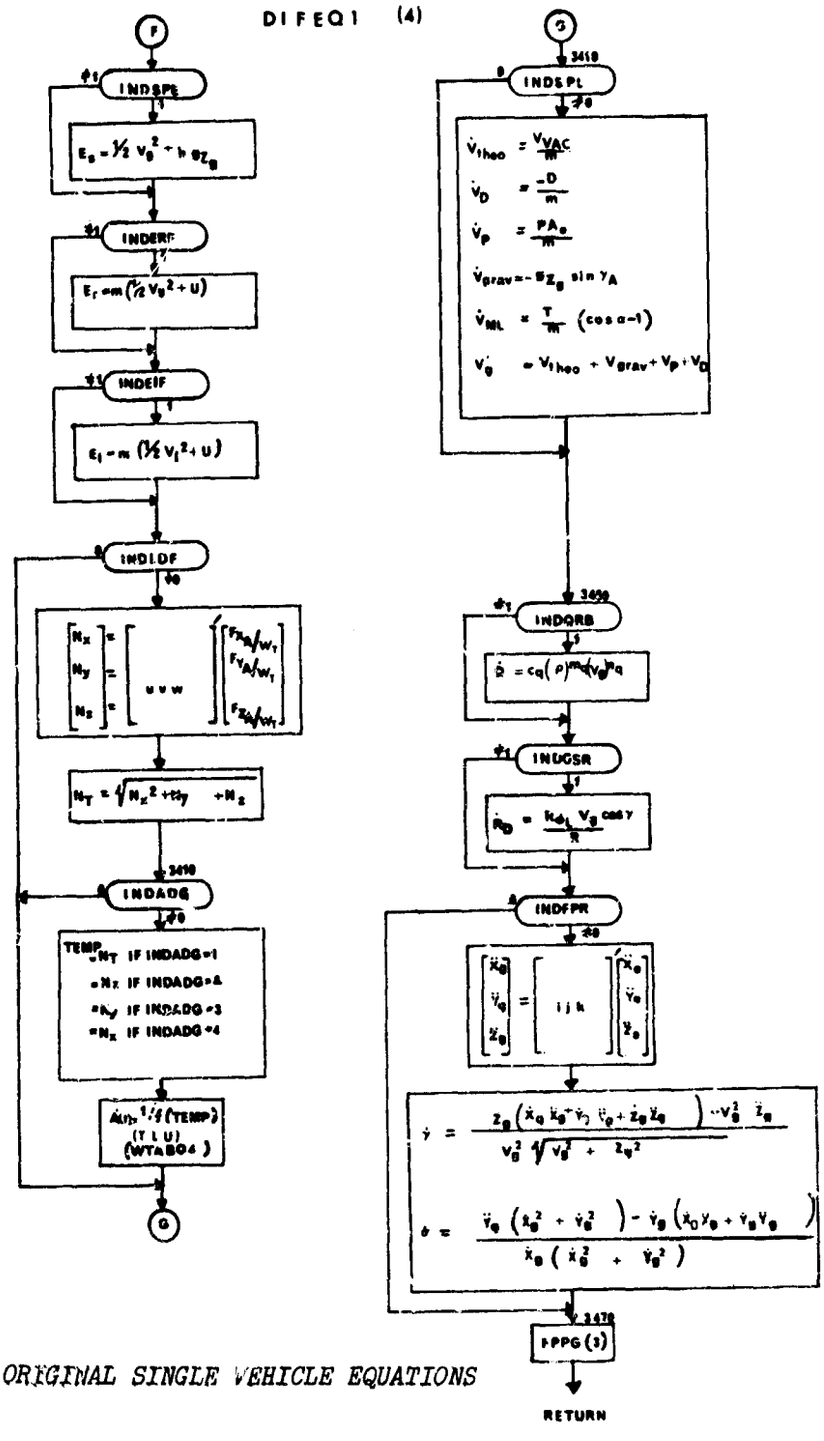
ORIGINAL SINGLE VEHICLE EQUATIONS



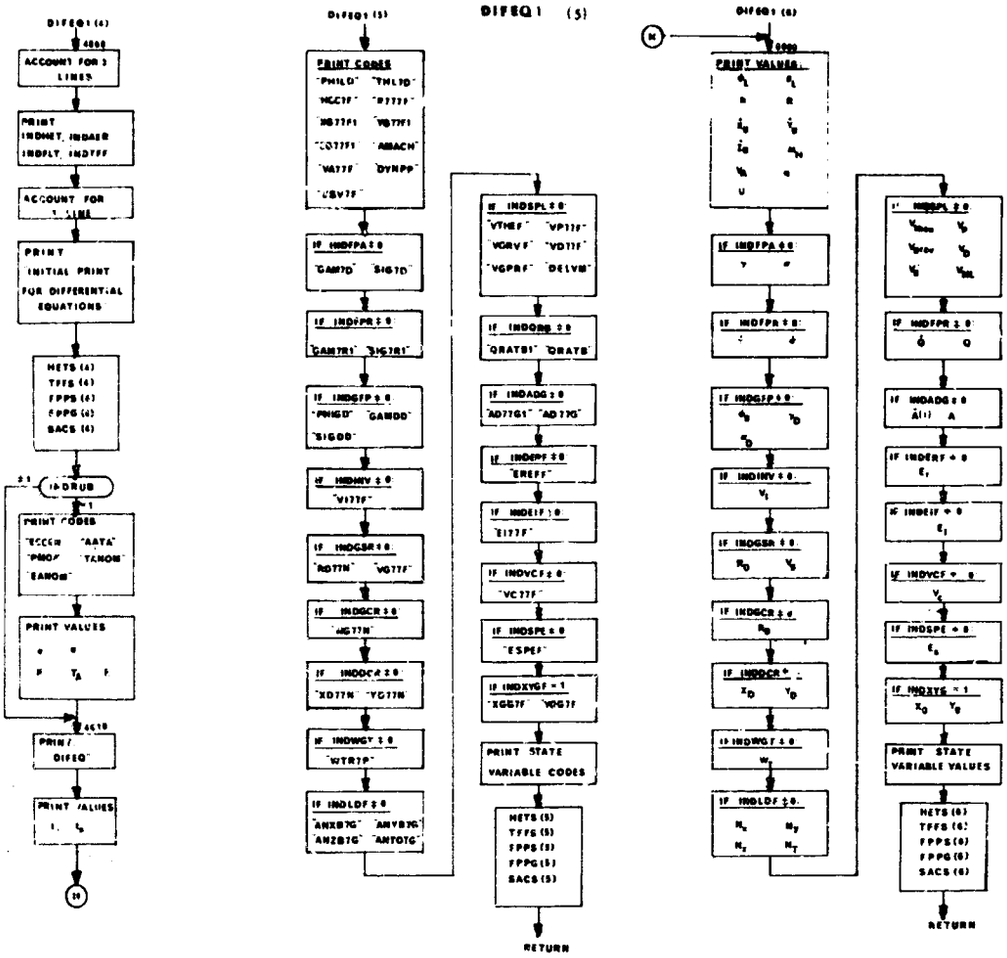
ORIGINAL SINGLE VEHICLE EQUATIONS



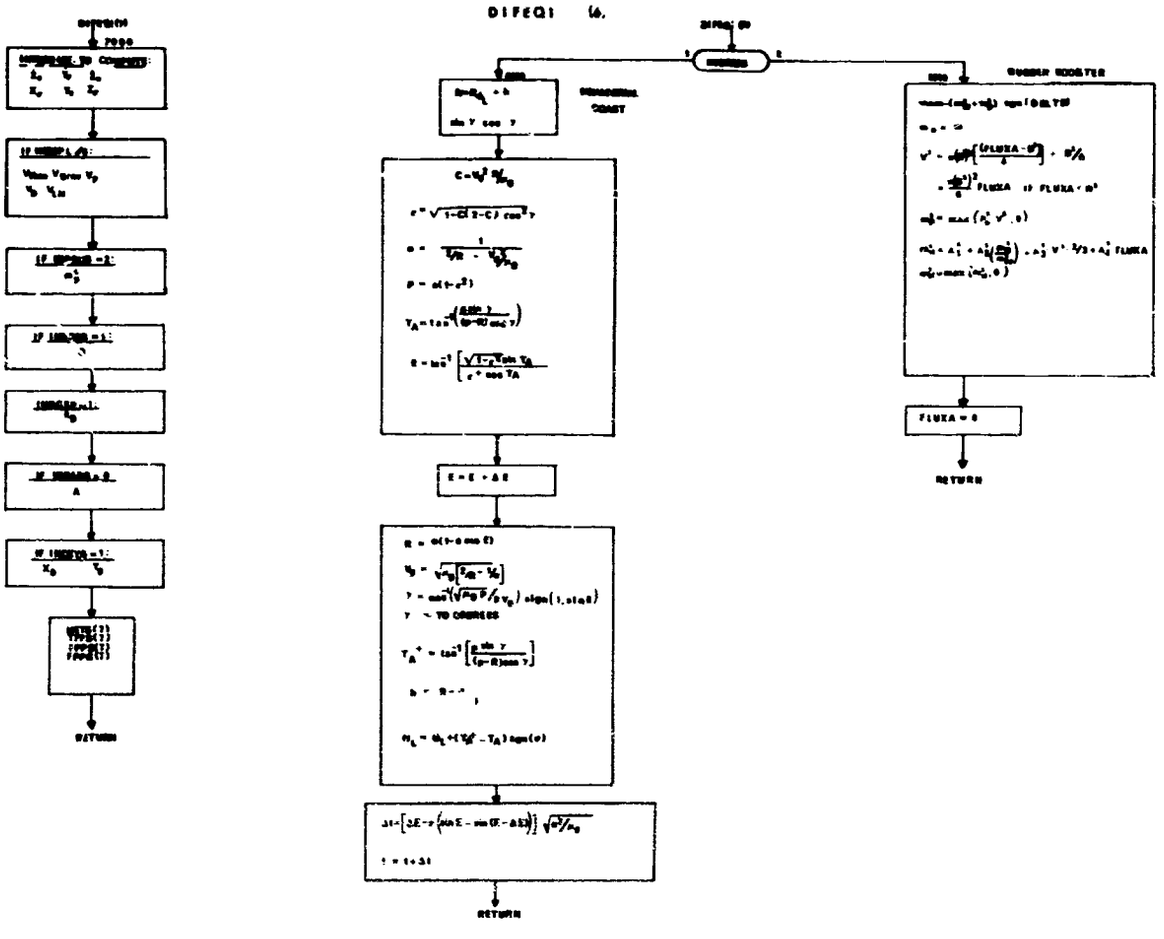
ORIGINAL SINGLE VEHICLE EQUATIONS



ORIGINAL SINGLE VEHICLE EQUATIONS



ORIGINAL SINGLE VEHICLE EQUATIONS



ORIGINAL SINGLE VEHICLE EQUATIONS

24. DIFEQ3 - Dummy Subroutine

DIFEQ4 - Dummy Subroutine

DIFEQ5 - Sample Arbitrary Differential Equations

This is intended only as an example of how a new set of differential equations may be programmed for use in conjunction with the optimization program. The equations of this example are in cylindrical coordinates and describe a two-dimensional point mass with constant thrust in a vacuum having direction of thrust as a control variable. No flow chart is included with this subroutine.

DIFEQ6 - Dummy Subroutine

25. TLUREV - Two Dimensional Table Look-up Routine (Special)

Purpose

To linearly interpolate in a table which has just two points.

Usage

CALL TLUREV (X, C, Y, IER)

where

X is the argument

Y is the functional value

IER is set to 1 if extrapolation was necessary

C is the array containing the points of the table according to the following format

C(1) = Not used

C(2) = X_1

C(3) = Y_1

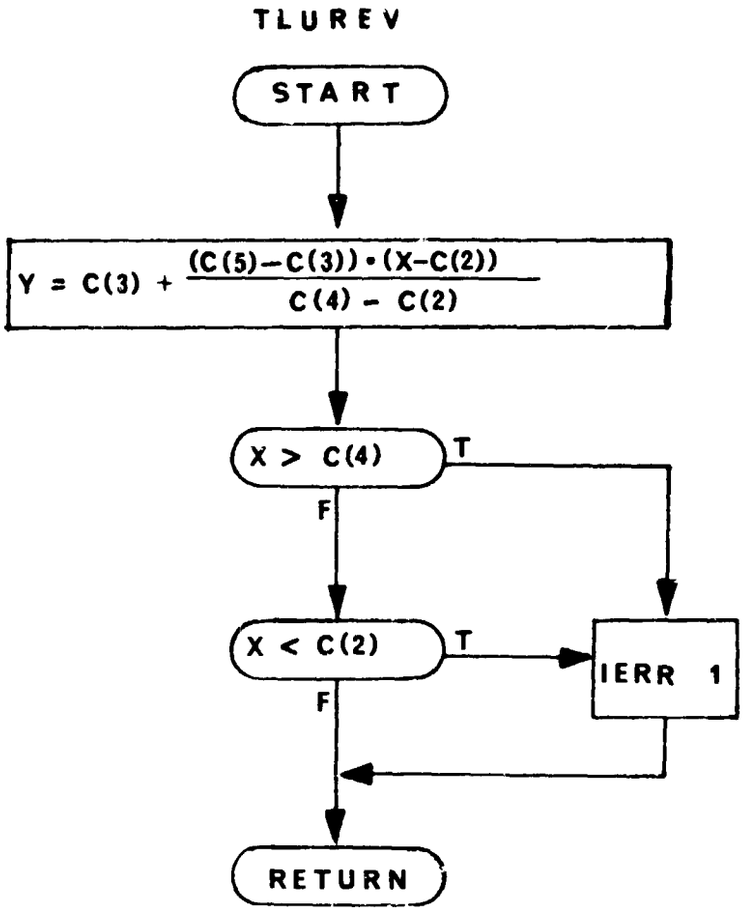
C(4) = X_2

C(5) = Y_2

It is assumed that $X_1 < X_2$

Remarks

MANTGT and REV are the only routines which use this routine.


TLUREV
START
Y = C(3) + (C(5)−C(3))•(X−C(2)) / C(4) − C(2)
X > C(4)
T
F
X < C(2)
T
IERR 1
F
RETURN

26. ACOS - Arc Cosine Routine

Purpose

To compute the arc cosine of a normalized floating point argument X.

Method

For $|X| < 7.4505806 \times 10^{-9}$ the arc cosine is set equal to $\pi/2$. For $x = 1.$ arc cosine is set equal to zero and for $X = -1.$ arc cosine is set equal to π. When the argument $X \neq \pm 1.$ the routine gives the arc cosine in radians from 0 to π.

Usage

The arc cosine is computed using the statement

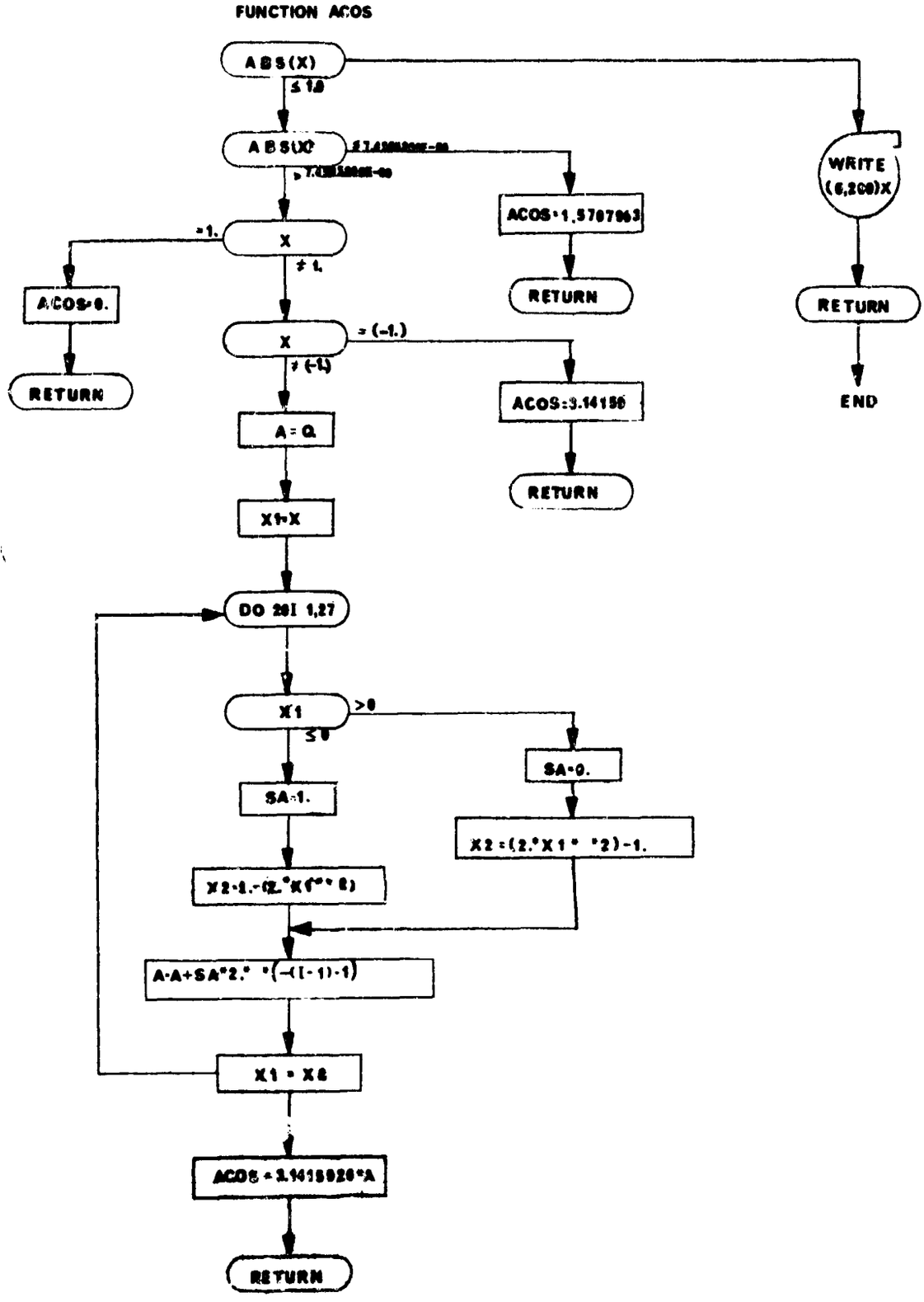
$$Y = ACOS(X)$$

where, $|X| \leq 1.$ and $Y = Cos^{-1}X$.

Remarks

No error returns are provided.

FUNCTION ACOS



27. ASIN - Arc Sine Routine

Purpose

To compute the arc sine of a normalized floating point argument X.

Usage

The arc sine is computed using the statement

ASIN = 1.5707963 - ACOS(X)

Y = ASIN(X)

where $|X| \leq 1.$ and $Y = \sin^{-1} X$

Remarks

No error returns are provided

28. TLU and TLU2 - Two-Dimensional Table Look Up Routine

Purpose

Given an argument X, to compute Y = f(X) from a table of X and Y values by linear interpolation.

Method

The table of X values is searched until for some i, $X_i < X < X_{i+1}$. Linear interpolation is then performed. If, for some i, $X = X_i$ then Y is set to Y_i.

Usage

Entry is made via the statement,

```
CALL TLU (X,C,Y,IND)
```

where, X = Variable name of the argument.

C = Array name of the curve being used.

Y = Variable name of the interpolated value.

IND = 0, no errors indicated.
1, limit of the curve has been exceeded and the result is an extrapolation using the last two points.

The curve C must be stored as follows:

C(1) = N = Number of points in curve (fixed point integer)

$C(2) = X_1$

$C(3) = Y_1$

$C(4) = X_2$

$C(5) = Y_2$

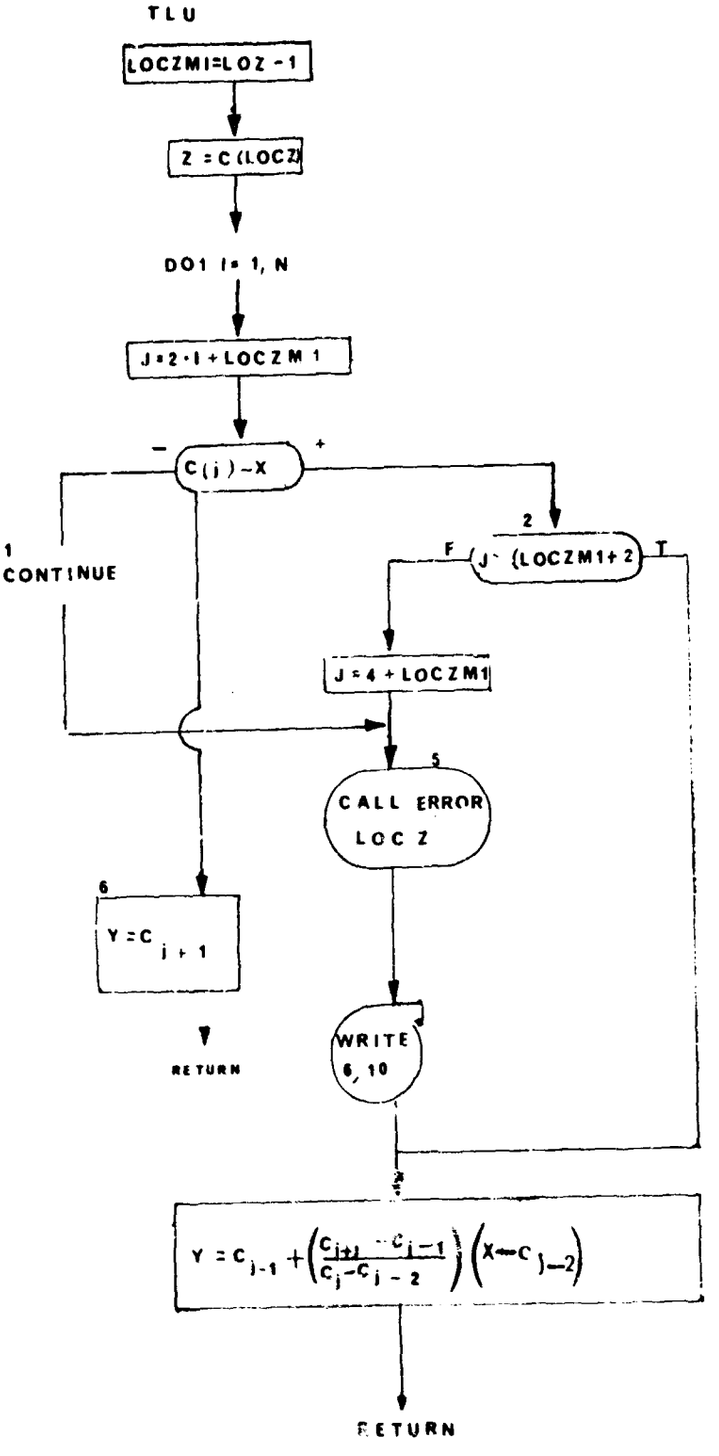
.
.
.

$C(2N) = X_N$

$C(2N+1) = Y_N$

Remarks:

A flow chart for TLU is presented. TLU2 is identical to TLU except for use of vehicle 2 COMMON blocks.


TLU
LOCZM1=LOZ-1
Z=C(LOCZ)
DO1 I=1, N
J=2*I+LOCZM1
C(j)-X
−
+
1
CONTINUE
2
F
J=(LOCZM1+2)
T
J=4+LOCZM1
5
CALL ERROR
LOC Z
6
Y=C j+1
RETURN
WRITE
6, 10
Y = C j-1 + ((C j+1 - C j-1)/(C j - C j-2)) (X - C j-2)
RETURN

29. ATAN2 - Arctangent Routine

Purpose

To compute the arctangent of the quotient of two normalized floating point quantities, A/B, with proper quadrant control.

Method

The routine computes the quotient X/Y. The arctangent is computed with quadrant according to the sign of Y and X. If $X = 0$ and $Y \neq 0$, the routine computes

$$Y = \tan^{-1}(Y/X) = \frac{Y}{|Y|} \cdot \frac{\pi}{2}$$

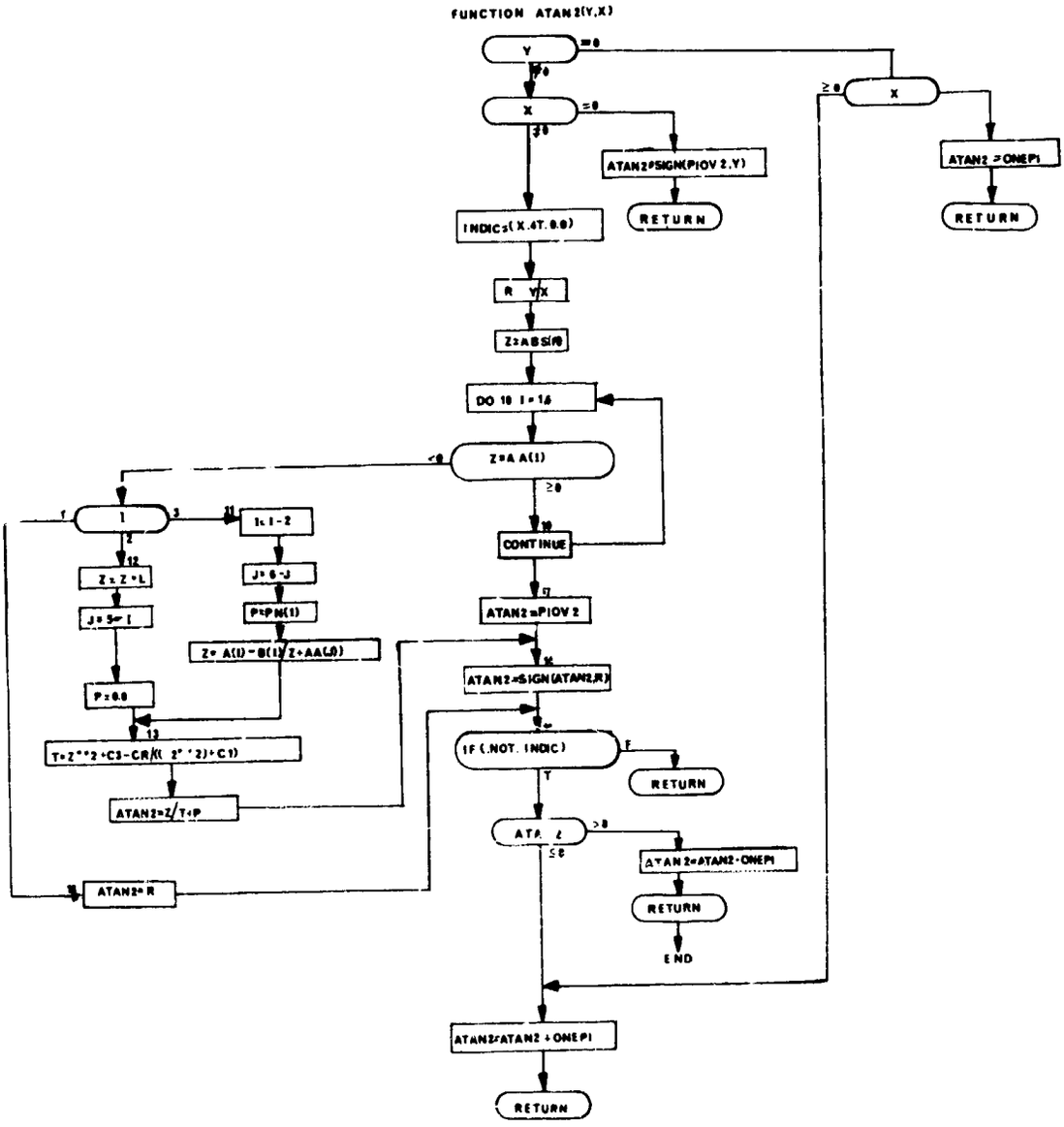
If $X = 0$ and $Y = 0$, it computes

$$Y = \tan^{-1}(Y/X) = 0$$

Usage

The arctangent of Y/X is obtained via,

$$Y = \text{ATAN 2 } (Y,X)$$


FUNCTION ATAN2(Y,X)
Y
X
ATAN2=SIGN(PIOV2,Y)
RETURN
ATAN2=ONEPI
RETURN
INDIC=(X.4T.0.0)
R Y/X
Z=ABS(R)
DO 10 I=1,6
Z=AA(I)
CONTINUE
ATAN2=PIOV2
ATAN2=SIGN(ATAN2,R)
IF (.NOT. INDIC)
RETURN
ATAN2=ATAN2-ONEPI
RETURN
END
ATAN2=ATAN2+ONEPI
RETURN
ATAN2=R
P=0.0
ATAN2=Z/T+P

30 . TLU1 - Two - Dimensional Table Look-up Routine

Purpose

Given an argument X, to compute $Y_i = f(X)$ by linear interpolation, where $i \geq 1$.

Method

The table of X values is searched until for some i, $X_i < X < X_{i+1}$. Linear interpolation is then performed on the number of curves requested. If for some i, $X = X_i$, then the Y_i are set to that corresponding value of the dependent variable.

Usage

Entry is made via the statement.

```
CALL TLU1 (N, TAPLE,TIME,NDEP,ANS,IND)
```

where,

N = The number of points in each curve.
TABLE = The array name containing the sets of curves.
TIME = The variable name of the argument.
NDEP = The number of curves contained in the array CURVE.
ANS = The array name into which the NDEP answers are to be stored.
IND = 0, no errors indicated and all interpolations seem to be good.
1, the argument X lies outside the range of the independent variable. If less than the range, the first points were used. If greater, the last points were used.

The table "CURVE" must have been set up as follows:

CURVE(1) = X_1

CURVE(2) = X_2

CURVE(3) = X_3

.
.
.

CURVE(N) = X_N

CURVE(N+1) = Y_{1_1}

CURVE(N+2) = Y_{1_2}

CURVE(N+3) = Y_{1_3}

.
.
.
.

CURVE(2N) = Y_{1_N}

CURVE(2N+1) = Y_{2_1}

CURVE(2N+2) = Y_{2_2}

CURVE(2N+3) = Y_{2_3}

.

.

.

CURVE(3N) = Y_{2_N}

CURVE(3N+1) = Y_{3_1}

.

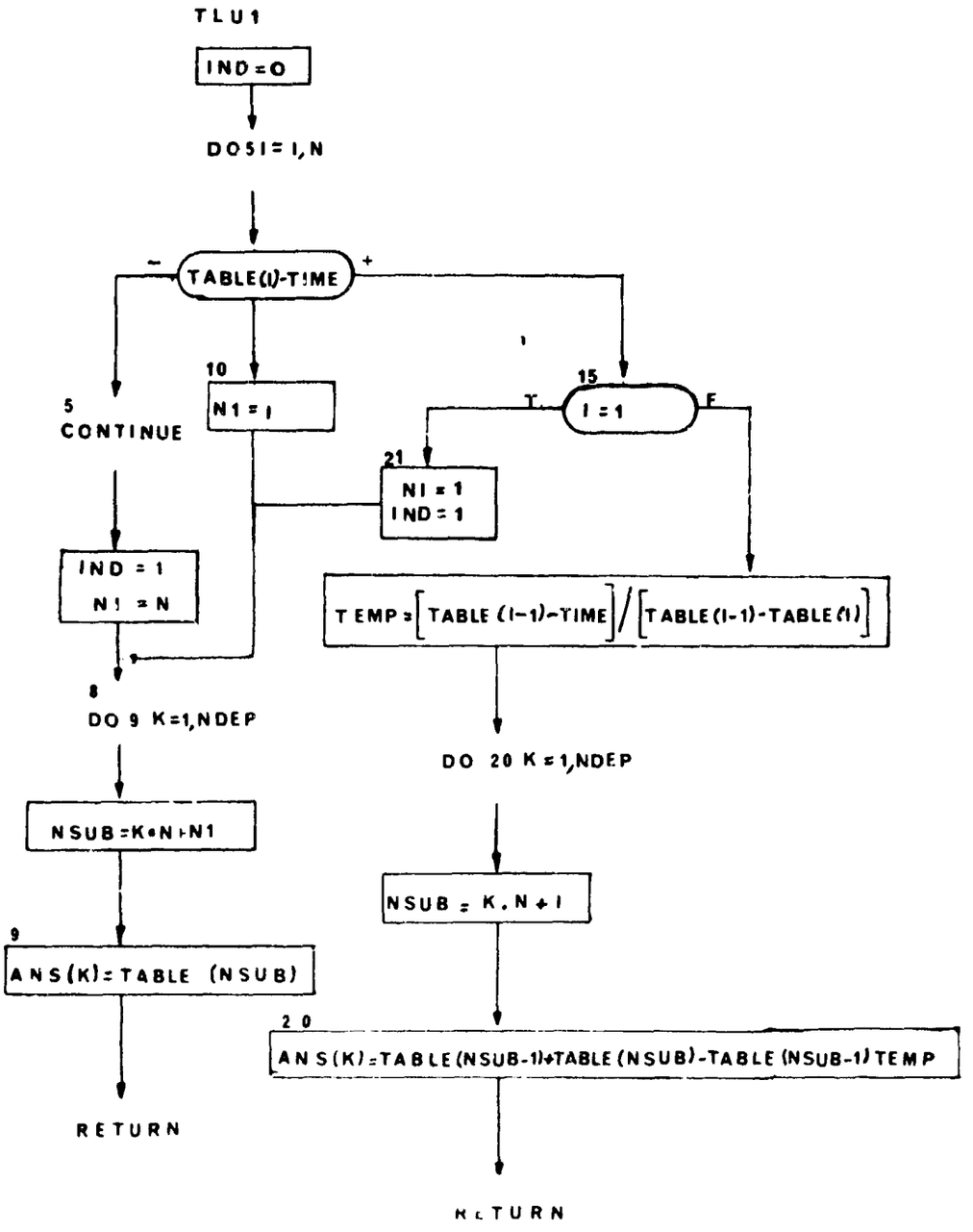
.

.

CURVE((NDEP+1)*N) = Y_{NDEP_N}

Remarks

No other routines are called from this routine.


TLU1
IND=0
DO 5 I=1,N
TABLE(I)-TIME
-
+
10
N1 = I
5
CONTINUE
15
I = 1
T
F
21
N1 = 1
IND = 1
IND = 1
N1 = N
TEMP = [TABLE (I-1)-TIME] / [TABLE(I-1)-TABLE(I)]
8
DO 9 K=1,NDEP
DO 20 K = 1,NDEP
NSUB=K*N+N1
NSUB = K.N + I
9
ANS(K)=TABLE (NSUB)
20
ANS(K)=TABLE(NSUB-1)+TABLE(NSUB)-TABLE(NSUB-1)*TEMP
RETURN
RETURN

31. TIMREV and TIMREV2 - Time Point Collection Routine

Purpose:

During the forward trajectory, to build an array of stage time points that must be "hit" during the corresponding stage of the reverse trajectory.

Usage:

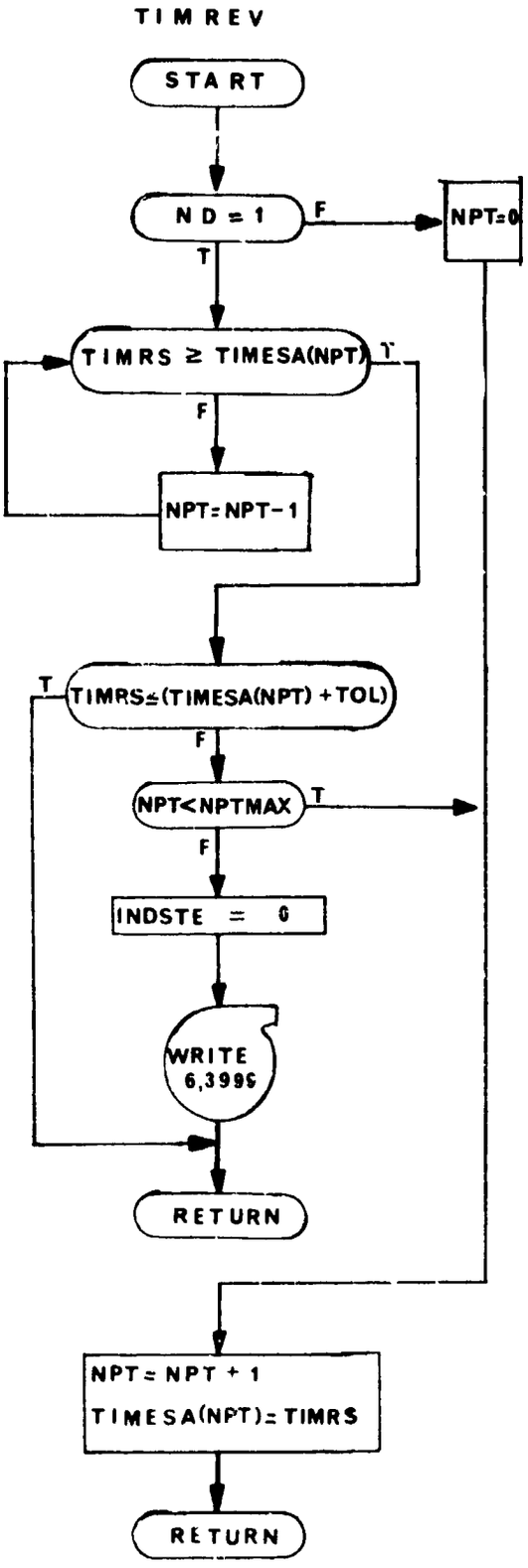
CALL TIMREV (TIMRS, TOL)

TIMRS	the stage time point that is to be added to the list of points that must be hit in the corresponding stage of the reverse trajectory.
TOL	a tolerance: if a previous point of the list differs from the current point to be added by less than TOL, the current point is not added.

The name of the array of points is TIMESA. The TIMESA array is built in monotonically increasing order. If for any reason TIMRS is less than a point in the TIMESA array, all points greater than TIMRS in the TIMESA array are annihilated. NPT is the current number of points in the array. NPT must not exceed the dimension of the TIMESA array; if it does the case will be terminated. The first point of a stage will always be inserted in the TIMESA array.

This routine was written in anticipation of the need to hit stage time points in reverse for a number of different reasons. In particular, the routines PARTS and CTVS call TIMREV. It is imperative that PARTS (3) call TIMREV with the current stage time and a tolerance of 0.

TIMREV2 builds the array of stage time points for the second vehicle reverse trajectory. It should be noted that all variational optimization problems must be phrased in terms of the first vehicle data. In cooperative variational optimization, a program modification setting second vehicle control variables to names in the first vehicle directory is required. A flow chart for TIMREV is presented. TIMREV2 is identical to TIMREV except for use of the second vehicle's COMMON blocks.


TIMREV
START
ND = 1
F
NPT=0
T
TIMRS ≥ TIMESA(NPT)
T
F
NPT=NPT-1
T
TIMRS≤(TIMESA(NPT) +TOL)
F
NPT<NPTMAX
T
F
INDSTE = 0
WRITE
6,3996
RETURN
NPT= NPT + 1
TIMESA(NPT)= TIMRS
RETURN

32. PSUBR-Evaluation of Partial Routine

Purpose:

To calculate matrices of partial derivatives numerically.

Usage:

This is a general routine; it is used in the following manner:

suppose $y_1 = f_1(x_1, \ldots, x_n)$

$\vdots$

$y_m = f_m(x_1, \ldots, x_n)$

and we want to compute

$$J = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_1} \\ \vdots & & \vdots \\ \frac{\partial y_m}{\partial x_1} & & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

then a call to PSUBR should be made as follows:

```
CALL PSUBR (N1,LX,LXST,HB,N2,LY,ANS,JP,P,LFCNS)
```

where N1 is n

LX is an array containing the COMMON locations of the variables $x_1, \ldots, x_n$.

LXST is an array containing the COMMON locations of same variables x_1*, ..., x_n* to be used in determining perturbation sizes.

HB is an array of the minimum perturbation sizes to be used for the respective variables $x_1, \ldots, x_n$.

N2 is m

LY is an array of the COMMON locations of the variable $y_1, \ldots, y_m$.

ANS is J, the array in which the answer is to be stored (this must be dimensioned 15 x 15)

JP is an array of indicators one for each variable $x_1, \ldots, x_m$. $JP_i = 0$ means that the i'th column of J is known to be 0.

P is a constant used to determine the perturbation size.

LFCNS is an indicator used to direct PSUBR to the code for the functions $f_1, \ldots, f_m$.

Method:

The elements of J are computed a column at a time by the following approximation

$$\frac{\partial y_i}{\partial x_j} = \frac{f_i(x_1, \ldots, x_j + h_j, \ldots, x_n) - f_i(x_1, \ldots, x_j - h_j, \ldots, x_n)}{2h_j}$$

where $h_j = \max (HB_j, |x_j^*| \ 10^{-P})$

If $JP_j = 0$ then $\frac{\partial y_i}{\partial x_j}$ is set = 0 for i = 1, ..., m. This eliminates the need for calling the routine to calculate the f functions for column j.

The variables in PSUBR are dimensioned to compute a 15 x 15 matrix of partials; PSUBR stores the answer in ANS which it expects to be dimensioned 15 x 15.

After ANS has been computed and before returning PSUBR makes a final call to compute the f functions with the original values of the x's. This is intended to insure that values of variables after computing partials are the same as the value before computing partials.

Remarks

(1) PSUBR is called only by PARTS.

(2) P is PEXN in all calls; PEXN is input.

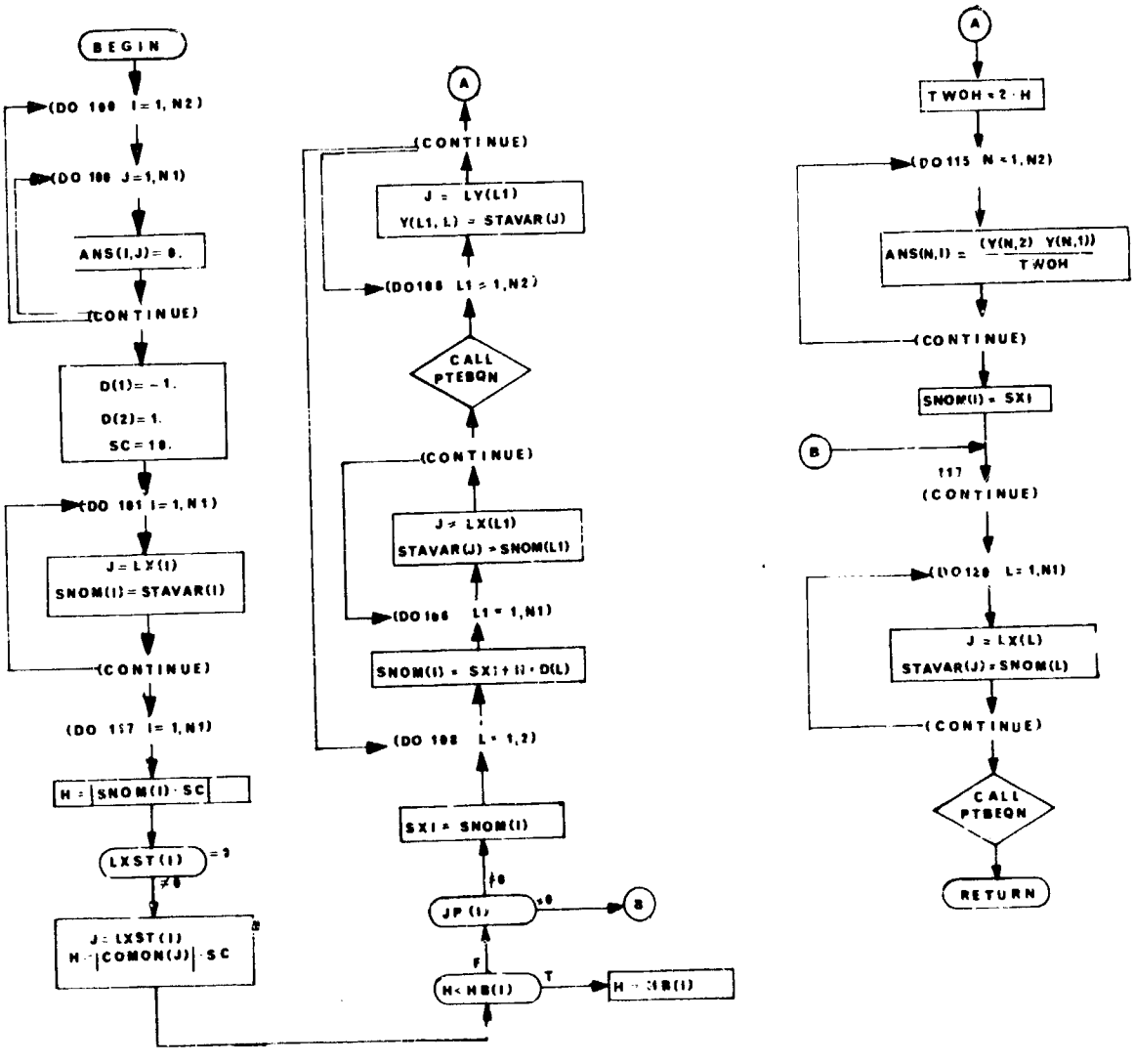
(3) HB is either HBARN, HBARM, HBARI, OR HBART depending on the call; all of these are input.

(4) PSUBR charges the routine PTBEQN with the responsibility for driving the calculations of the f functions specified by LFCNS.

(5) The basic state variables for the second vehicle may be perturbed through PSUBR. Appropriate transformations to permit such perturbations are introduced through subroutines ONETWO and TWOONE.

(6) Perturbation of vehicle two's control variables in a cooperative variational optimization requires a program modification in ONETWO and TWOONE.

PSUBR


BEGIN
(DO 100 I=1,N2)
(DO 100 J=1,N1)
ANS(I,J)=0.
(CONTINUE)
D(1)=-1.
D(2)=1.
SC=10.
(DO 101 I=1,N1)
J=LX(I)
SNOM(I)=STAVAR(I)
(CONTINUE)
(DO 117 I=1,N1)
H=|SNOM(I)·SC|
LXST(I)
=0
≠0
J=LXST(I)
H=|COMON(J)|·SC
A
(CONTINUE)
J=LY(L1)
Y(L1,L)=STAVAR(J)
(DO 108 L1=1,N2)
CALL PTBEQN
(CONTINUE)
J=LX(L1)
STAVAR(J)=SNOM(L1)
(DO 106 L1=1,N1)
SNOM(I)=SXI+H·D(L)
(DO 108 L=1,2)
SXI=SNOM(I)
JP(I)
=0
B
F
T
H<HB(I)
A
TWOH=2·H
(DO 115 N=1,N2)
ANS(N,I)=(Y(N,2)-Y(N,1))/TWOH
(CONTINUE)
SNOM(I)=SXI
B
117
(CONTINUE)
(DO 120 L=1,N1)
J=LX(L)
STAVAR(J)=SNOM(L)
(CONTINUE)
CALL PTBEQN
RETURN

33. PRPACK - Blocking Routine for Partials

Purpose

To enter values into an array for output to tape.

Method

All non-zero numbers (integer and floating point) are packed into a large array before being output to tape.

Usage

Linkage to this routine is made via the statement.

```
CALL PRPACK (MCONT,NPOINT,NSTATE,IFM,IGM,IT,ND,IALP,IDEL,IWA,
IWAI,IWBI,INDPMT,IPARTS,IK,IENTRY)
```

where,

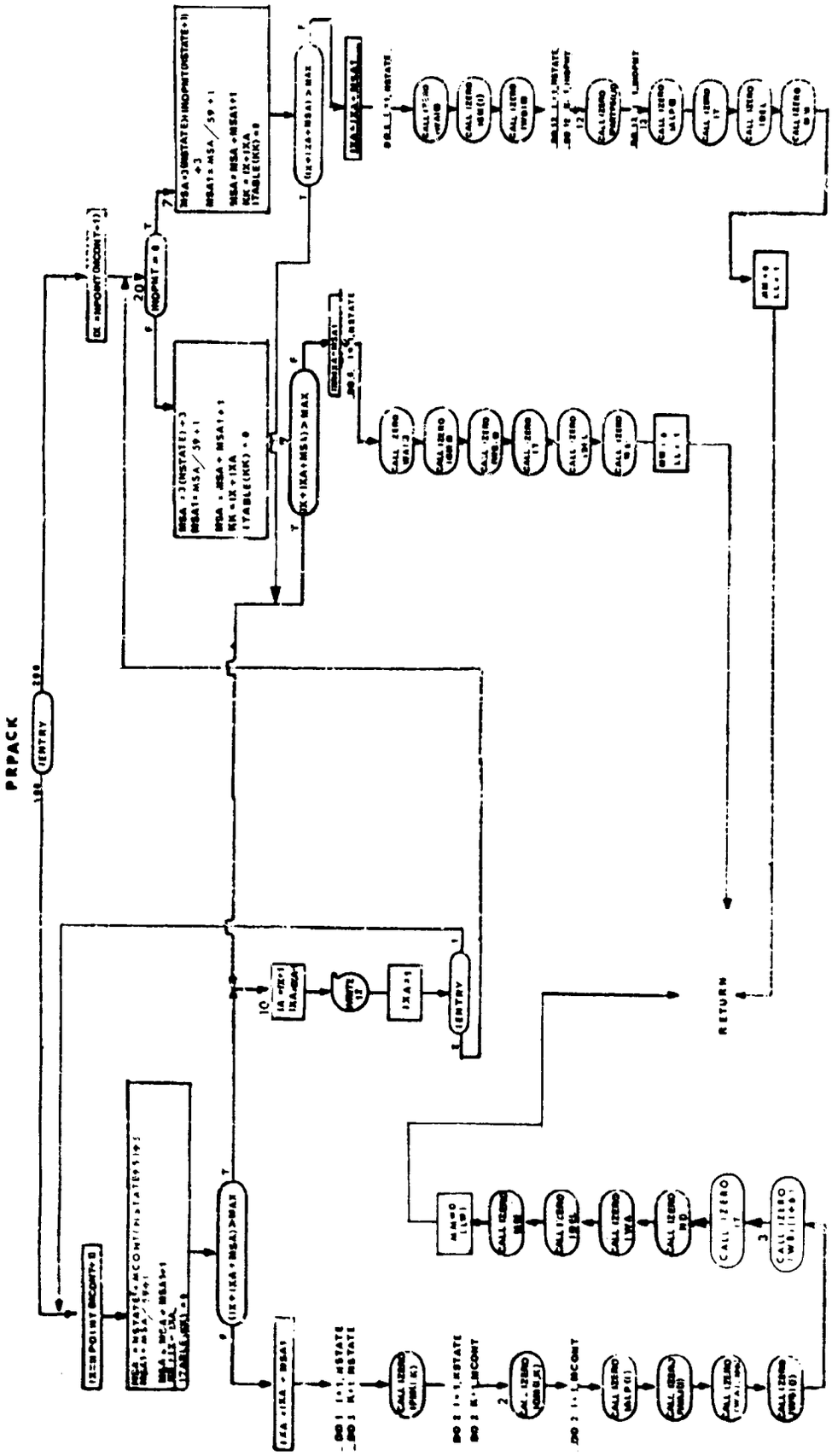
MCONT	is the number of control variables.
NPOINT	is the number of values in CTABLE.
NSTATE	is the number of State Variables.
IFM	is a (NSTATE*NSTATE) Matrix.
IGM	is a (MCONT*NSTATE) Matrix.
IT	TIME
ND	is a control that is passed to the reverse, segment 0 at the beginning of the trajectory, 1 within the stage and 2 at the end of the stage.
IALP	CTABLE points for that time.
IDEL	TIME step.
IWA	Weighting Matrix indicator
IWAI	Constants used to construct the W' Matrix.
IWBI	Constants used to construct the W' Matrix.
INDPMT	Number of end constraints.
IPARTS	array of terminal partials
IK	not used.
IENTRY	serves as an entry point to the subroutine.

COMMON References

CTABLE	in numbered COMMON of 1

Subroutines Called

IZERO
FORTRAN I/O routines.


PRPACK
RETURN

34. FLUSH1 and FLUSH12 - Buffer Flush Routine for PRPACK

Purpose

To write on tape a partially-filled buffer.

Method

To test and determine if a buffer is partially filled and if so to write that part out on tape.

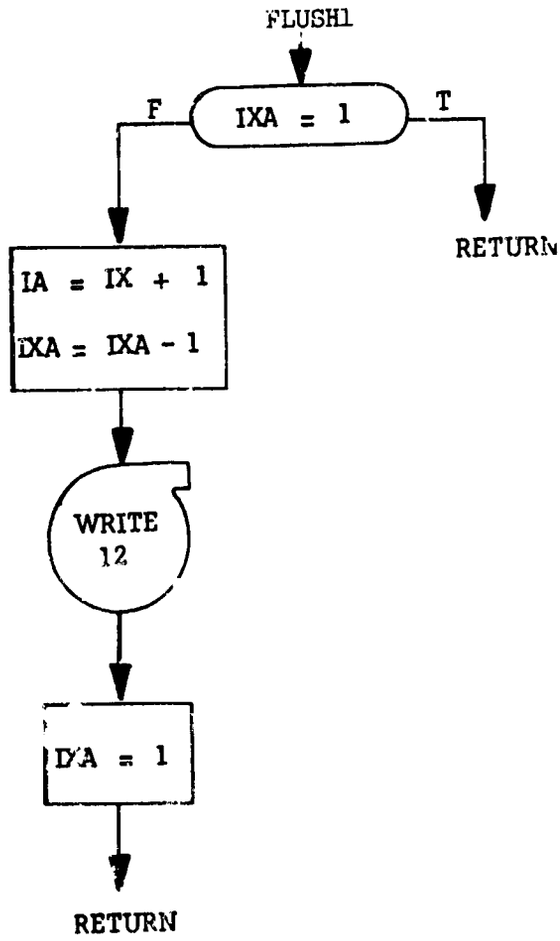
Usage

Entry is made to this routine by the following statement:

```
CALL FLUSH1
```

Remarks

Only the normal I/O FORTRAN routines are used by this routine. A flow chart for FLUSH1 is presented. FLUSH12 is identical except for the use of vehicle 2 COMMON blocks and units.


FLUSH1
IXA = 1
F
T
RETURN
IA = IX + 1
IXA = IXA - 1
WRITE
12
IXA = 1
RETURN

35. SETGRD -- Paper Plot Grid Size Routine

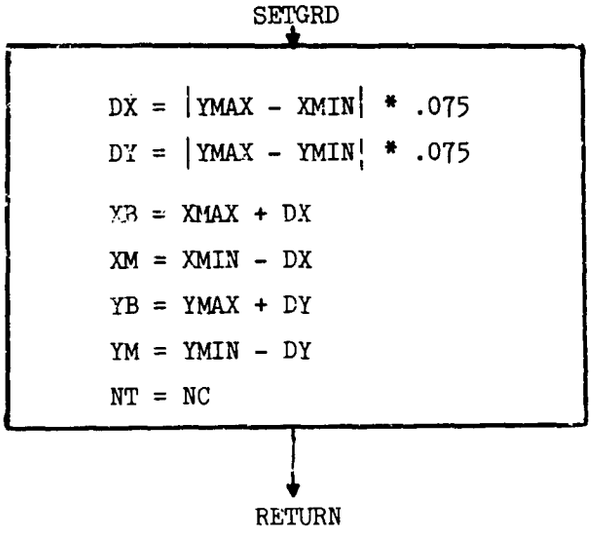
Purpose:

To establish maximum and minimum dimension for x and y axes for each paper plot.

Usage:

Call SETGRD (XMAX, XMIN, YMAX, YMIN, NC)

XMAX - Maximum value in the X array
XMIN - Minimum value in the X array
YMAX - Maximum value in the Y array
YMIN - Minimum value in the Y array
NC - Number of curves to the plotted



36. PAPERP - Printer-Plot Control Routine

Purpose:

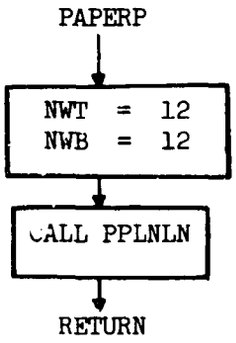
To provide on-line paper plot capability during equation of motion computation.

Usage:

Call PAPERP (X, Y, NX, TITLET, TITLEB)

Where

X	is an array that contains the values of the independent variable to be plotted
Y	is an array that contains the values of the dependent variable to be plotted
NX	is the number of data points to be plotted
TITLET	120 character title that will be placed at the top of the plot area
TITLEB	120 character title that will be placed at the bottom of the plot area



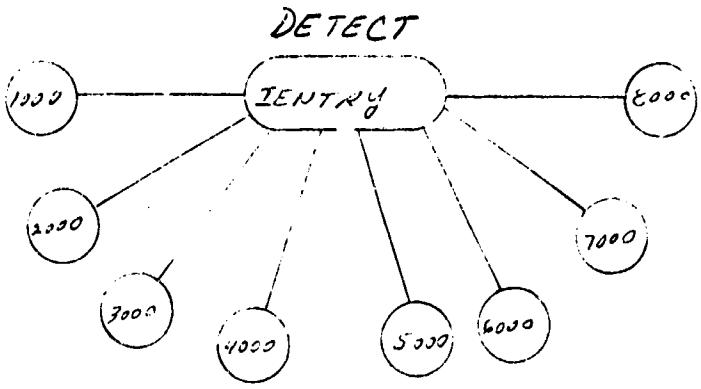
37. DETECT and DETECT2 - Sensor Control Program

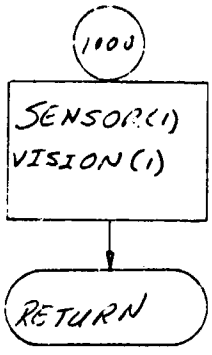
Purpose:

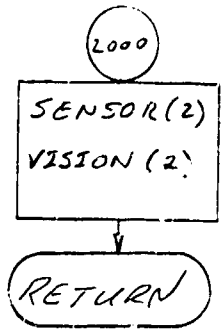
To control the sensor and vision routines.

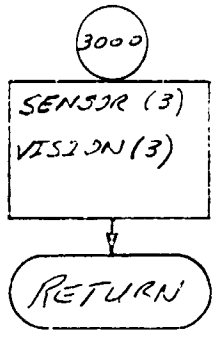
Remarks:

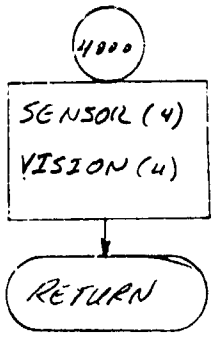
A flow chart for DETECT is presented. DETECT2 is identical except for the use of vehicle 2 auxiliary subroutines.

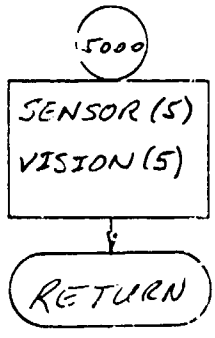

DETECT
1000
ENTRY
8000
2000
7000
3000
4000
5000
6000

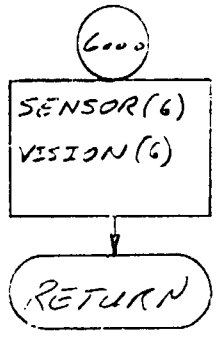

1000
SENSOR(1)
VISION(1)
RETURN

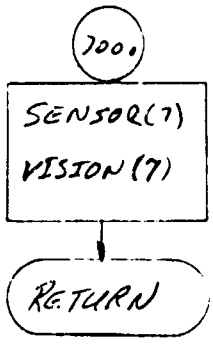

2000
SENSOR(2)
VISION(2)
RETURN

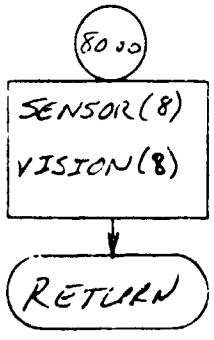

3000
SENSOR(3)
VISION(3)
RETURN


4000
SENSOR(4)
VISION(4)
RETURN


5000
SENSOR(5)
VISION(5)
RETURN


6000
SENSOR(6)
VISION(6)
RETURN


7000
SENSOR(7)
VISION(7)
RETURN


8000
SENSOR(8)
VISION(8)
RETURN

38. ROLE1 and ROLE2 - Role Selection Subprogram

Purpose:

To define each vehicle's combative role on the basis of instantaneous vehicle states.

Method:

Each vehicle role is selected on the basis of their relative states. Role selection includes the following:

a. ATTACK

b. OFFENSIVE

c. DEFENSIVE

d. EVASIVE

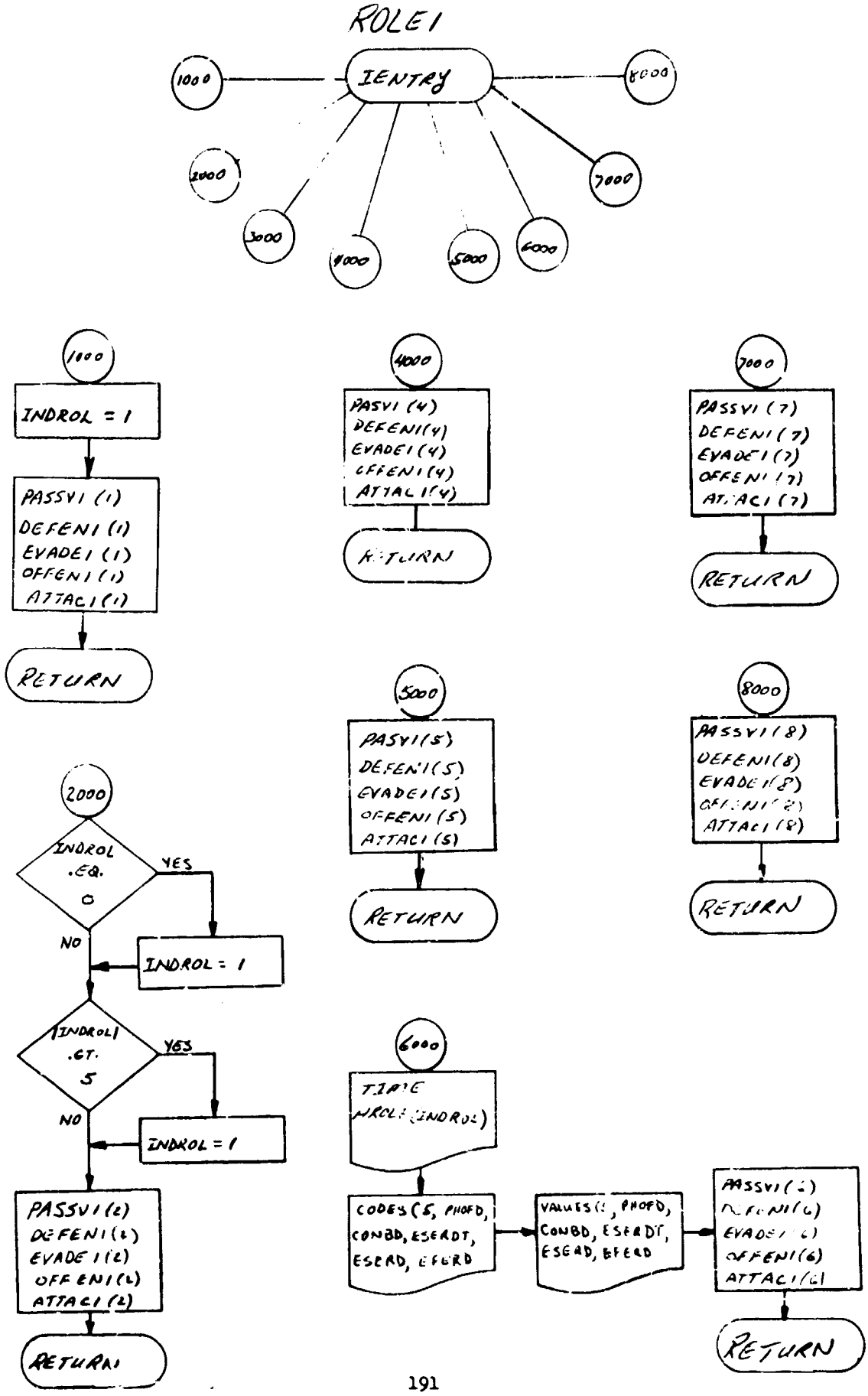
An override permits selection of a fifth role

e. PASSIVE

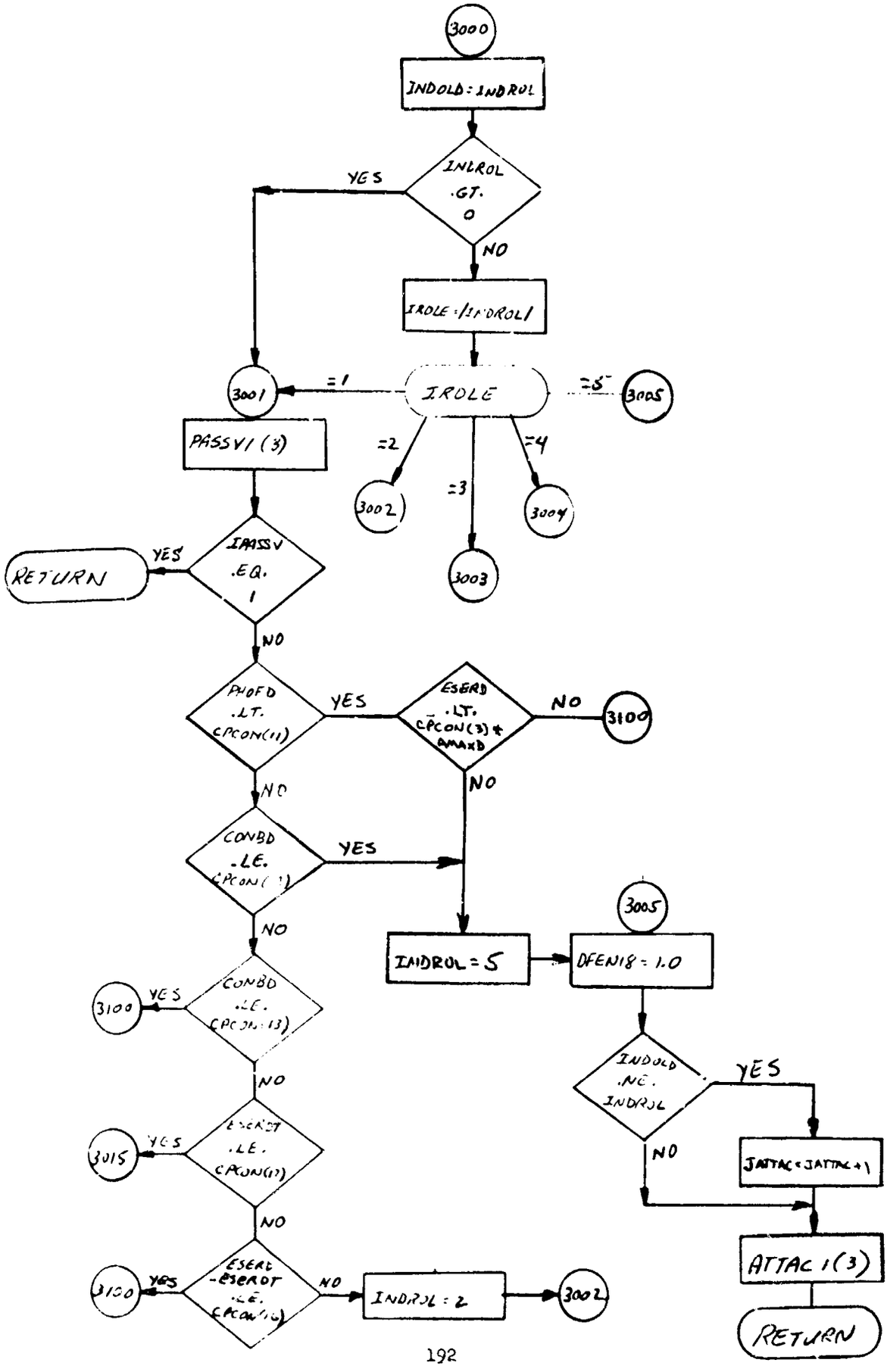
The subprogram contains the standard EXE ENTRY points.

Remarks:

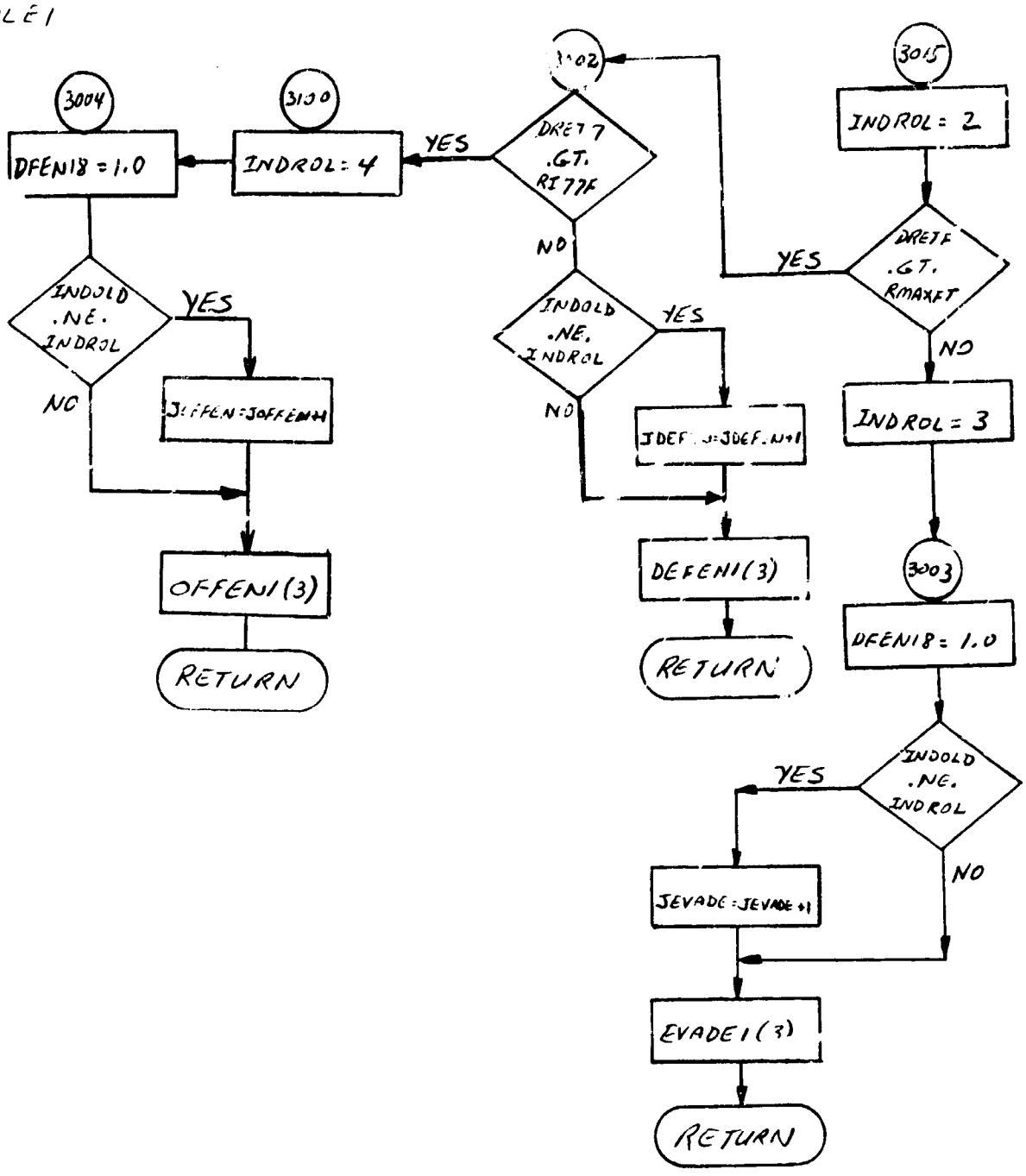
A flow chart for ROLE1 is presented. ROLE2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.


ROLE1
IENTRY
1000
2000
3000
4000
5000
6000
7000
8000
1000
INDROL = 1
PASSVI (1)
DEFENI (1)
EVADEI (1)
OFFENI (1)
ATTACI (1)
RETURN
2000
INDROL .EQ. 0
YES
NO
INDROL = 1
INDROLI .GT. 5
YES
NO
INDROL = 1
PASSVI (2)
DEFENI (2)
EVADEI (2)
OFFENI (2)
ATTACI (2)
RETURN
4000
PASSVI (4)
DEFENI (4)
EVADEI (4)
OFFENI (4)
ATTACI (4)
RETURN
5000
PASSVI (5)
DEFENI (5)
EVADEI (5)
OFFENI (5)
ATTACI (5)
RETURN
6000
TIME
CODES (5, PHOFD, CONBD, ESERDT, ESERD, EFERD
VALUES (5, PHOFD, CONBD, ESERDT, ESERD, EFERD
PASSVI (6)
DEFENI (6)
EVADEI (6)
OFFENI (6)
ATTACI (6)
RETURN
7000
PASSVI (7)
DEFENI (7)
EVADEI (7)
OFFENI (7)
ATTACI (7)
RETURN
8000
PASSVI (8)
DEFENI (8)
EVADEI (8)
OFFENI (8)
ATTACI (8)
RETURN

ROLE 1


3000
INDOLD = INDROL
INDROL .GT. 0
YES
NO
IROLE = |INDROL|
IROLE
=1
=2
=3
=4
=5
3001
3002
3003
3004
3005
PASSV1 (3)
IPASSV .EQ. 1
YES
RETURN
NO
PHOFD .LT. CPCON(11)
YES
ESERD .LT. CPCON(3) * AMAXD
NO
3100
NO
NO
CONBD .LE. CPCON(
YES
NO
INDROL = 5
3005
DFEN18 = 1.0
CONBD .LE. CPCON(13)
YES
3100
NO
INDOLD .NE. INDROL
YES
NO
JATTAC = JATTAC + 1
ESERDT .LE. CPCON(17)
YES
3015
NO
ESERD - ESERDT .LE. CPCON(16)
YES
3100
NO
INDROL = 2
3002
ATTAC 1(3)
RETURN

ROLE1


3004
DFENI8 = 1.0
INDOLD .NE. INDROL
YES
NO
JOFFEN=JOFFEN+1
OFFENI(3)
RETURN
3100
INDROL = 4
YES
DRET 7 .GT. RI77F
NO
INDOLD .NE. INDROL
YES
NO
JDEF. J=JDEF. N+1
DEFENI(3)
RETURN
3015
INDROL = 2
DRETF .GT. RMAXFT
YES
NO
INDROL = 3
3003
DFENI8 = 1.0
INDOLD .NE. INDROL
YES
NO
JEVADE=JEVADE+1
EVADEI(3)
RETURN

39. HLIMIT and HLIMIT2 - Minimum Altitude Constraint Routine

Purpose:

To force a pull-up maneuver when a minimum altitude boundary is violated.

Method:

A minimum altitude is defined as

a. Constant

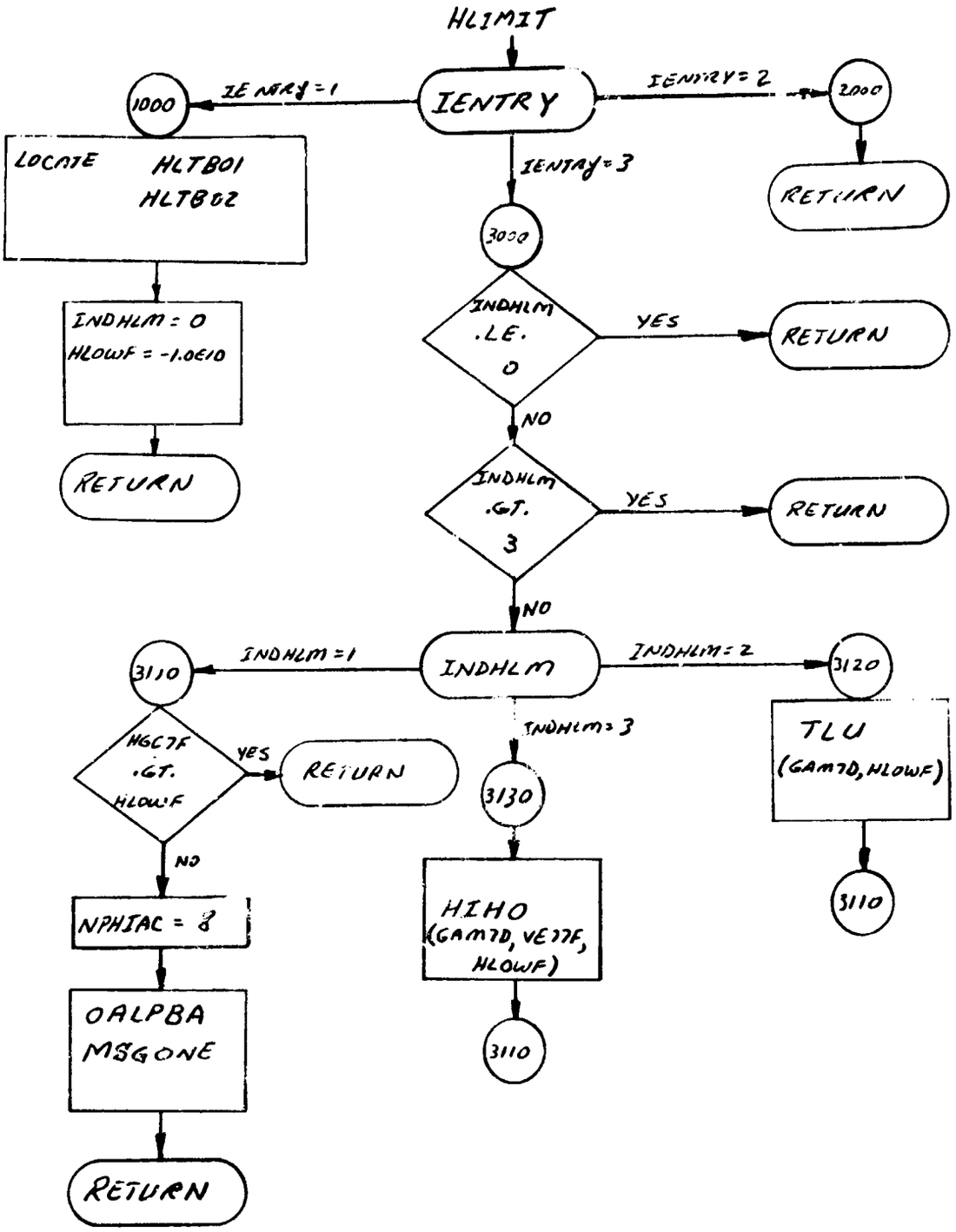
b. Function of γ

c. Function of γ, V

When this minimum altitude limit is not satisfied, the angle-of-attack, bank-angle combination,which maximizes vertical force component, is used. Whenever the altitude limiter is employed, an appropriate message is printed by MSGONE or MSGONE2.

Remarks:

A flow chart for HLIMIT is presented. HLIMIT2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.


HLIMIT
IENTRY
IENTRY = 1
IENTRY = 2
IENTRY = 3
1000
2000
3000
LOCATE HLTB01 HLTB02
RETURN
INDHLM = 0
HLOWF = -1.0E10
RETURN
INDHLM .LE. 0
YES
RETURN
NO
INDHLM .GT. 3
YES
RETURN
NO
INDHLM
INDHLM = 1
INDHLM = 2
INDHLM = 3
3110
3120
3130
HGCTF .GT. HLOWF
YES
RETURN
NO
NPHIAC = 8
OALPBA MSGONE
RETURN
HIHO (GAMTD, VETTF, HLOWF)
3110
TLU (GAMTD, HLOWF)
3110

40. ANGLES and ANGLES2 - Relative Angular Orientation Routines

Purpose:

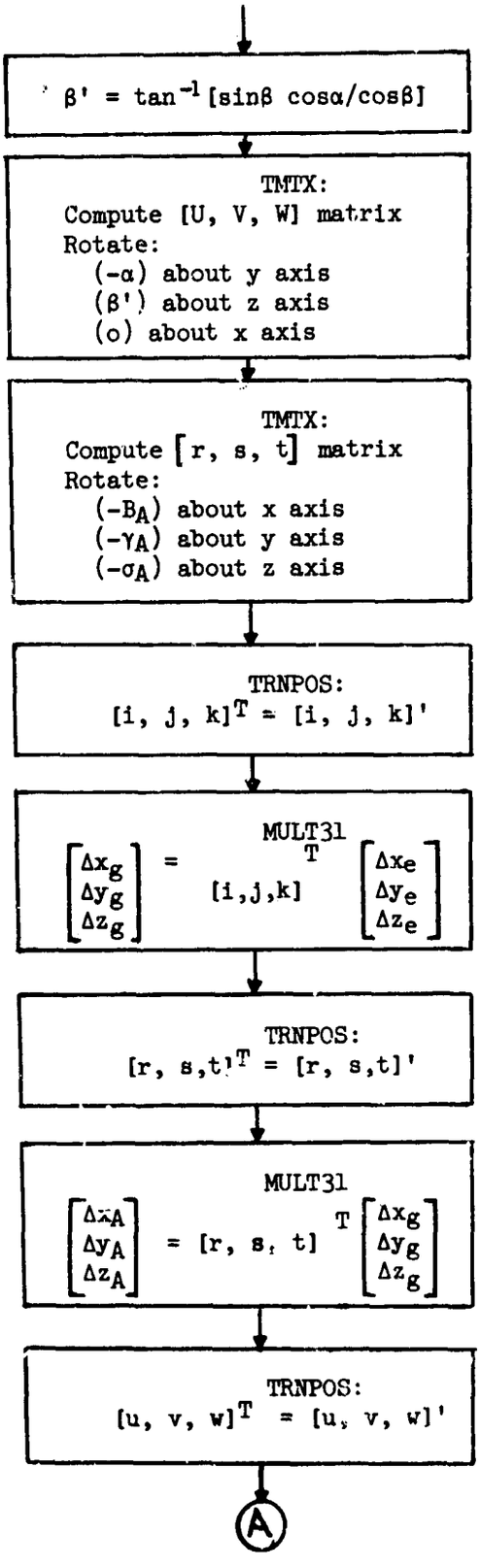
To compute the relative angular orientation of the two vehicles in body axes and to compute steering errors from the reference vector and associated functions.

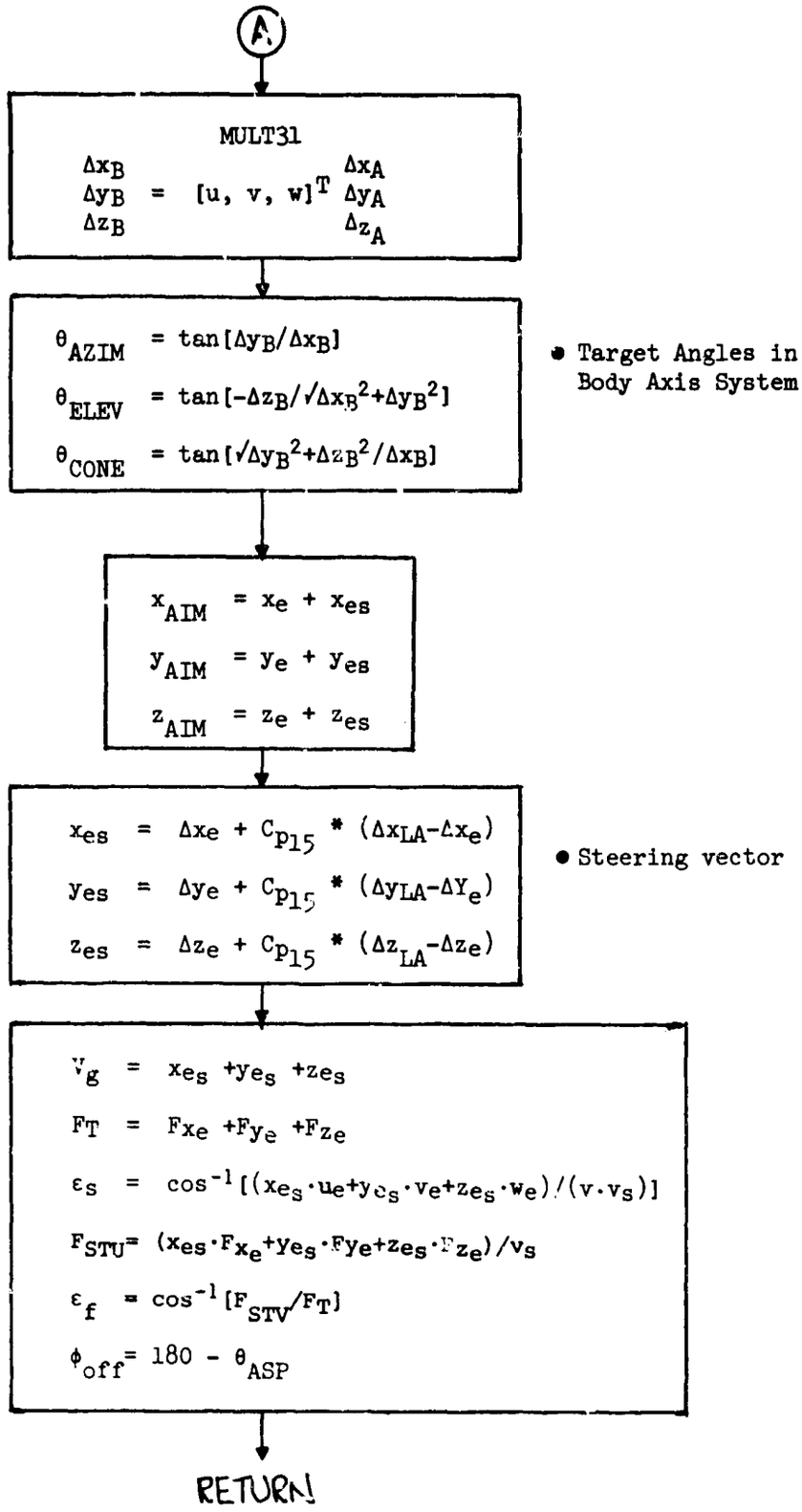
Usage:

Called from the combat logic during equation of motion computations.

Remarks:

A flow chart for ANGLES is presented. ANGLES2 is identical except for the use of vehicle 2 COMMON blocks.


β' = tan⁻¹ [sinβ cosα/cosβ]
TMTX:
Compute [U, V, W] matrix
Rotate:
(-α) about y axis
(β') about z axis
(o) about x axis
TMTX:
Compute [r, s, t] matrix
Rotate:
(-B_A) about x axis
(-γ_A) about y axis
(-σ_A) about z axis
TRNPOS:
[i, j, k]^T = [i, j, k]'
MULT31
[Δx_g, Δy_g, Δz_g] = [i,j,k]^T [Δx_e, Δy_e, Δz_e]
TRNPOS:
[r, s,t]^T = [r, s,t]'
MULT31
[Δx_A, Δy_A, Δz_A] = [r, s, t]^T [Δx_g, Δy_g, Δz_g]
TRNPOS:
[u, v, w]^T = [u, v, w]'
A


A
MULT31
$\begin{matrix}\Delta x_B \\ \Delta y_B \\ \Delta z_B\end{matrix} = [u, v, w]^T \begin{matrix}\Delta x_A \\ \Delta y_A \\ \Delta z_A\end{matrix}$
$\theta_{AZIM} = \tan[\Delta y_B/\Delta x_B]$
$\theta_{ELEV} = \tan[-\Delta z_B/\sqrt{\Delta x_B^2+\Delta y_B^2}]$
$\theta_{CONE} = \tan[\sqrt{\Delta y_B^2+\Delta z_B^2}/\Delta x_B]$
• Target Angles in Body Axis System
$x_{AIM} = x_e + x_{es}$
$y_{AIM} = y_e + y_{es}$
$z_{AIM} = z_e + z_{es}$
$x_{es} = \Delta x_e + C_{p15} * (\Delta x_{LA}-\Delta x_e)$
$y_{es} = \Delta y_e + C_{p15} * (\Delta y_{LA}-\Delta Y_e)$
$z_{es} = \Delta z_e + C_{p15} * (\Delta z_{LA}-\Delta z_e)$
• Steering vector
$V_g = x_{es} + y_{es} + z_{es}$
$F_T = F_{xe} + F_{ye} + F_{ze}$
$\varepsilon_s = \cos^{-1}[(x_{es}\cdot u_e + y_{es}\cdot v_e + z_{es}\cdot w_e)/(v\cdot v_s)]$
$F_{STU} = (x_{es}\cdot F_{xe} + y_{es}\cdot F_{ye} + z_{es}\cdot F_{ze})/v_s$
$\varepsilon_f = \cos^{-1}[F_{STV}/F_T]$
$\phi_{off} = 180 - \theta_{ASP}$
RETURN!

41. CRATE and CRATE2 - Finite Control Rate Routines

Purpose:

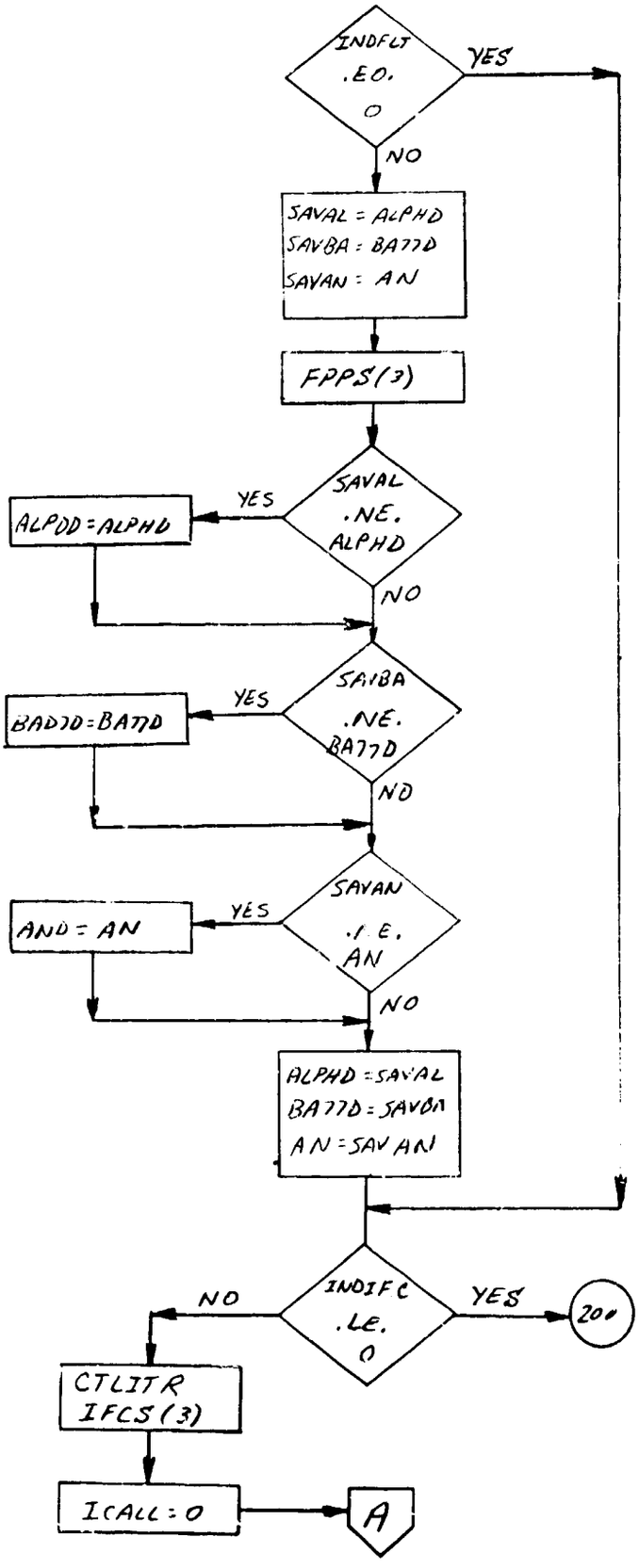
To introduce a finite control rate capability into combat simulations.

Method:

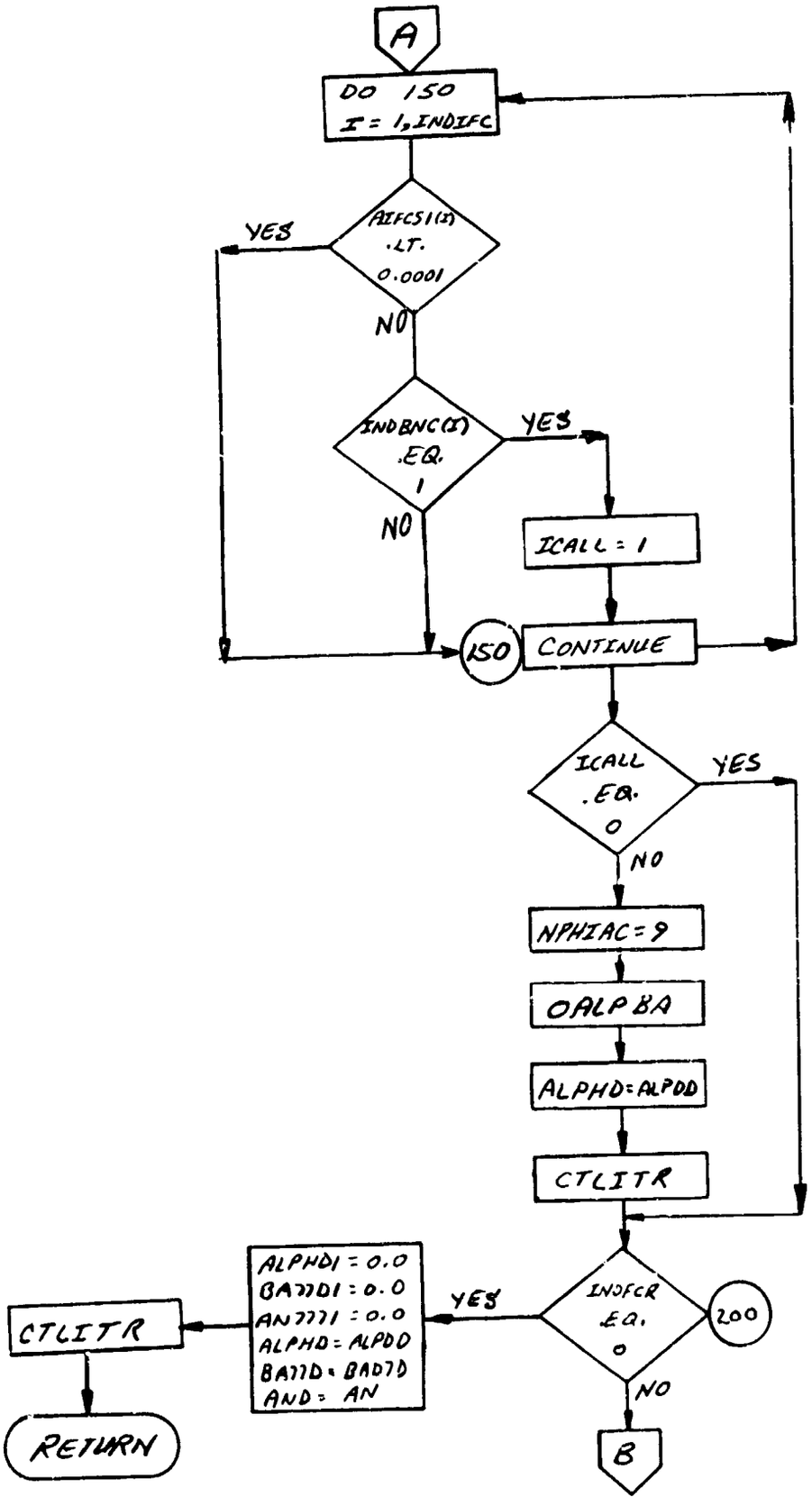
Desired control vector components are defined by either combative logic COMBAT and COMBAT2 or the flight plan programmer routines, FPPS or FPPS2. The instantaneous control error is then used to define an error magnitude-dependent control rate. Control values are obtained by integration of their rates. Logic to maintain controllable inequalities is incorporated in the finite control rate option.

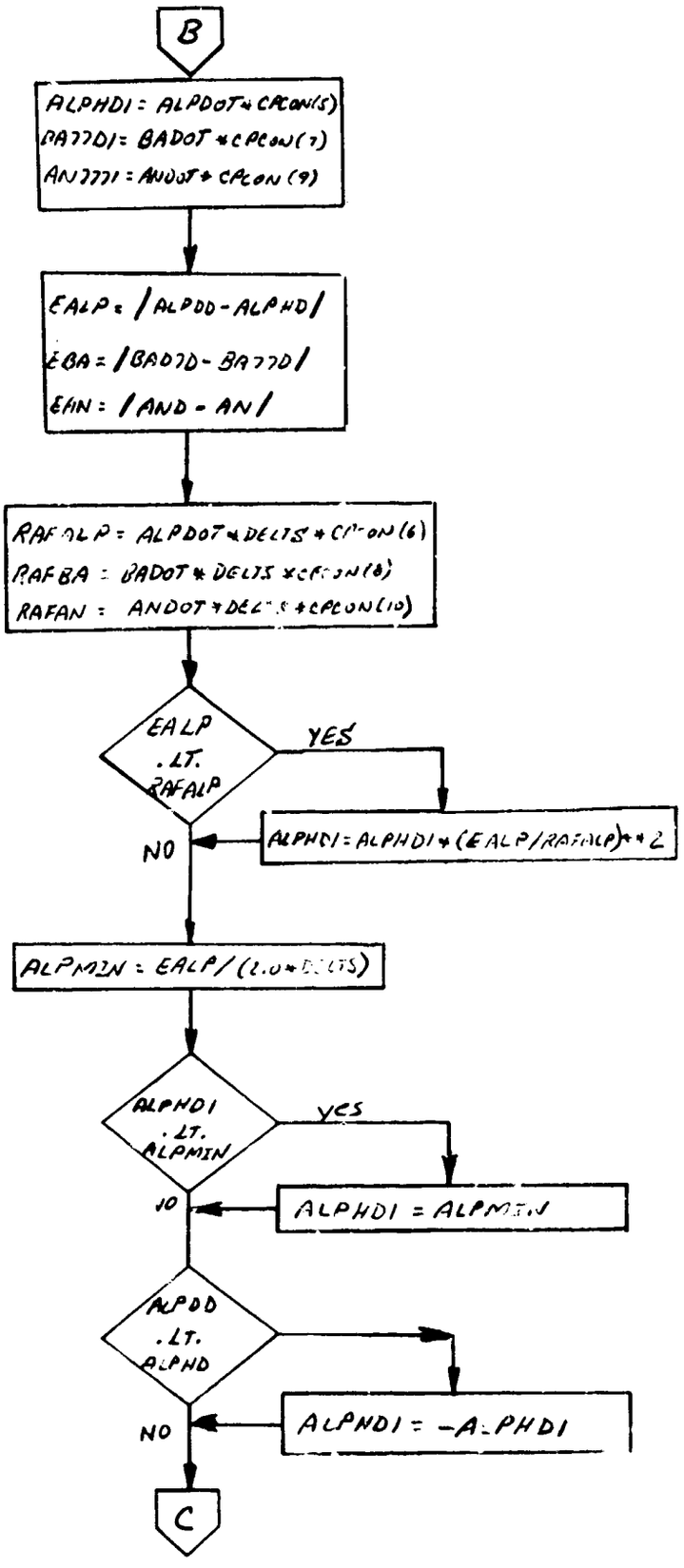
Remarks:

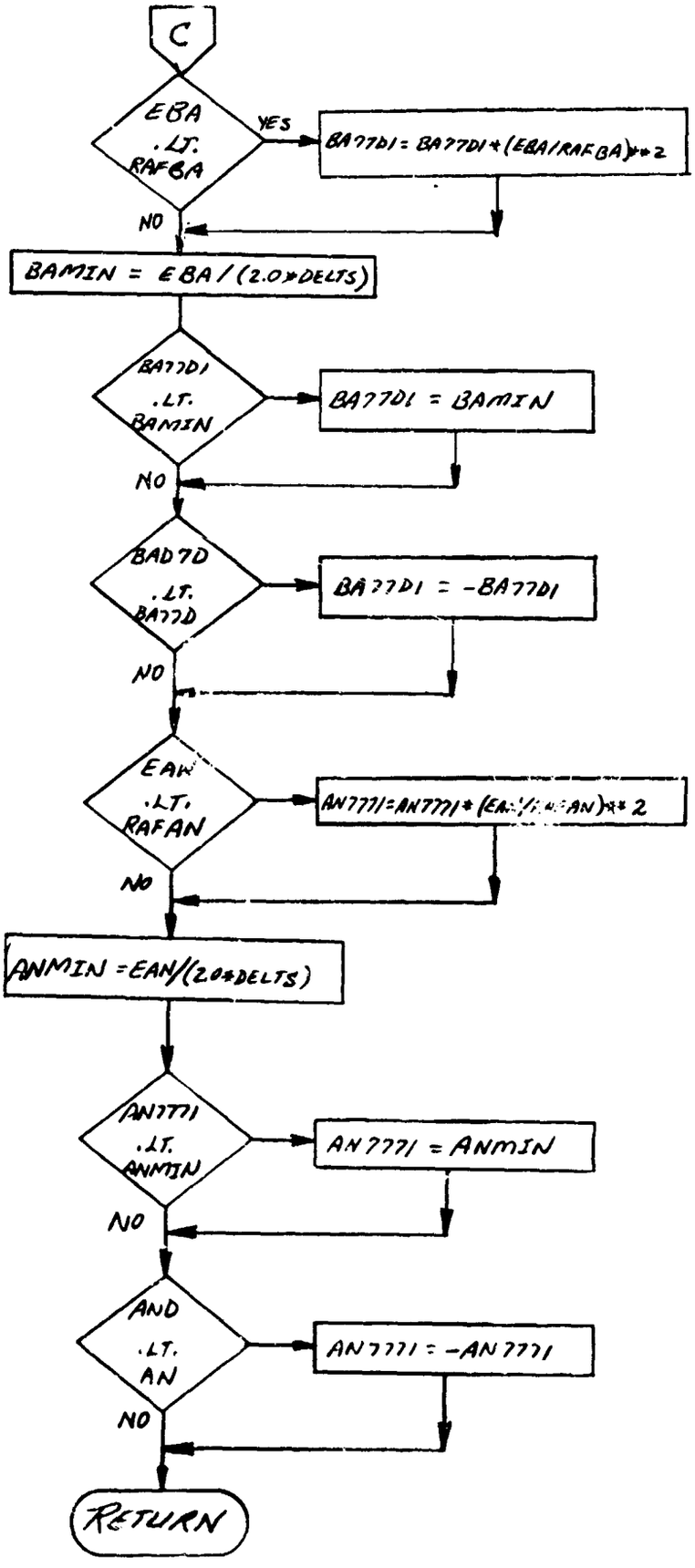
A flow chart for CRATE is presented. CRATE2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.


INDFLT .EQ. 0
YES
NO
SAVAL = ALPHD
SAVBA = BATTD
SAVAN = AN
FPPS(3)
SAVAL .NE. ALPHD
YES
ALPOD = ALPHD
NO
SAVBA .NE. BATTD
YES
BADTD = BATTD
NO
SAVAN .NE. AN
YES
AND = AN
NO
ALPHD = SAVAL
BATTD = SAVBA
AN = SAVAN
INDIFC .LE. 0
NO
YES
200
CTLITR IFCS(3)
ICALL = 0
A

CRATE


A
DO 150
I = 1, INDIFC
AIFCS1(I)
.LT.
0.0001
YES
NO
INDBNC(I)
.EQ.
1
YES
NO
ICALL = 1
150
CONTINUE
ICALL
.EQ.
0
YES
NO
NPHIAC = 9
OALPBA
ALPHD = ALPDD
CTLITR
INDFCR
.EQ.
0
YES
200
NO
B
ALPHD1 = 0.0
BA77D1 = 0.0
AN7777 = 0.0
ALPHD = ALPDD
BA77D = BAD7D
AND = AN
CTLITR
RETURN


B
ALPHDI = ALPDOT * CACON(5)
BA77DI = BADOT * CACON(7)
AN7771 = ANDOT * CACON(9)
EALP = /ALPDD - ALPHD/
EBA = /BADJD - BA77D/
EAN = /AND - AN/
RAFALP = ALPDOT * DELTS * CACON(6)
RAFBA = BADOT * DELTS * CACON(8)
RAFAN = ANDOT * DELTS * CACON(10)
EALP .LT. RAFALP
YES
ALPHDI = ALPHDI * (EALP/RAFALP) ** 2
NO
ALPMIN = EALP/(2.0 * DELTS)
ALPHDI .LT. ALPMIN
YES
ALPHDI = ALPMIN
NO
ALPDD .LT. ALPHD
ALPNDI = -ALPHDI
C


C
EBA .LT. RAFBA
YES
BA7701 = BA7701 * (EBA/RAFBA)**2
NO
BAMIN = EBA / (2.0*DELTS)
BA7701 .LT. BAMIN
BA7701 = BAMIN
NO
BAD7D .LT. BA77D
BA7701 = -BA7701
NO
EAN .LT. RAFAN
AN7771=AN7771 * (EA
AN)**2
NO
ANMIN =EAN/(2.0*DELTS)
AN7771 .LT. ANMIN
AN7771 = ANMIN
NO
AND .LT. AN
AN7771 = -AN7771
NO
RETURN

42. TIM001 and TIM0012 - Tabular Time Point Routine

Purpose:

To provide additional time points that the integration routine must "hit" in the forward trajectory.

Usage:

CALL TIM001 (VAL,TPT)

VAL is the current time value.

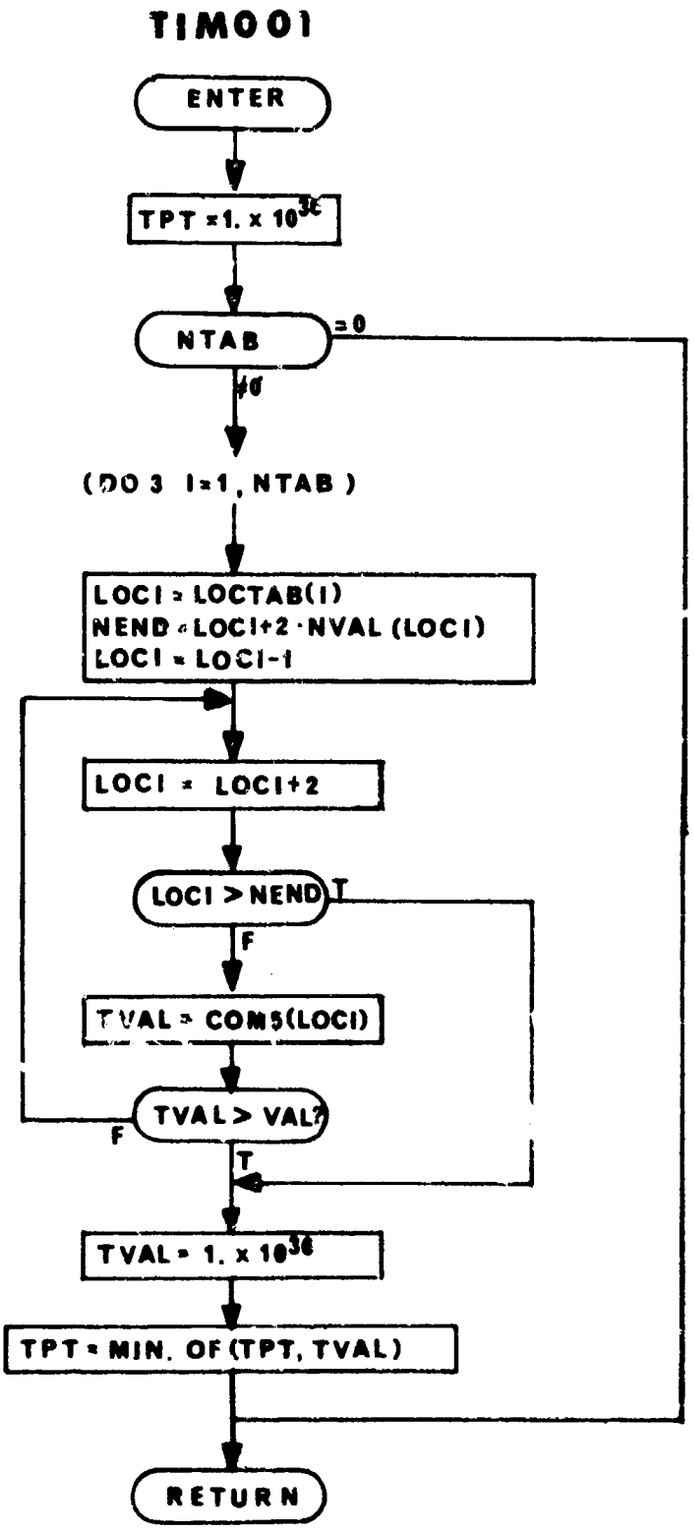
TPT is the next time point larger than VAL which should be "hit" in addition to other time points normally "hit" in the forward trajectory. TIM001 performs a table look-up on specified 2-dimensional tables to find the smallest time value in these tables which is larger than VAL.

If no tables are specified TIM001 returns 1.E36 for TPT.

TIM001 will use the tables specified on the TIMTAB card.

Remarks:

TIM001 is called by TIMID. TIM0012 is called by TIMID2 and provides for additional time points in the vehicle 2 trajectory. TIM0012 is identical to TIM001 except for the use of the vehicle 2 COMMON blocks. A flow chart for TIM001 is presented.


TIM001
ENTER
TPT =1. x 10^36
NTAB
=0
≠0
(DO 3 I=1, NTAB)
LOCI = LOCTAB(I)
NEND = LOCI+2 · NVAL (LOCI)
LOCI = LOCI-1
LOCI = LOCI+2
LOCI > NEND
T
F
TVAL = COM$(LOCI)
TVAL > VAL?
F
T
TVAL = 1. x 10^36
TPT = MIN. OF (TPT, TVAL)
RETURN

43. DEQPRE and DEQPRE2 - Equation of Motion Pre-Data Initialization

Purpose:

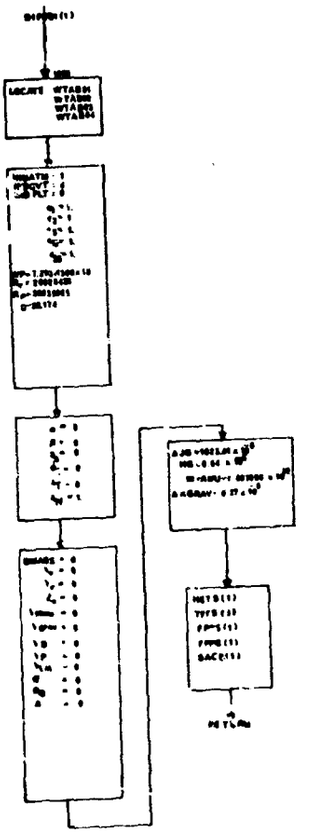
Perform the DIFEQ1(1) and DIFEQ2(1) pre-data read functions for vehicle 1 and vehicle 2.

Method:

Nominal values of indicators are set, and the integrated variable values are set to zero. Nominal values are set for standard constraints used in the equations (e.g., polar and equatorial radius of planet).

Remarks:

A flow chart for DEQPRE is presented. DEQPRE2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines and subprograms.



Subroutine DEQPRE

44. FIRFUN and FIRFUN2 - Fire Control Subprogram

Purpose:

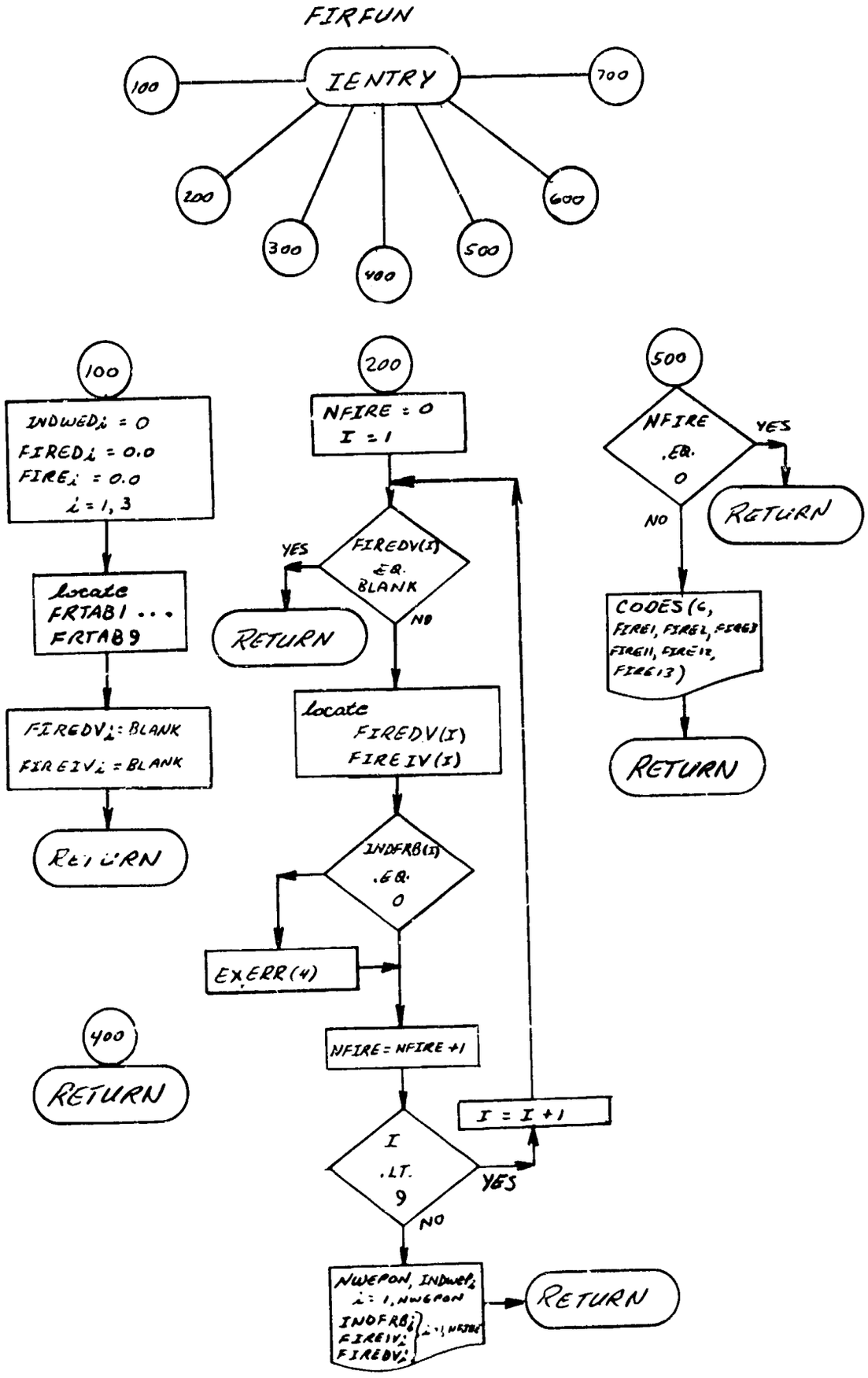
To compute selected fire control function effective time maneuvers.

Method:

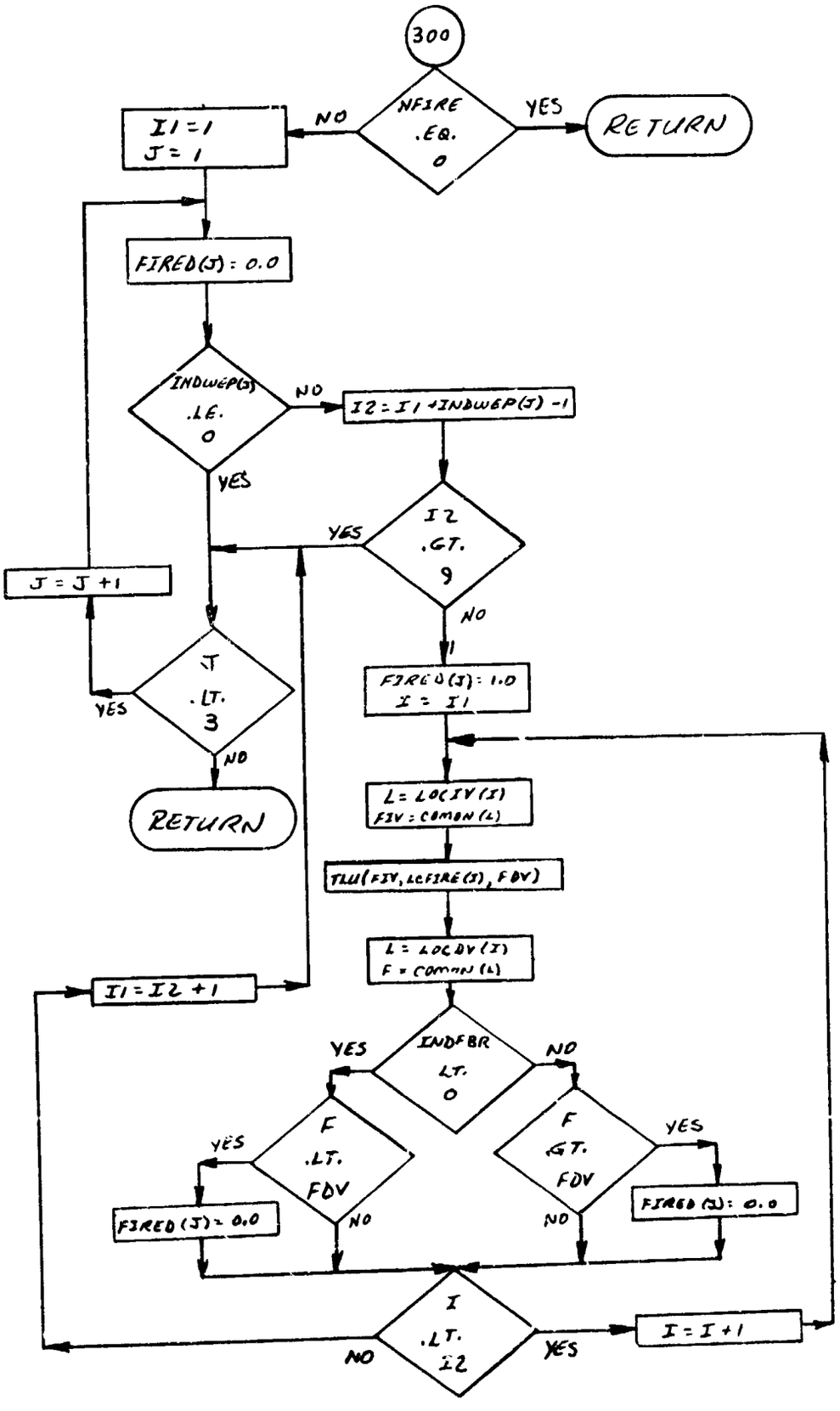
The fire control subroutines integrate the period of time that each of three fire control characteristic function constraints are satisfied. The three fire control functions for each vehicle may involve a combination of up to nine trajectory variable values. FIRFUN and FIRFUN2 have the standard EXE entry points. The subprograms are called by DIFEQ and DIFEQ2, respectively.

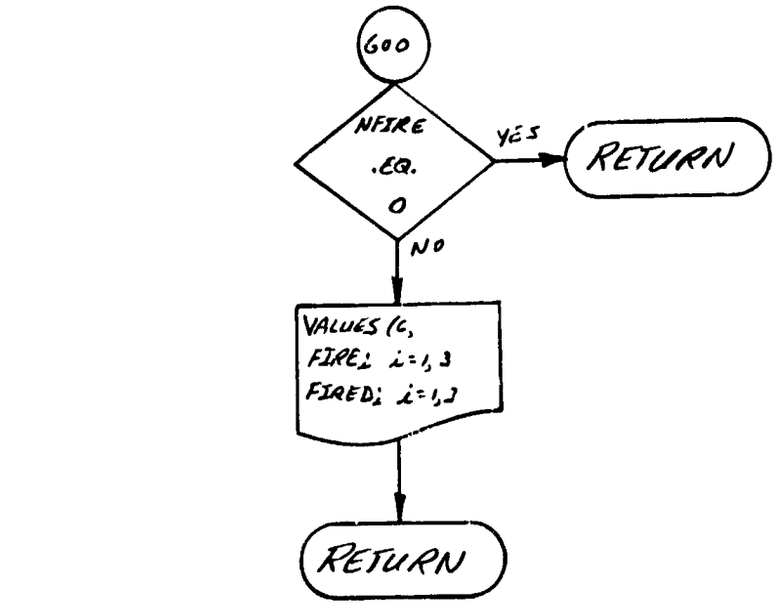
Remarks:

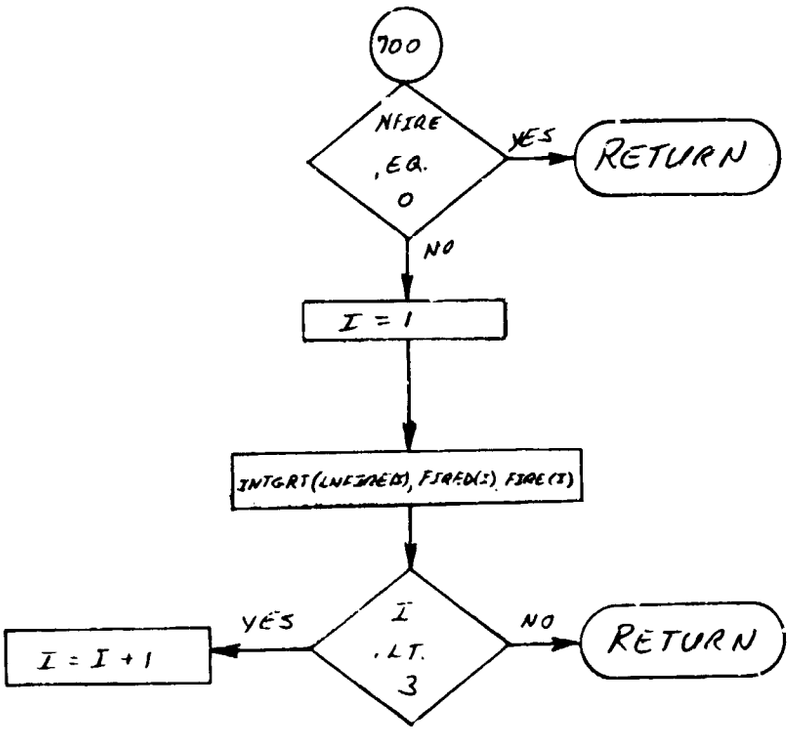
A flow chart for FIRFUN is presented. FIRFUN2 is identical to FIRFUN except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.


FIRFUN
IENTRY
100
200
300
400
500
600
700
100
INDWED$_i$ = 0
FIRED$_i$ = 0.0
FIRE$_i$ = 0.0
i = 1, 3
Locate FRTAB1 ... FRTAB9
FIREDV$_i$ = BLANK
FIREIV$_i$ = BLANK
RETURN
200
NFIRE = 0
I = 1
FIREDV(I) .EQ. BLANK
YES
NO
RETURN
Locate FIREDV(I) FIREIV(I)
INDFRB(I) .GE. 0
EXERR(4)
NFIRE = NFIRE + 1
I = I + 1
I .LT. 9
YES
NO
NWEPON, INDWED$_i$ i = 1, NWEPON
INDFRB$_i$ FIREIV$_i$ FIREDV$_i$ i = 1, NSTRE
RETURN
400
RETURN
500
NFIRE .EQ. 0
YES
NO
RETURN
CODES(6, FIRE1, FIRE2, FIRE3, FIRE11, FIRE12, FIRE13)
RETURN

FIRFUN


300
NFIRE .EQ. 0
YES
RETURN
NO
I1 = 1
J = 1
FIRED(J) = 0.0
INDWEP(J) .LE. 0
NO
I2 = I1 + INDWEP(J) - 1
YES
I2 .GT. 9
YES
NO
FIRED(J) = 1.0
I = I1
J = J + 1
J .LT. 3
YES
NO
RETURN
L = LOC(IV(I))
FIV = COMON(L)
TLU(FIV, LCFIRE(J), FDV)
L = LOC(DV(I))
F = COMON(L)
I1 = I2 + 1
INDFBR .LT. 0
YES
NO
F .LT. FDV
YES
NO
FIRED(J) = 0.0
F .GT. FDV
YES
NO
FIRED(J) = 0.0
I .LT. I2
NO
YES
I = I + 1


600
NFIRE
.EQ.
0
YES
RETURN
NO
VALUES (6,
FIRE(I) I=1,3
FIRED(I) I=1,3
RETURN


700
NFIRE
.EQ.
0
YES
RETURN
NO
I = 1
INTGRT(LNFIRED(I), FIRED(I), FIRE(I))
I
.LT.
3
YES
I = I + 1
NO
RETURN

45. GAM91 and GAM92 - Flight through Vertical Routines

Purpose:

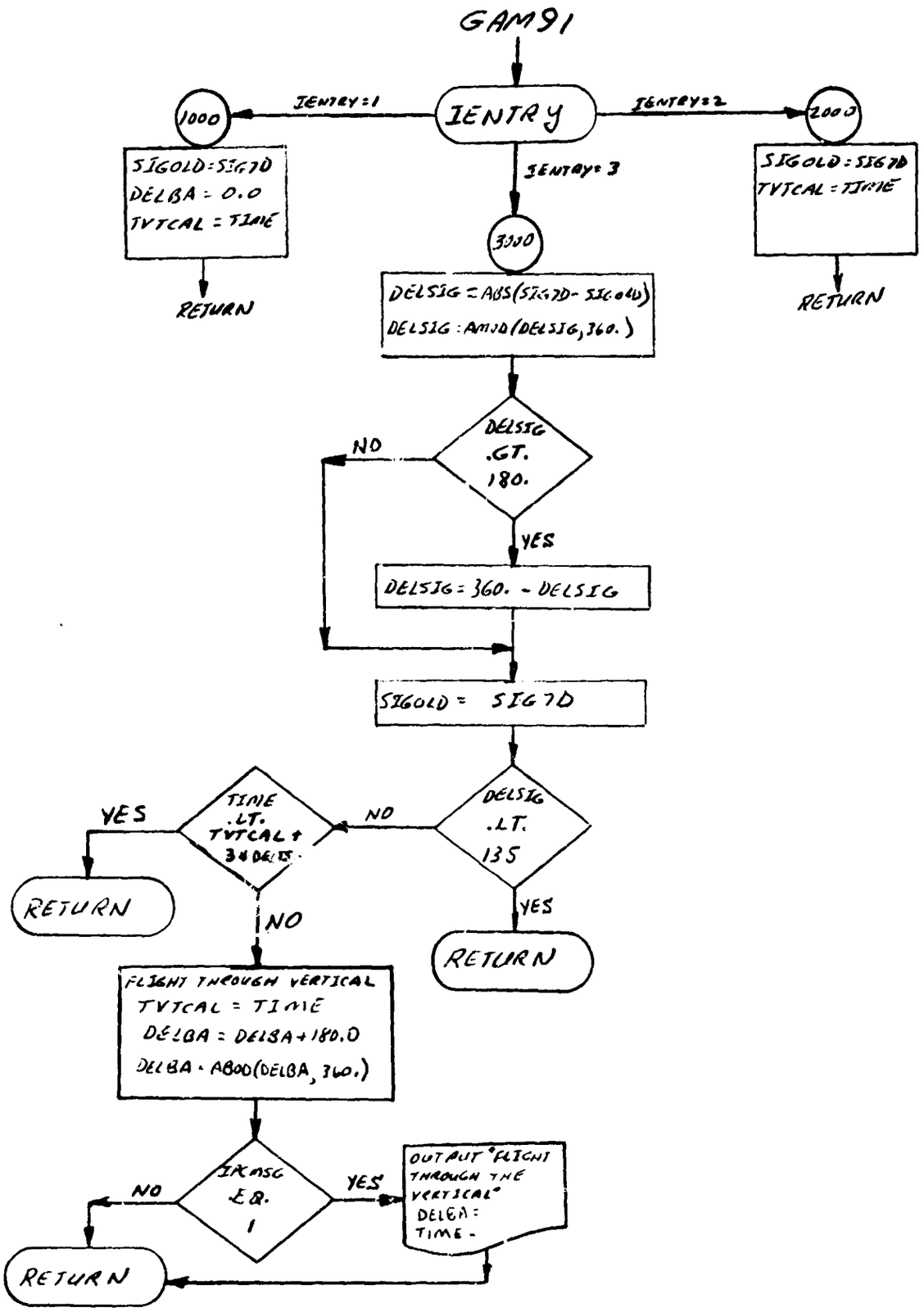
To permit each vehicle to fly through the vertical without "locking on" to the wind axis.

Method:

Whenever programmed logic indicates that a vehicle has flown through the vertical, a 180 degree rotation about the velocity vector is introduced. This rotation is combined with the 180 degree rotation introduced by the wind axis transformation at flight through the vertical. The resulting combined rotation permits a smooth passage through the vertical, and Immelmann- or Split S-like maneuvers can be performed.

Remarks:

A flow chart for GAM91 is presented. GAM92 is identical except for use of vehicle 2 COMMON blocks.


GAM91
IENTRY
IENTRY=1
IENTRY=2
IENTRY=3
1000
2000
3000
SIGOLD=SIG7D
DELBA = 0.0
TVTCAL = TIME
RETURN
SIGOLD=SIG7D
TVTCAL = TIME
RETURN
DELSIG = ABS(SIG7D - SIGOLD)
DELSIG = AMOD(DELSIG, 360.)
DELSIG .GT. 180.
NO
YES
DELSIG = 360. - DELSIG
SIGOLD = SIG7D
TIME .LT. TVTCAL + 3 * DELTS
YES
NO
DELSIG .LT. 135
YES
RETURN
NO
RETURN
FLIGHT THROUGH VERTICAL
TVTCAL = TIME
DELBA = DELBA + 180.0
DELBA = AMOD(DELBA, 360.)
IRASC .EQ. 1
NO
YES
RETURN
OUTPUT "FLIGHT THROUGH THE VERTICAL"
DELBA =
TIME =

46. DEQINI and DEQINI2 - Equation of Motion Post-Data Initialization

Purpose:

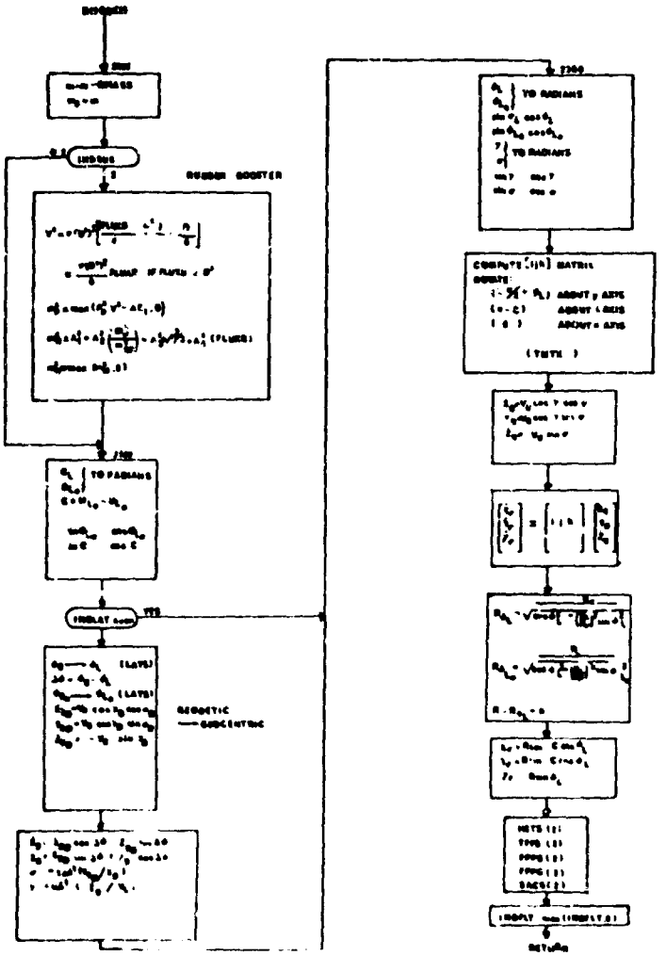
Perform the DIFEQ1(2) and DIFEQ2(2) post-data read function for vehicle 1 and vehicle 2.

Method:

This is the initial transformation. It is always performed at the beginning of a trajectory; it may be performed at the beginning of a major stage (see EXTRAN and EXTRAN2). Also, it may be used in certain combinations for the h-transformation (see EXTRAN and EXTRAN2).

Remarks:

A flow chart for DEQINI is presented. DEQINI2 is identical except for the use of vehicle 2 COMMON blocks, auxiliary subroutines, and subprograms.



SUBROUTINE DEQINI

NOT REPRODUCIBLE

47. DEQBCI and DEQBCI2 - Derivative Calculation Before Control Definition

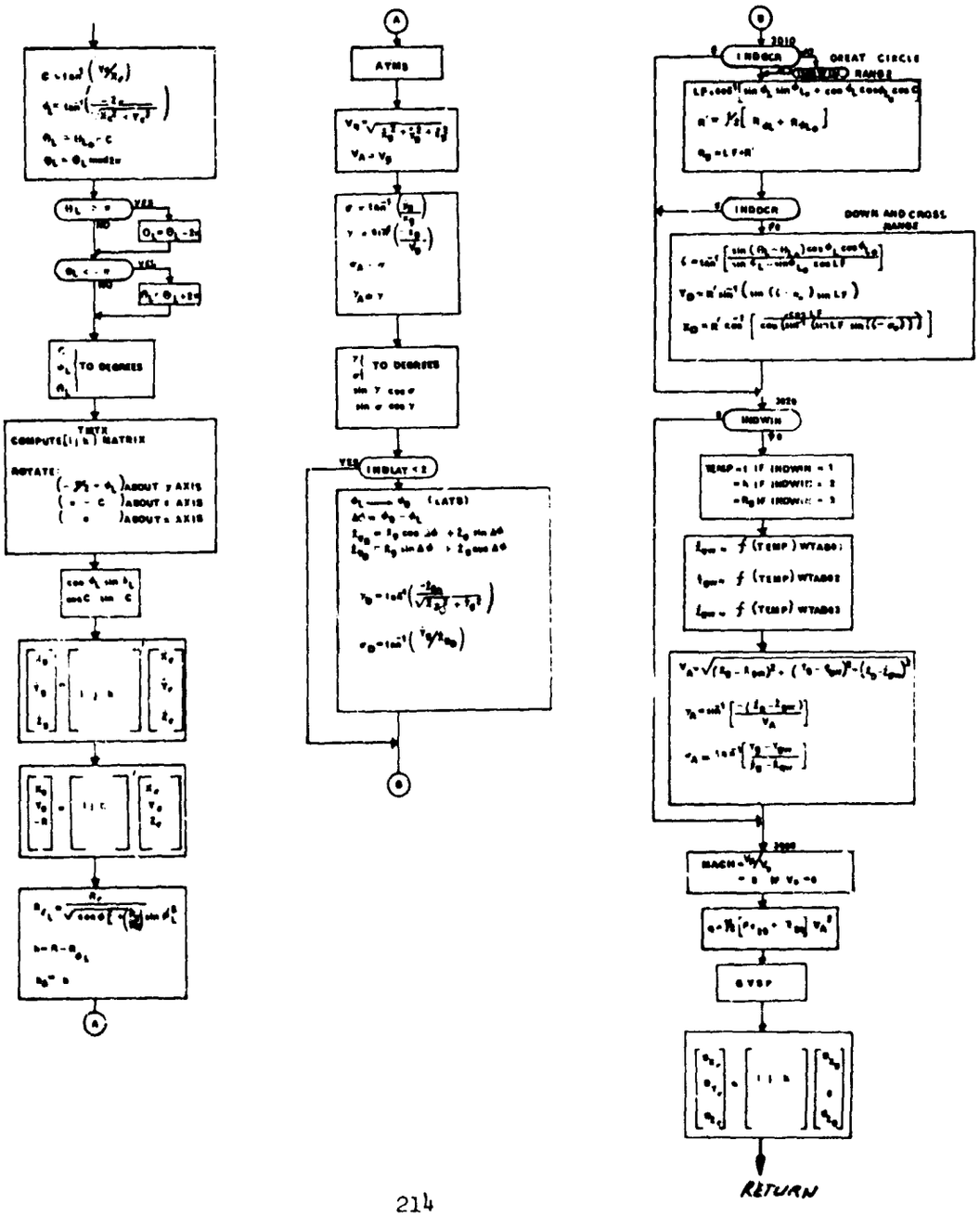
Purpose:

Perform that part of the equation of motion derivative calculation which precedes the vehicle control definition.

Remarks:

A flow chart for DEQBCI is presented; DEQBCI2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.

DEQBCI



48. FPPS and FPPS2 - Flight Plan Programmer

Purpose:

To provide an alternative for computing a variable (which would otherwise be a control variable) as a tabular function of any variable already computed.

Method:

It is legitimate to use FPPS only to compute those variables which are allowed to be control variables. Up to six different control type variables may be computed through the use of FPPS. FPPS is a subprogram of DIFEQ1; it has the standard entry points. The data for FPPS may be changed at any major stage at which DIFEQ1(2) is called, FPPS will be active only if there is data for it and only if INDFLT is input non-zero.

Example

To compute α = f (Mach) the function f is defined pointwise by a two dimensional table (in the standard manner for setting up tables). It is imperative that Mach have been computed before FPPS is called to evaluate α.

When using FPPS, it is imperative to check the coding to be sure that the independent variable of the "f" function has actually been computed at the time FPPS 3) is called; if such is not the case then it is sometimes possible to permute the order of calculations in such a manner that this criteria will be satisfied.

Input Data

This data must go in the stage data. The data may be inserted at any stage at which the initial transformation is executed.

Example

FPPIV	BCD	2SIG7D^{b}AMACH
FPPDV	BCD	2BA77D^{b}ALPHD
FTABO1		4,-90.,180.,-.1,180.
FTABO2		2,0.,10.,2.,5.
INDFLT		1

FPPIV defines the names (at most 6) of the respective independent variables for the tables FTABO1 through FTABO6.

FPPDV — defines the names of the respective dependent variables for the tables FTAB01 through FTAB06.

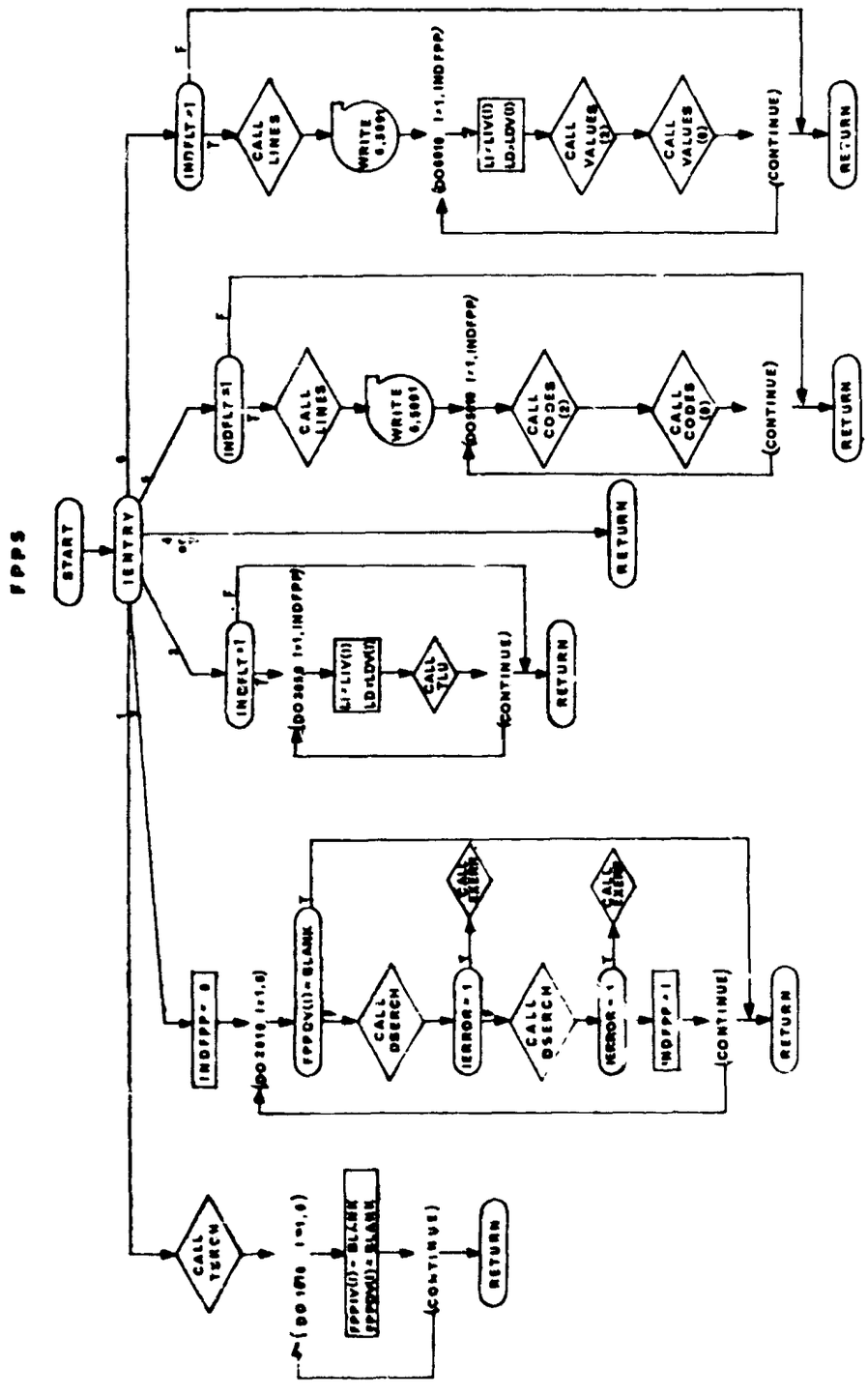
FTAB01 : FTAB06 — two dimensional tables corresponding to the definitions set up in FPPIV and FPPDV.

INDFLT ≠ 0 turns on FPPS
= 0 turns off FPPS

Great care must be exercised in how FPPS is used.

Remarks:

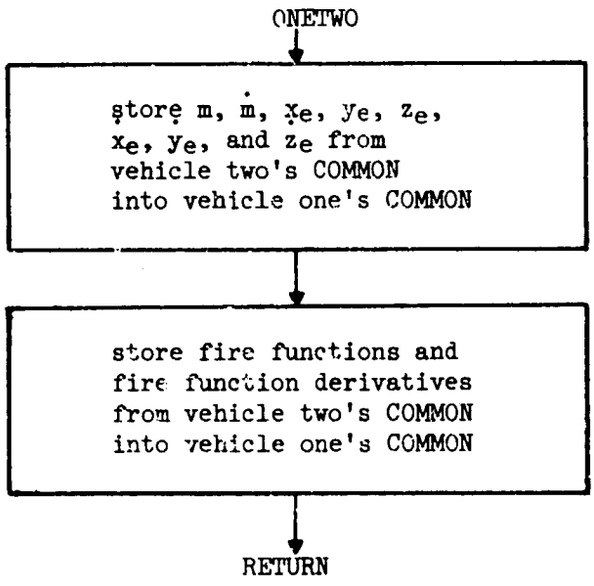
A flow chart for FPPS is presented. FPPS2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.


FPPS
START
IENTRY
RETURN
INDFLT=1
CALL LINES
WRITE
DO 6010 I=1,INDFPP
LI=LIV(I)
LD=LDV(I)
CALL VALUES (3)
CALL VALUES (6)
CONTINUE
INDFPP=0
DO 2010 I=1,6
FPPDV(I)=BLANK
CALL BSERCH
IERROR=1
INDFPP=I
CALL CODES (3)
CALL CODES (6)
CALL TERCH
FPPIV(I)=BLANK
FPPDV(I)=BLANK

49. ONETWO - Transformation of Selected Vehicle 2 Variable to Vehicle 1 COMMON

Purpose:

ONETWO takes variables having a unique name in vehicle 2's COMMON and transfers the variable value to a specified location in vehicle 1's COMMON. The routine is used for the inverse TWOONE transformation when vehicle 1's perturbation equations are employed to perturb a vehicle 2 function, or when a variational problem is terminated. Additional variable transformations may be introduced by simply adding the transformations to *both* ONETWO and TWOONE.



50. CTLITR and CTLITR2 - Control Dependent Derivative Calculation

Purpose:

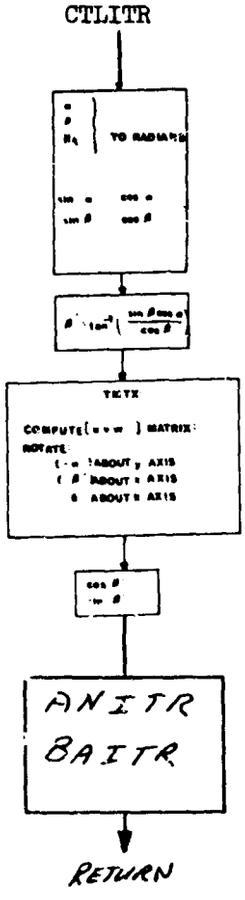
CTLITR and CTLITR2 control the calculation of the portion of the derivative calculation which depends on the instantaneous control variable values.

Method:

Angle-of-attack and sideslip functions including the (u,v,w) matrix are computed directly. Throttle effects are computed through ANITR or ANITR2. Bank angle effects are computed in BAITR or BAITR2.

Remarks:

A flow chart for CTLITR is provided. CTLITR2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



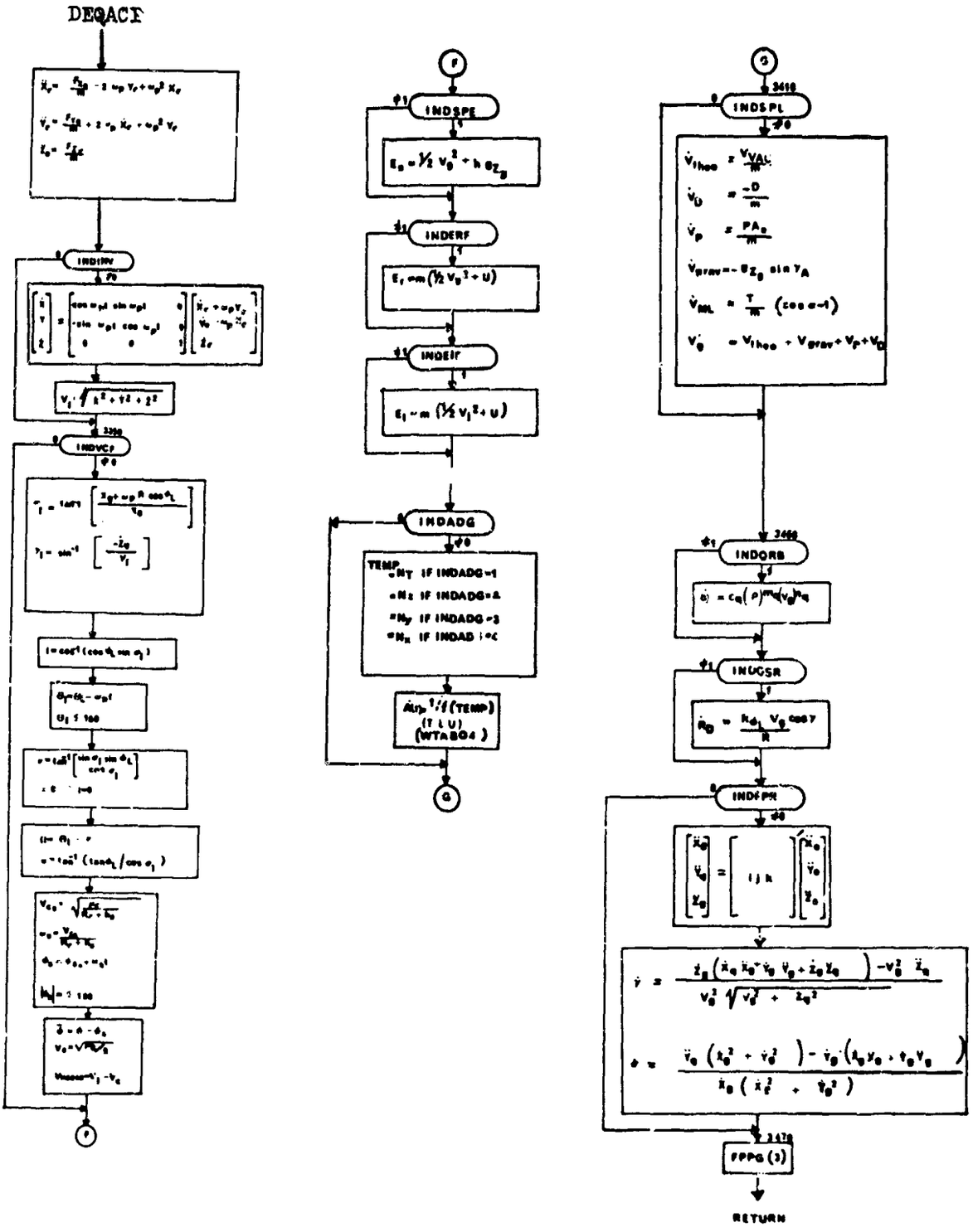
51. DEQACI and DEQACI2 - Derivative Calculation after Control Definition

Purpose:

To carry out secondary derivative calculations following computation of the control dependent functions (e.g., $\dot{\gamma}$ and $\dot{\sigma}$ computations).

Remarks:

A flow chart for DEQACI is presented. DEQACI2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.


DEQACF
INDIRV
INDVCP
INDSPE
INDERF
INDEIF
INDADG
INDSPL
INDQRB
INDGSR
INDFPR
FPPG (3)
RETURN

52. FPPG and FPPG2 - Gamma Command Flight Plan Programmer

Purpose:

To provide the capability for flying a specified flight path angle (γ) time history by computation of the angle of attack,

Method:

The change in angle of attack is computed to correspond to the second term of the Taylor expansion of α (γ) plus a small correction term.

$$\alpha = \alpha_i + C\ ((\gamma_c - \gamma) + (\dot{\gamma}_c - \dot{\gamma}))$$

where

$$C = m\ V_A/(T \cos \alpha + L/\gamma)$$

$$\dot{\gamma}_c = \left[(\gamma_c)_t - (\gamma_c)_{t\ -\ \Delta t}\right]/\Delta t$$

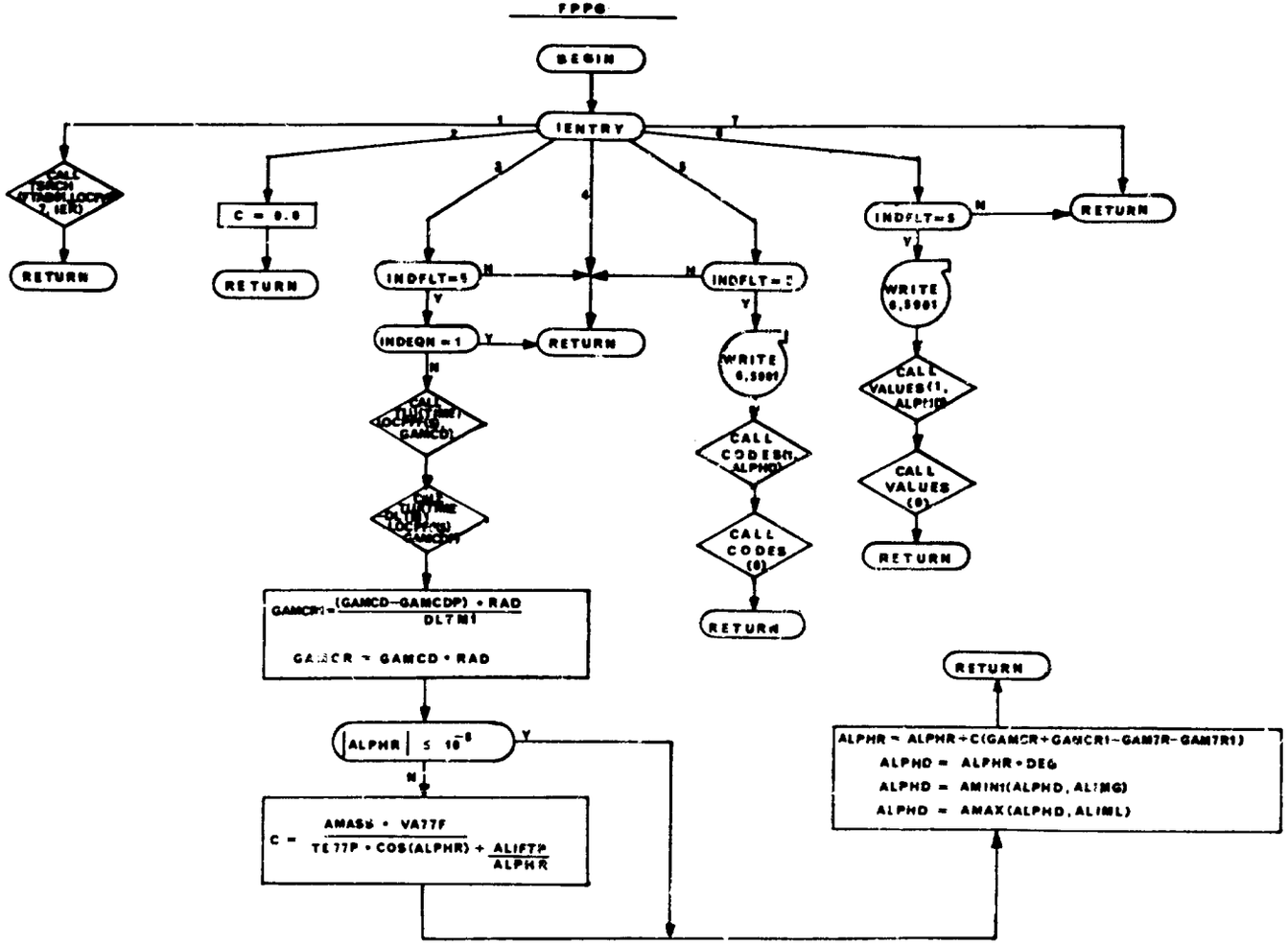
Δt is set equal to 1. second by the program.

α_i is either an initial input value of α, or the value remaining from previous computation at the entry into FPPG(3).

α is then limited between a maximum value (ALIMG) set equal to 24° by the program, and a lower limit (ALIML) set equal to -4° by the program.

Remarks:

A flow chart for FPPG is presented. FPPG2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.


FPPG
BEGIN
IENTRY
RETURN
C = 0.0
RETURN
INDFLT=5
INDEQN = 1
RETURN
INDFLT=5
WRITE 6,5901
CALL CODES(1, ALPHD)
CALL CODES (0)
RETURN
INDFLT=5
RETURN
WRITE 6,5901
CALL VALUES(1, ALPHD)
CALL VALUES (0)
RETURN
GAMCR1 = (GAMCD−GAMCDP) • RAD / DLTM1
GAMCR = GAMCD • RAD
|ALPHR| ≤ 10^-8
C = AMASS • VA77F / TE77P • COS(ALPHR) + ALIFTP / ALPHR
RETURN
ALPHR = ALPHR+C(GAMCR+GAMCR1−GAM7R−GAM7R1)
ALPHD = ALPHR • DEG
ALPHD = AMIN1(ALPHD, ALIMG)
ALPHD = AMAX(ALPHD, ALIML)

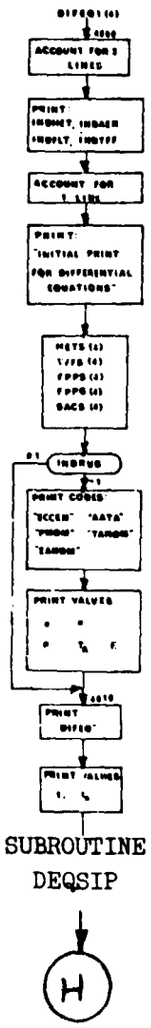
53. DEQSIP and DEQSIP2 - Derivative Evaluation Initial Point

Purpose:

To provide an initial print heading for the differential equations of motion output.

Remarks:

A flow chart for DEQSPI is provided. DEQSPI2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



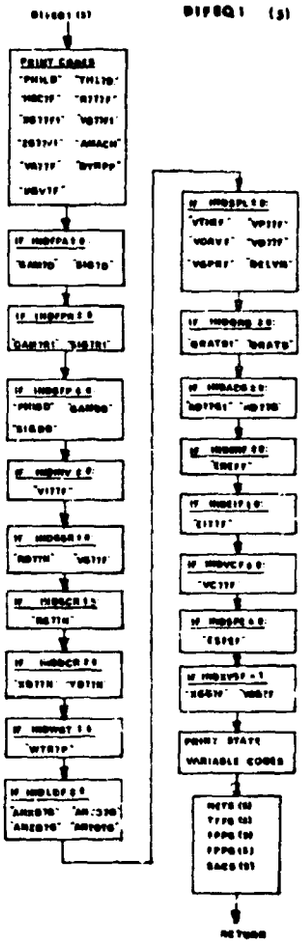
54. DEQCOD and DEQCOD2 - Trajectory Code Print

Purpose:

To provide a trajectory history code print for the selected output variables.

Remarks:

A flow chart for DEQCOD is presented. DEQCOD2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



SUBROUTINE DEQCOD

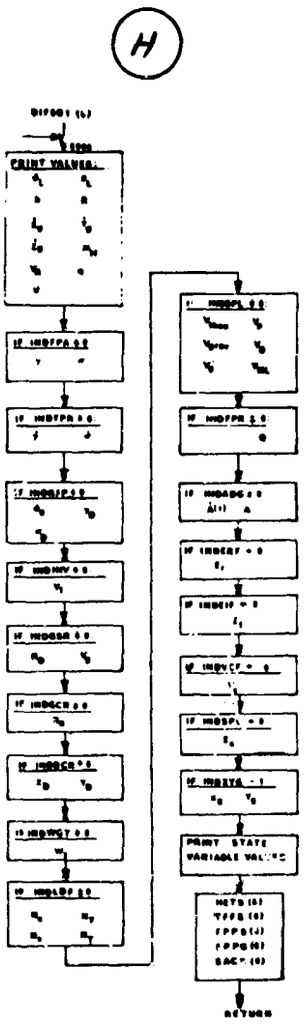
55. DEQVAL and DEQVAL2 - Trajectory History Print

Purpose:

To provide a history of selected variable values along a trajectory.

Remarks:

A flow chart for DEQVAL which generates the first vehicle output values is presented. DEQVAL2 which provides the second vehicle output values is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



SUBROUTINE DEQVAL

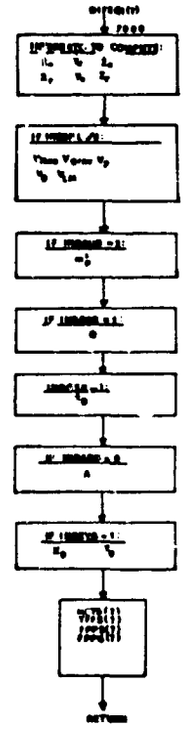
56. DEQIV and DEQIV2 - Integrated Variable Specification

Purpose:

To define active integrated variables.

Remarks:

A flow chart for DEQIV is presented. DEQIV2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



SUBROUTINE DEQIV

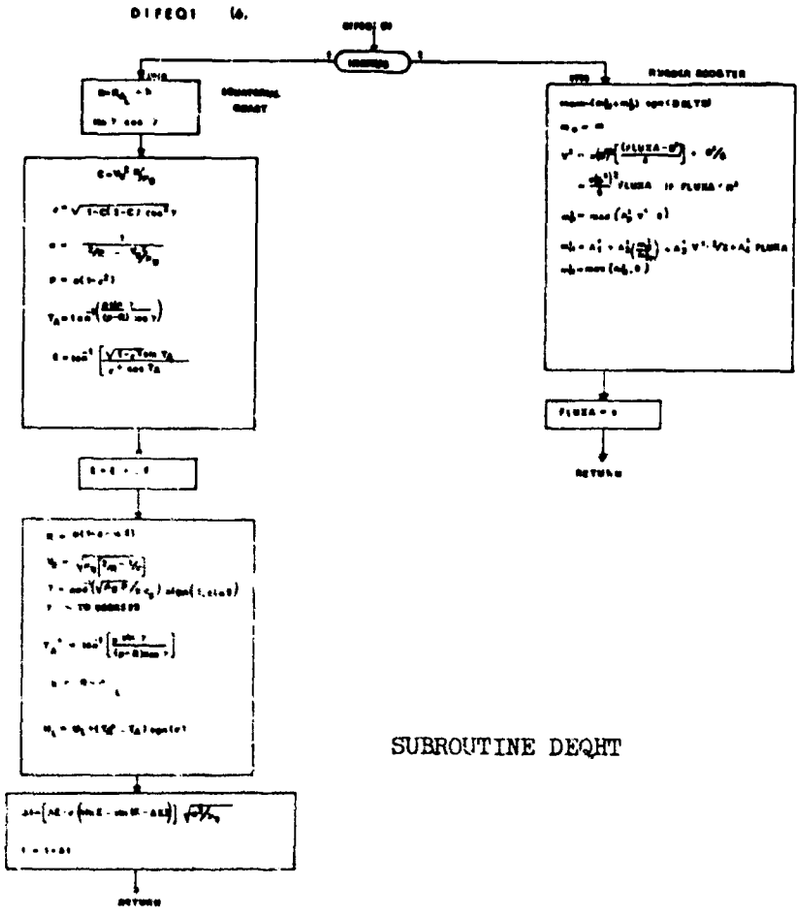
57. DEQHT and DEQHT2 - Trajectory H-Transformation Subroutines

Purpose:

To carry out specified h-transformations at selected stage points.

Remarks:

A flow chart for DEQHT is presented. DEQHT2 is identical except for the use of vehicle 2 COMMON blocks.



SUBROUTINE DEQHT

NOT REPRODUCIBLE

58. ERROR and ERROR2 - General Table Error Routine

Purpose

To provide a method of indicating the table which may possibly contain an error.

Method

Given the subscript of the curve in error, the routine will search the subscript table and find the corresponding BCD word. This word will then be printed as:

"TABLE ERROR AAAAAA"

where AAAAAA is the BCD name of table. If the name cannot be found in the directory

"TABLE ERROR

... Location of table not listed in directory..."

is printed and a return to the calling program is made. In either case INDSTE is set to zero.

Usage

Entry is made to the routine with the following statement:

CALL ERROR (LOCZ)

where LOCZ is the table subscript.

Subroutines Called

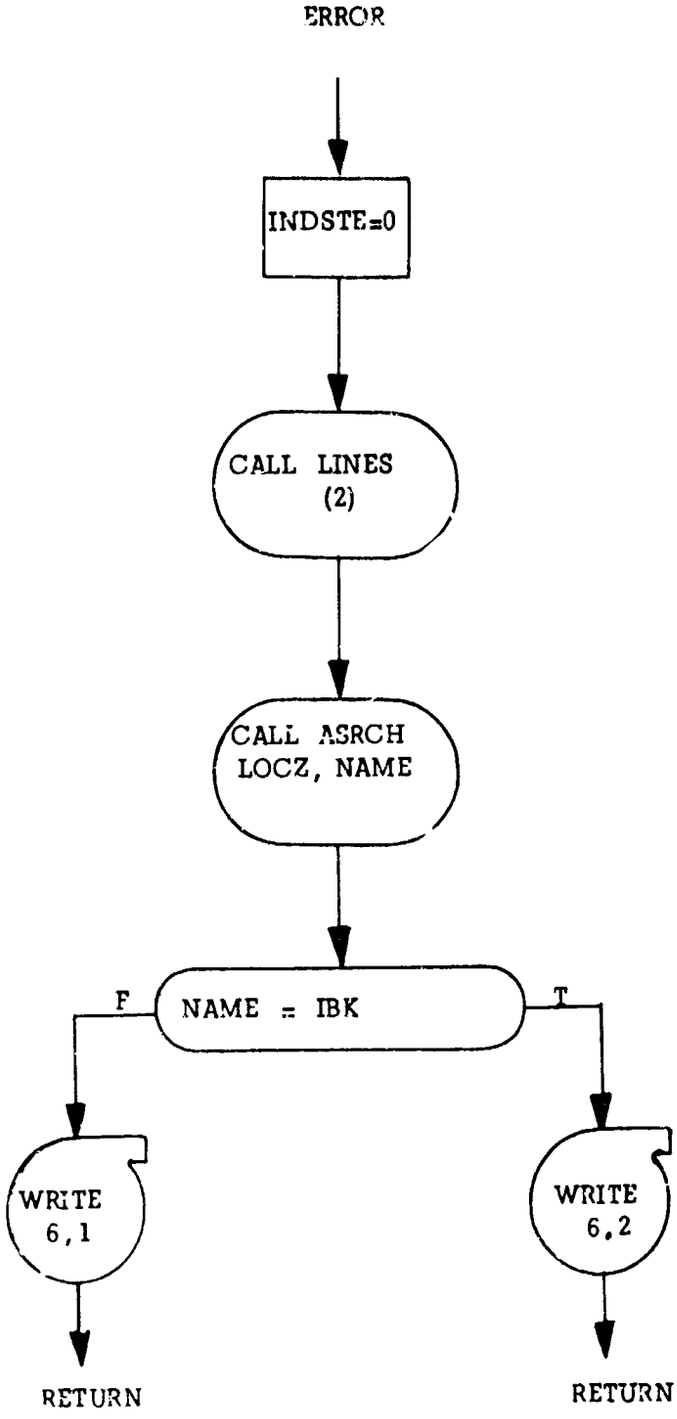
LINES

ASRCH

Normal FORTRAN I/O routines.

Remarks

A flow chart for ERROR is provided. ERROR2 is identical to ERROR except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.


ERROR
INDSTE=0
CALL LINES
(2)
CALL ASRCH
LOCZ, NAME
F
NAME = IBK
T
WRITE
6,1
WRITE
6,2
RETURN
RETURN

59. PTBEQN and PTBEQN2 - Driver Routines for Equations

Purpose:

To duplicate the sequence of calls to subprograms made by EXE for entry points 2 and 3.

Usage:

CALL PTBEQN (IENTRY)

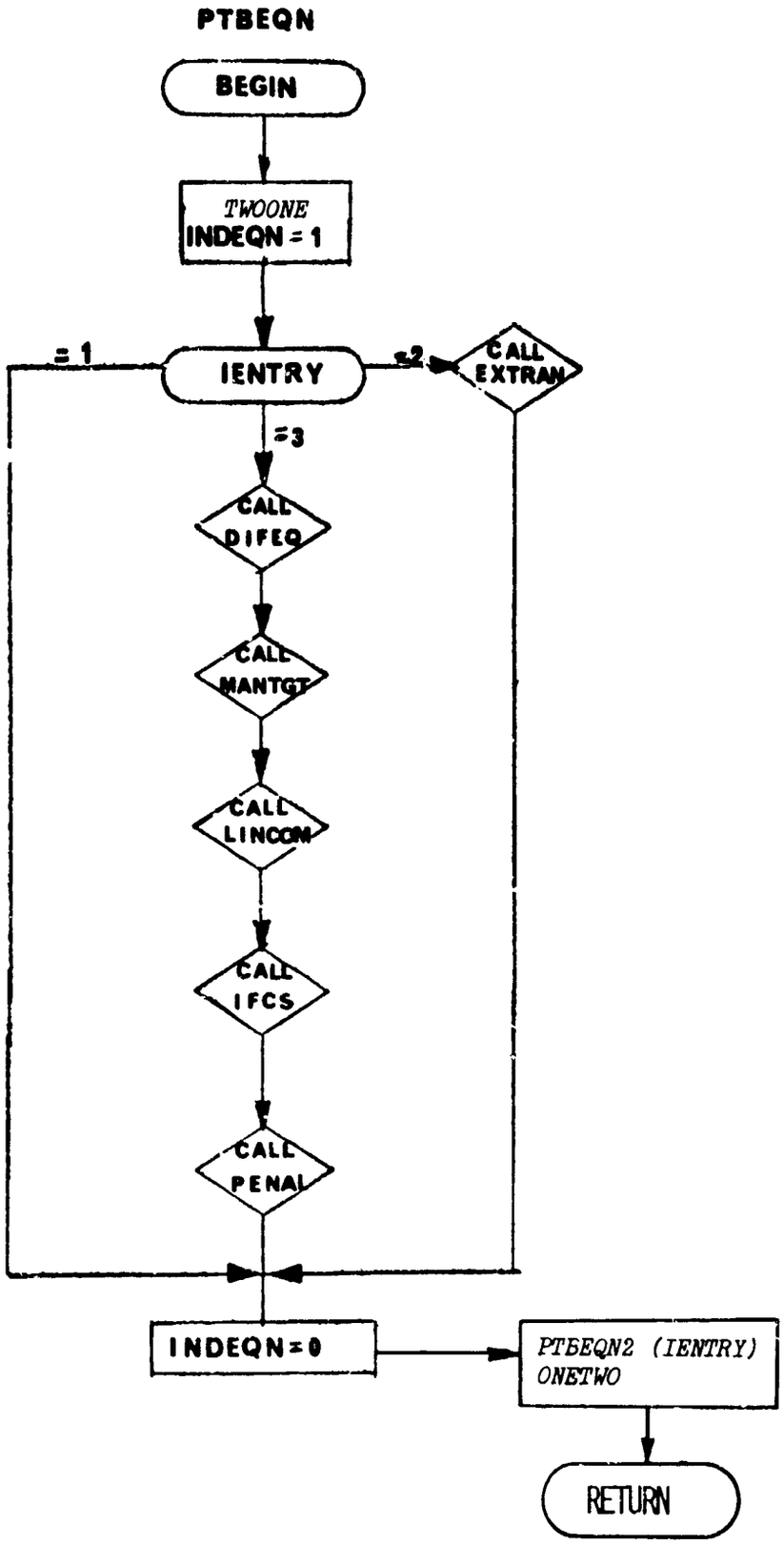
IENTRY = 2 make calls to entry points 2

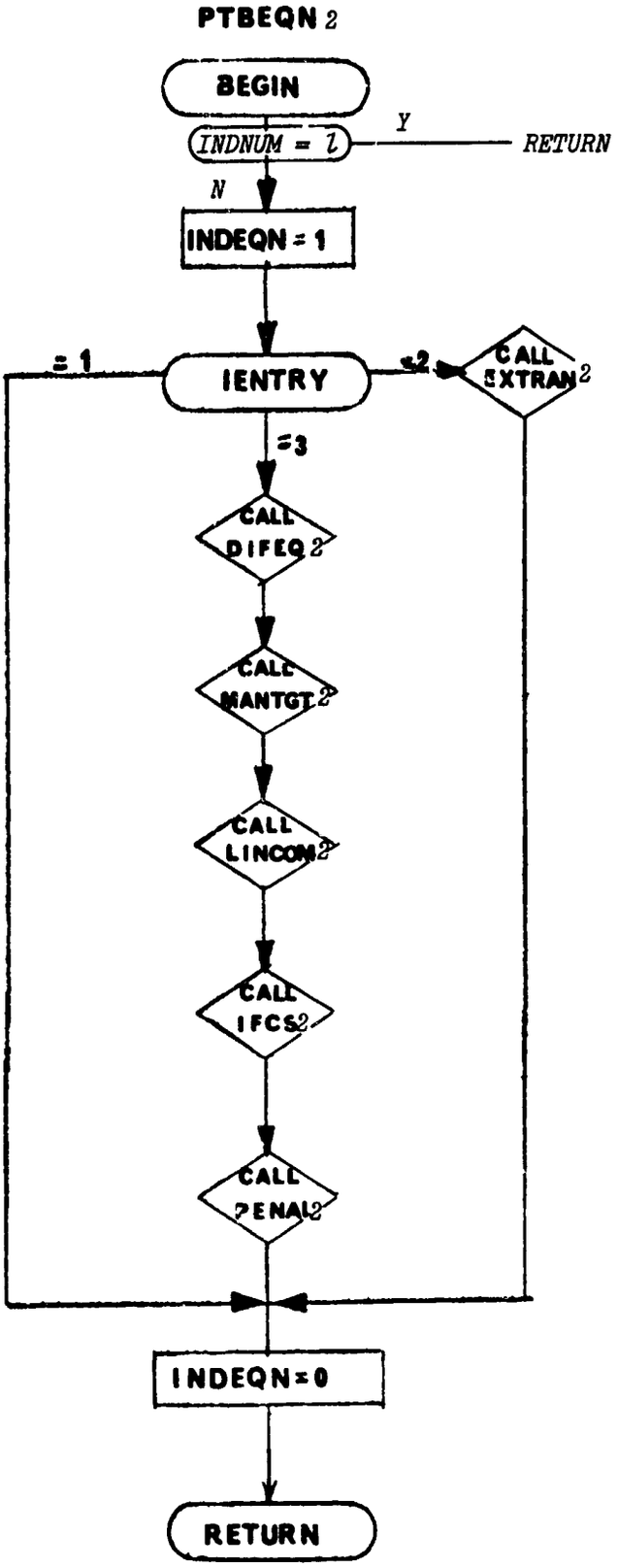
IENTRY = 3 make calls to entry points 3

PTBEQN is called only by the PSUBR routine. IENTRY is an argument in the call to PSUBR from PARTS; PSUBR transmits it to PTBEQN. PTBEQN may be looked upon as that routine which drives the particular functional calculation necessary for computing partial derivatives numerically. The indicator INDEQN is set to 1 upon entering PTBEQN and is set to 0 upon leaving; this indicator is not essential to any of the logic but is designed only for diagnostic purposes.

When IENTRY = 2, PTBEQN only calls EXTRAN (3). EXTRAN(3) takes care of evaluating the functional calculation for the initial transformation of major stages (either initial conditions or n-transformation).

PTBEQN controls the transformation of selected vehicle 2 functions into unique vehicle 1 functions for use in two vehicle variational optimization problems. This is achieved by use of the transformation subroutines TWOONE and ONETWO. The selected vehicle 2 functions are perturbed by PTBEQN2 which is called from PTBEQN. Flow charts for both PTBEQN and PTBEQN2 are presented.

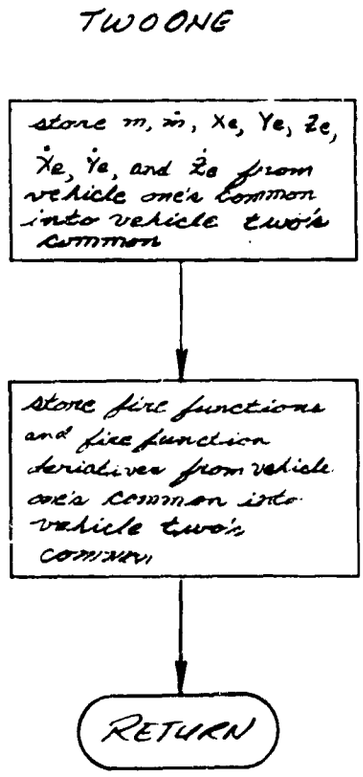

PTBEQN
BEGIN
TWOONE
INDEQN = 1
IENTRY
= 1
=2
CALL EXTRAN
=3
CALL DIFEQ
CALL MANTGT
CALL LINCOM
CALL IFCS
CALL PENAL
INDEQN=0
PTBEQN2 (IENTRY)
ONETWO
RETURN


PTBEQN 2
BEGIN
INDNUM = 1
Y
RETURN
N
INDEQN = 1
IENTRY
= 1
=2
CALL EXTRAN 2
=3
CALL DIFEQ 2
CALL MANTGT 2
CALL LINCOM 2
CALL IFCS 2
CALL PENAL 2
INDEQN=0
RETURN

60. TWOONE - Transformation of Selected Vehicle 1 Variable to Vehicle 2 COMMON

Purpose:

TWOONE takes variables having a unique name in vehicle 1's COMMON and transfers the variable value to a specified location in vehicle 2's COMMON. The routine is used for the inverse ONETWO transformation when vehicle 1's perturbation equations are employed to perturb a vehicle 2 function. Additional variable transformations may be introduced as discussed in the ONETWO write-up.



61. IZERO and IZERO2 - Packs Non-Zero Numbers

Purpose

Test the argument for non-zero and set up indicators for packing.

Method

The argument is tested for zero and a switch is set for each word. The switch is on for zero and off for non-zero.

Usage

Entry is made to the routine with the following statement:

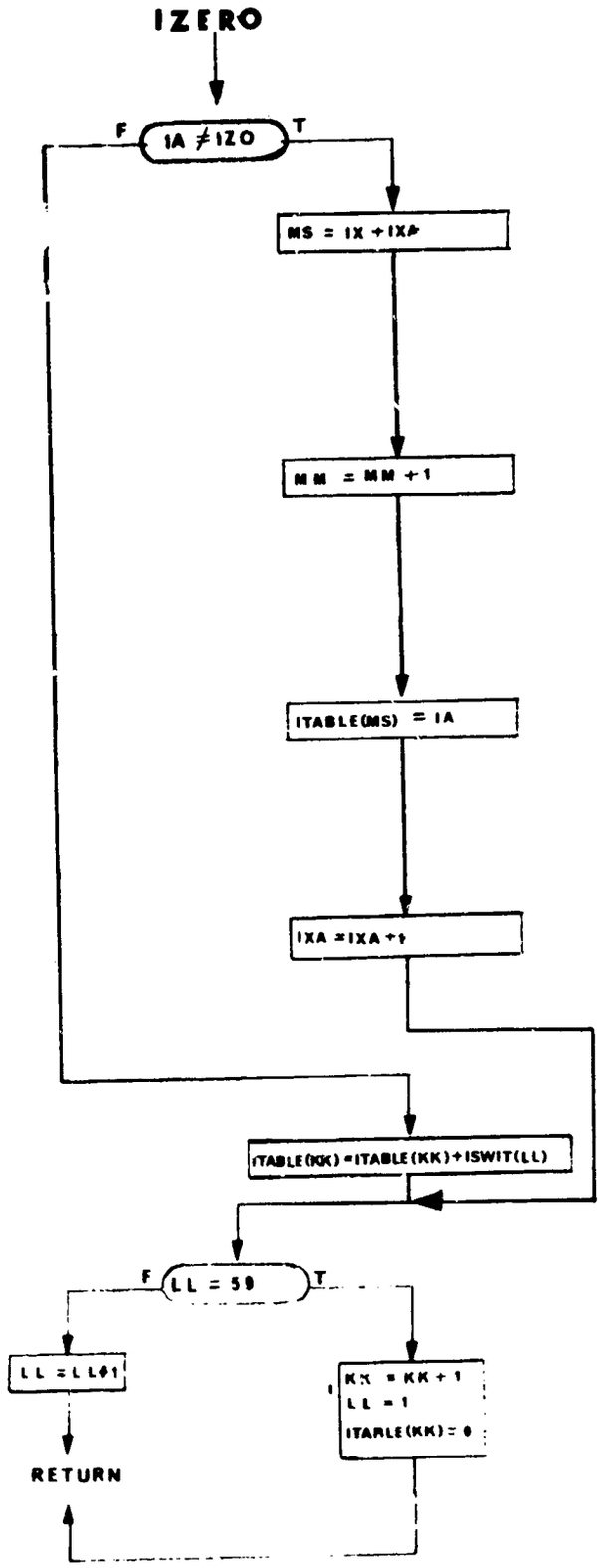
CALL IZERO(IA)

where

IA = The number to be packed.

Remarks

No other subroutines are called from this routine.


IZERO
F
IA ≠ IZO
T
MS = IX + IXA
MM = MM + 1
ITABLE(MS) = IA
IXA = IXA + 1
ITABLE(KK) = ITABLE(KK) + ISWIT(LL)
F
LL = 59
T
LL = LL + 1
KK = KK + 1
LL = 1
ITABLE(KK) = 0
RETURN

62. PPLNLN - Main Paper-Plot Routine

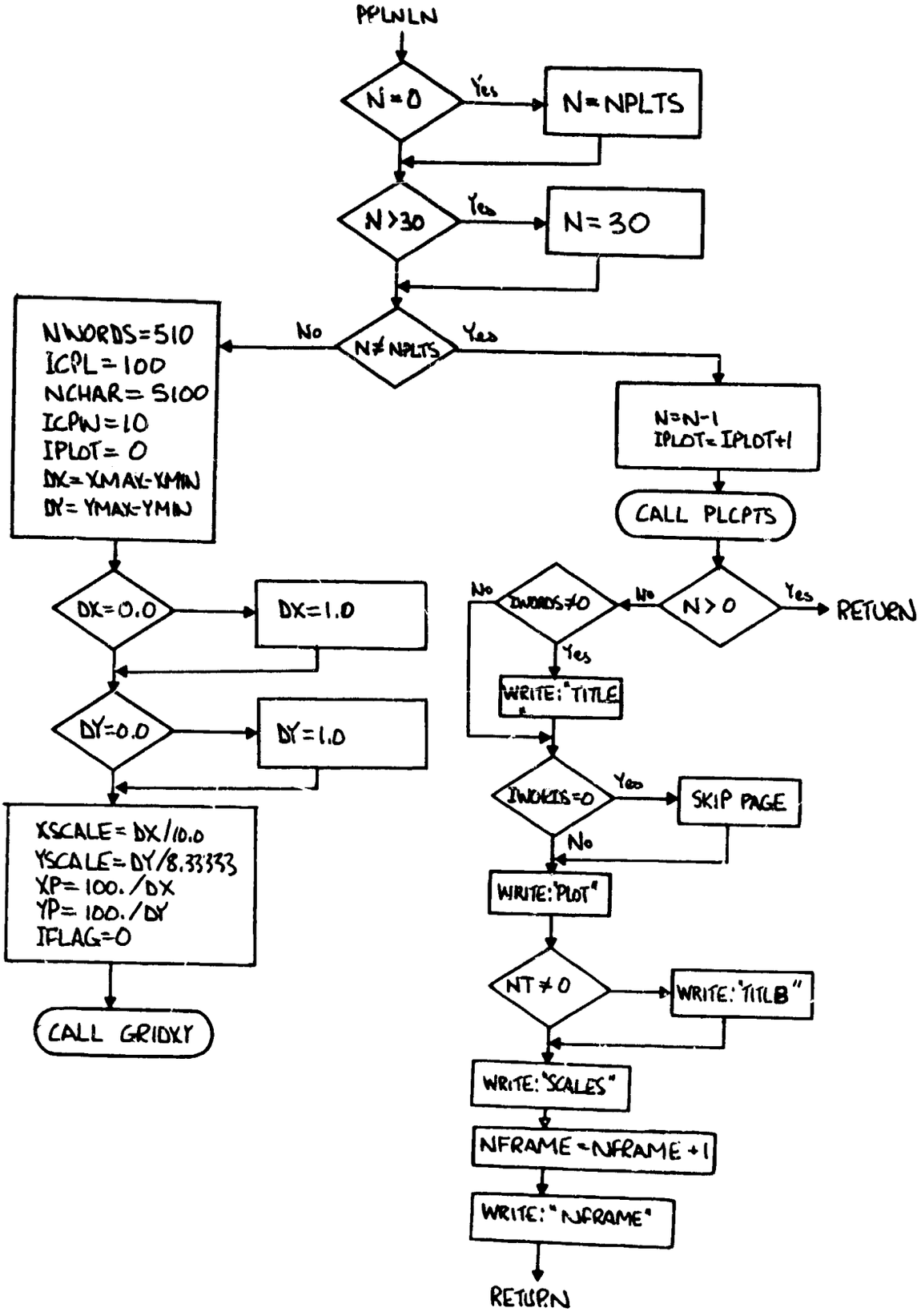
Purpose:

To control scale size, construct grids, plot points, and title each paper plot.

Usage:

```
Call PPLNLN (X, Y, NPTS, XMAX, XMIN, YMAX, YMIN, NPLTS,
     TITLE, IWORDS, TITLEB, NT)
```

x	is the array that contains the values of the independent variable to be plotted
y	is the array that contains the values of the dependent variable to be plotted
NPTS	number of points to be plotted
XMAX	maximum value of plot grid for x axis
XMIN	minimum value of plot grid for x axis
YMAX	maximum value of plot grid for y axis
YMIN	minimum value of plot grid for y axis
NPLTS	number of curves to be plotted on this frame
TITLE	title to be printed at the top of the plot
IWORDS	number of ten character words in TITLE
TITLEB	title to be printed at the bottom of the plot
NT	number of ten character words in TITLEB


PPLNLN
N=0
Yes
N=NPLTS
N>30
Yes
N=30
No
N≠NPLTS
Yes
NWORDS=510
ICPL=100
NCHAR=5100
ICPW=10
IPLOT=0
DX=XMAX-XMIN
DY=YMAX-YMIN
N=N-1
IPLOT=IPLOT+1
CALL PLCPTS
DX=0.0
DX=1.0
DY=0.0
DY=1.0
XSCALE=DX/10.0
YSCALE=DY/8.33333
XP=100./DX
YP=100./DY
IFLAG=0
CALL GRIDXY
No
IWORDS≠0
No
N>0
Yes
RETURN
Yes
WRITE:'TITLE'
IWORDS=0
Yes
SKIP PAGE
No
WRITE:'PLOT'
NT≠0
WRITE:'TITLB'
WRITE:'SCALES'
NFRAME=NFRAME+1
WRITE:'NFRAME'
RETURN

63. SENSOR and SENSOR2 - Vehicle Sensor Routines

Purpose:

To supply each vehicle with a system of sensors for detection of an opponent. Each vehicle may employ up to seven independent sensors.

Usage:

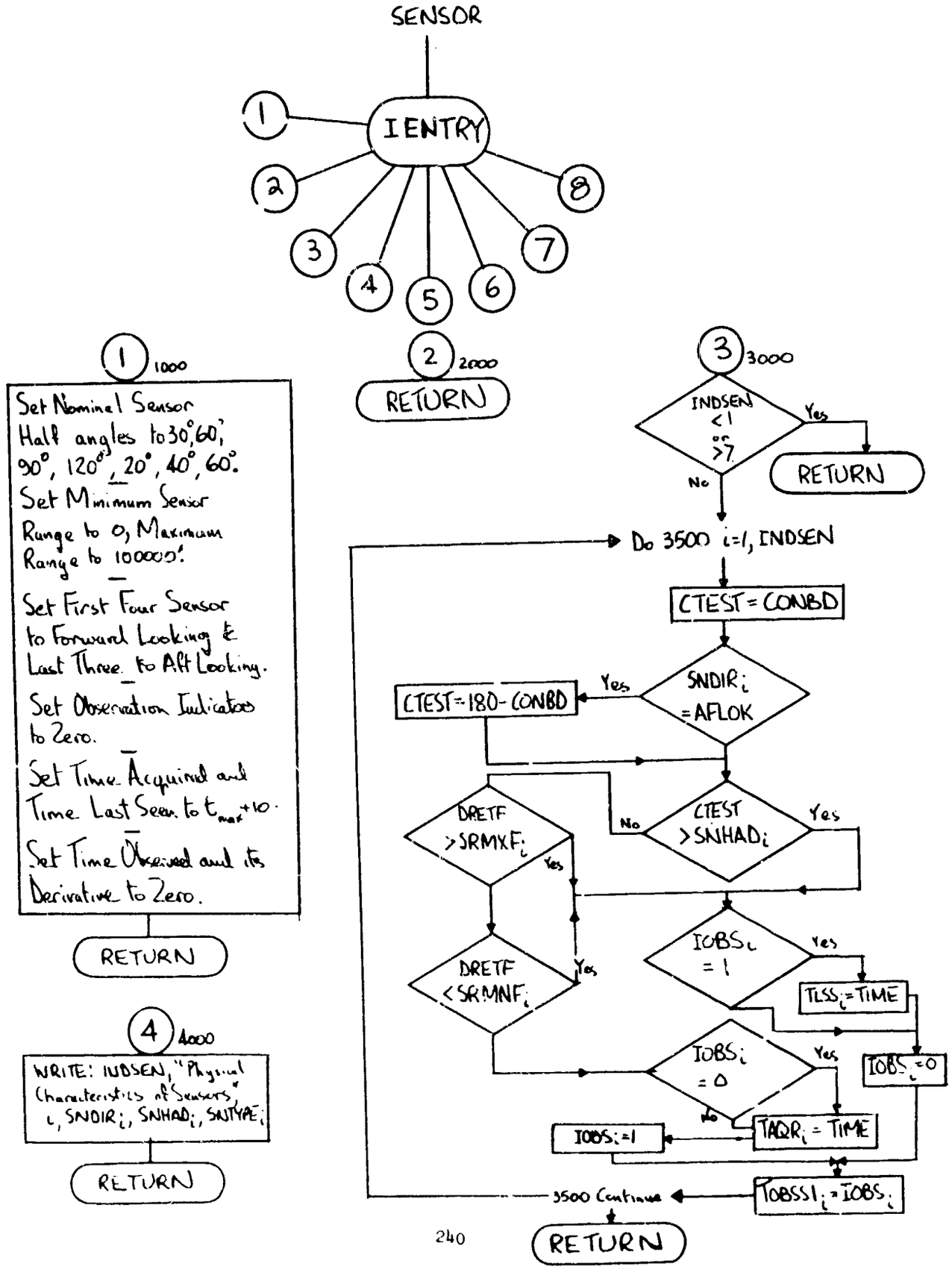
Given the opponent's position each sensor is checked sequentially to see whether or not the opponent can be observed. In the present program an opponent is observed when the target's cone angle (in body axes) is less than the sensor half angles and the target's range lies between minimum and maximum limits. The time each sensor observes an opponent is integrated and the time at which a sensor loses an opponent is preserved.

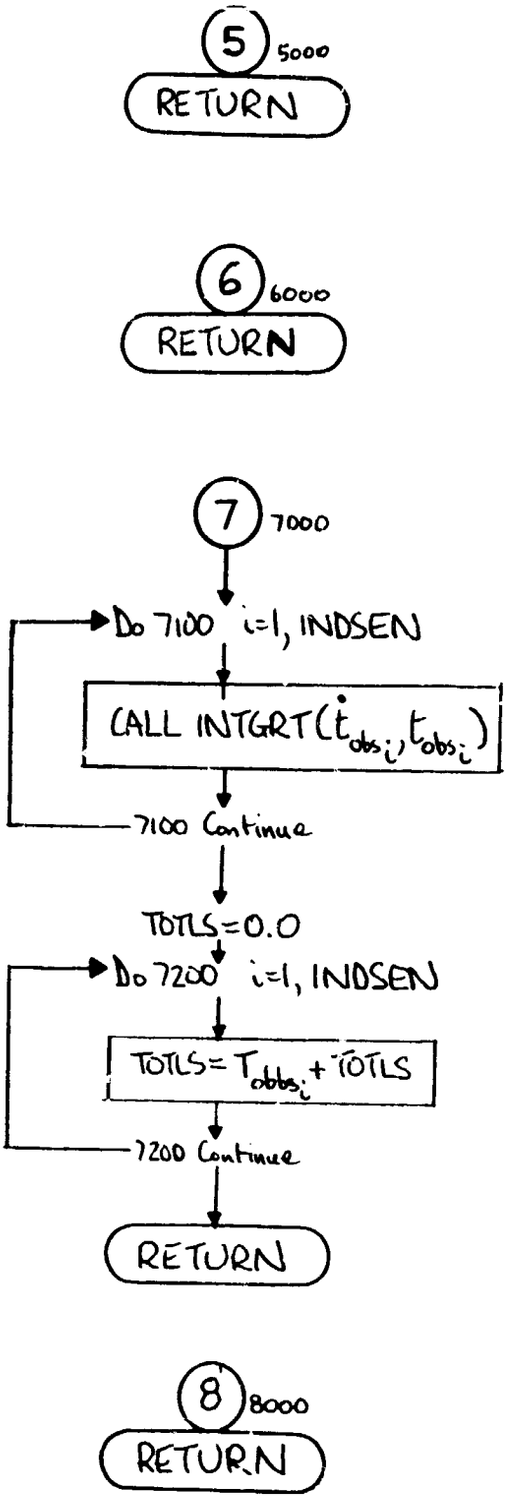
A call to SENSOR or SENSOR2 is made as follows:

CALL SENSOR (IENTRY) or CALL SENSOR2 (IENTRY)

IENTRY is the standard EXE entry point indicator.

A flow chart for SENSOR is provided. SENSOR2 is identical to SENSOR except for the use of vehicle 2 COMMON blocks.


SENSOR
IENTRY
1
2
3
4
5
6
7
8
1 1000
Set Nominal Sensor Half angles to 30°, 60°, 90°, 120°, 20°, 40°, 60°.
Set Minimum Sensor Range to 0, Maximum Range to 100000'.
Set First Four Sensor to Forward Looking & Last Three to Aft Looking.
Set Observation Indicators to Zero.
Set Time Acquired and Time Last Seen to t_{max}+10.
Set Time Observed and its Derivative to Zero.
RETURN
2 2000
RETURN
3 3000
INDSEN <1 or >7
Yes
RETURN
No
Do 3500 i=1, INDSEN
CTEST = CONBD
SNDIR$_i$ = AFLOK
Yes
CTEST = 180 - CONBD
DRETF > SRMXF$_i$
Yes
No
CTEST > SNHAD$_i$
Yes
DRETF < SRMNF$_i$
Yes
IOBS$_i$ = 1
Yes
TLSS$_i$ = TIME
IOBS$_i$ = 0
Yes
IOBS$_i$ = 0
No
IOBS$_i$ = 1
TAQR$_i$ = TIME
TOBSS1$_i$ = IOBS$_i$
3500 Continue
RETURN
4 4000
WRITE: INDSEN, "Physical Characteristics of Sensors", i, SNDIR$_i$, SNHAD$_i$, SNTYPE$_i$
RETURN


5
5000
RETURN
6
6000
RETURN
7
7000
Do 7100 i=1, INDSEN
CALL INTGRT($\dot{t}_{obs_i}$, t_{obs_i})
7100 Continue
TOTLS=0.0
Do 7200 i=1, INDSEN
TOTLS=T_{obbs_i}+TOTLS
7200 Continue
RETURN
8
8000
RETURN

64. VISION and VISION2 - Pilot Vision Routines

Purpose:

To supply each pilot with a visual detection model. Visual scans are made at specified time intervals.

Usage:

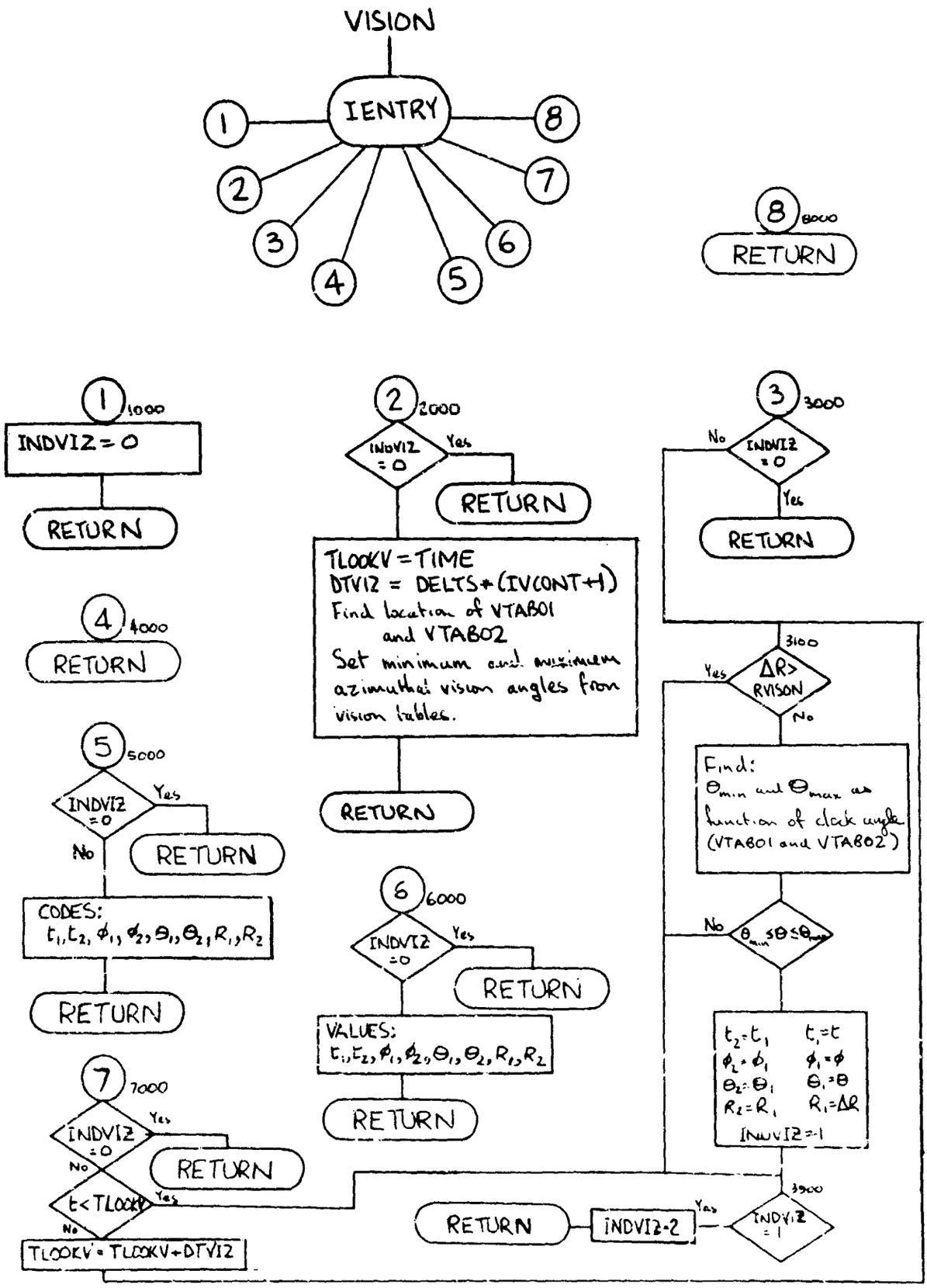
Given the opponent's position a periodic visual check is made to see if the opponent lies between tabular upper and lower elevation limits as a function of target azimuth. If the opponent lies in these elevation limits and is in visual range, it is assumed that he will be detected. A flag is set whenever an opponent is lost from sight and at all times the last two visual sightings are recorded in time, azimuth, elevation, and range senses.

A call to VISION or VISION2 is made as follows:

CALL VISION (IENTRY) or CALL VISION2 (IENTRY)

IENTRY is the standard EXE entry point indicator.

A flow chart for VISION is presented. VISION2 is identical except for the use of vehicle 2 COMMON blocks.

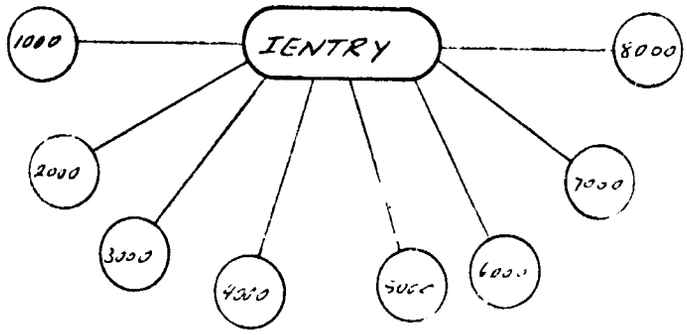

VISION
IENTRY
1
2
3
4
5
6
7
8
8 8000
RETURN
1 1000
INDVIZ = 0
RETURN
2 2000
INDVIZ = 0
Yes
RETURN
TLOOKV = TIME
DTVIZ = DELTS * (IVCONT+1)
Find location of VTAB01 and VTAB02
Set minimum and maximum azimuthal vision angles from vision tables.
RETURN
3 3000
No
INDVIZ = 0
Yes
RETURN
4 4000
RETURN
5 5000
INDVIZ = 0
Yes
RETURN
No
CODES:
$t_1, t_2, \phi_1, \phi_2, \Theta_1, \Theta_2, R_1, R_2$
RETURN
6 6000
INDVIZ = 0
Yes
RETURN
VALUES:
$t_1, t_2, \phi_1, \phi_2, \Theta_1, \Theta_2, R_1, R_2$
RETURN
7 7000
INDVIZ = 0
Yes
RETURN
No
t < TLOOKV
Yes
No
TLOOKV = TLOOKV + DTVIZ
3100
Yes
ΔR > RVISON
No
Find:
Θ_{min} and Θ_{max} as function of clock angle (VTAB01 and VTAB02)
No
$\Theta_{min} \leq \Theta \leq \Theta_{max}$
$t_2 = t_1$ $t_1 = t$
$\phi_2 = \phi_1$ $\phi_1 = \phi$
$\Theta_2 = \Theta_1$ $\Theta_1 = \Theta$
$R_2 = R_1$ $R_1 = \Delta R$
INDVIZ = 1
3900
INDVIZ = 1
Yes
INDVIZ = 2
RETURN

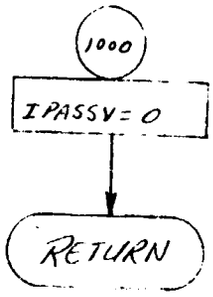
65. PASSV1 and PASSV2 - Passive Tactics Routine

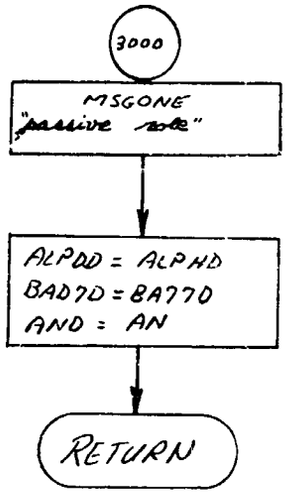
Purpose:

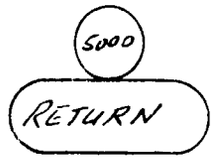
PASSV1 and PASSV2 serve to interface flight plan program options or fixed control history flight paths with the combat logic. In particular, it converts all such control specifications to the finite control rate option at the analyst's request. A flow chart for PASSV1 is presented. PASSV2 is identical except for use of vehicle 2 COMMON blocks.

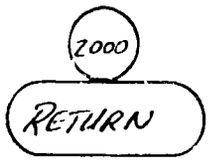
PASSV1

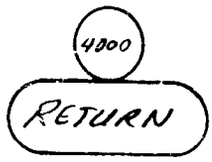

1000
IENTRY
8000
2000
7000
3000
4000
5000
6000


1000
IPASSV = 0
RETURN

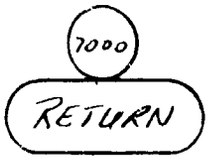

3000
MSGONE
"passive ~~ok~~"
ALPDD = ALPHD
BADJD = BAJJD
AND = AN
RETURN

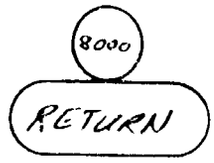

5000
RETURN


2000
RETURN


4000
RETURN

6000
RETURN


7000
RETURN


8000
RETURN

66. DEFEN1 and DEFEN2 - Defensive Tactics Routine

Purpose:

To select each vehicle's defensive tactic.

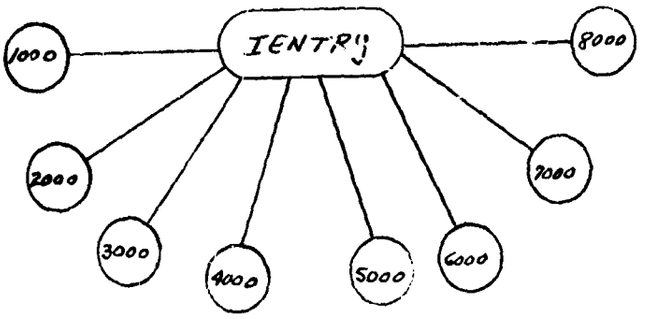
Method:

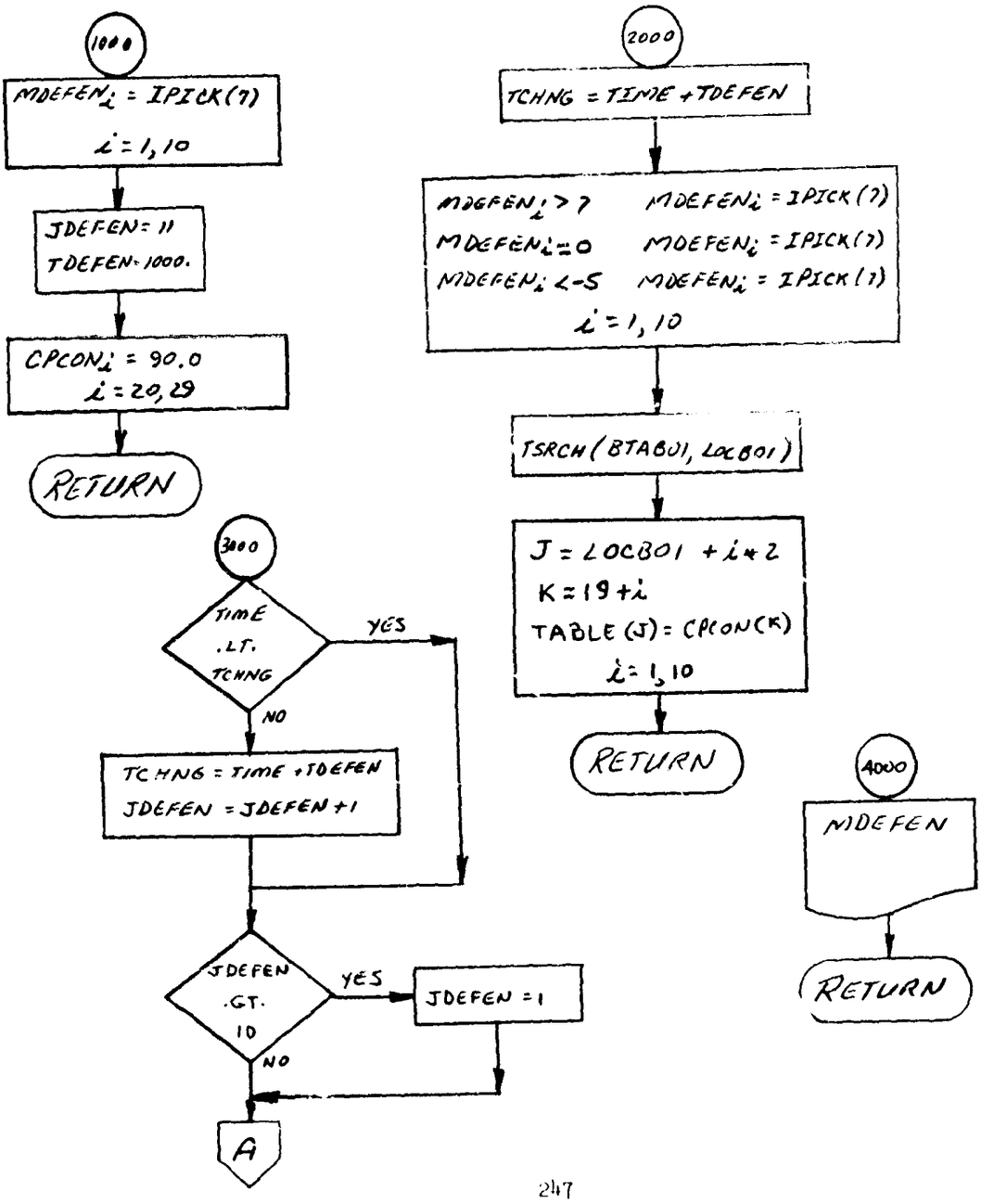
ROLE1 and ROLE2 have selected each vehicle's role. When a defensive role is selected, DEFEN1 and DEFEN2 define a specific defensive tactic for vehicles 1 and 2, respectively. Defensive tactics may be selected in random or ordered manner. An override to another specified role is also possible by use of FIXEDR or FIXEDR2. The analyst may also specify a minimum elapsed time between tactic changes.

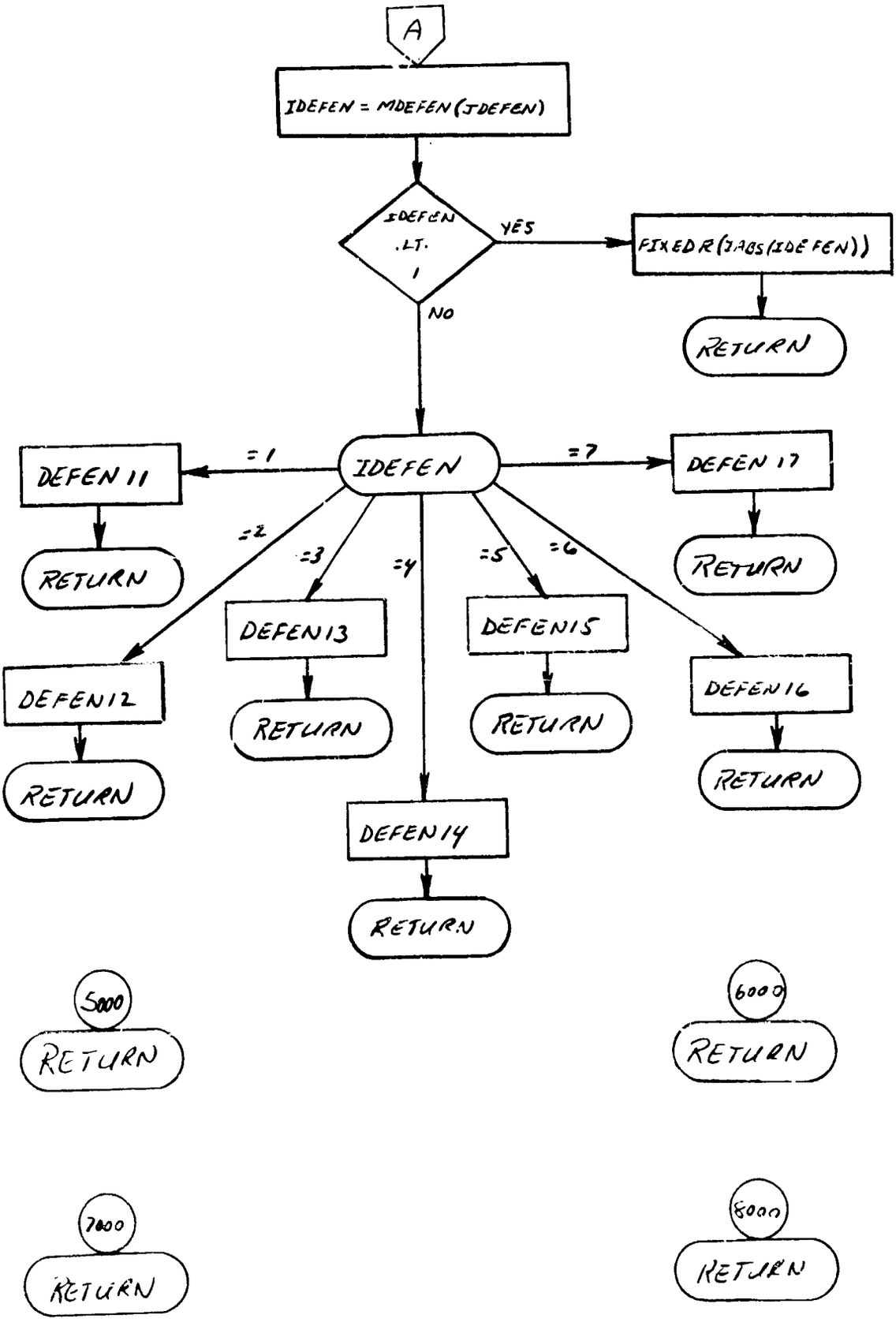
Remarks:

A flow chart for DEFEN1 is presented. DEFEN2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

DEFEN1


IENTRY
1000
2000
3000
4000
5000
6000
7000
8000


1000
MDEFEN$_i$ = IPICK(7)
i = 1,10
JDEFEN = 11
TDEFEN = 1000.
CPCON$_i$ = 90.0
i = 20,29
RETURN
2000
TCHNG = TIME + TDEFEN
MDEFEN$_i$ > 7 MDEFEN$_i$ = IPICK(7)
MDEFEN$_i$ = 0 MDEFEN$_i$ = IPICK(7)
MDEFEN$_i$ < 5 MDEFEN$_i$ = IPICK(7)
i = 1,10
TSRCH(BTAB01, LOCB01)
J = LOCB01 + i + 2
K = 19 + i
TABLE(J) = CPCON(K)
i = 1,10
RETURN
3000
TIME .LT. TCHNG
YES
NO
TCHNG = TIME + TDEFEN
JDEFEN = JDEFEN + 1
JDEFEN .GT. 10
YES
JDEFEN = 1
NO
A
4000
MDEFEN
RETURN


A
IDEFEN = MDEFEN(JDEFEN)
IDEFEN .LT. 1
YES
FIXED R(JABS(IDEFEN))
RETURN
NO
IDEFEN
=1
DEFEN 11
RETURN
=2
DEFEN12
RETURN
=3
DEFEN13
RETURN
=4
DEFEN14
RETURN
=5
DEFENIS
RETURN
=6
DEFEN16
RETURN
=7
DEFEN 17
RETURN
5000
RETURN
6000
RETURN
7000
RETURN
8000
RETURN

67. EVADE1 and EVADE2 - Evasive Tactics Routine

Purpose:

To select each vehicle's evasive tactic.

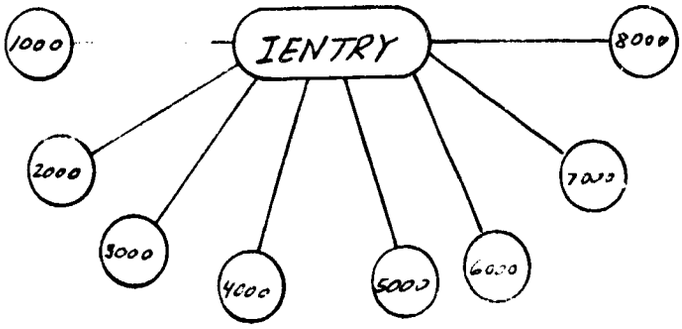
Method:

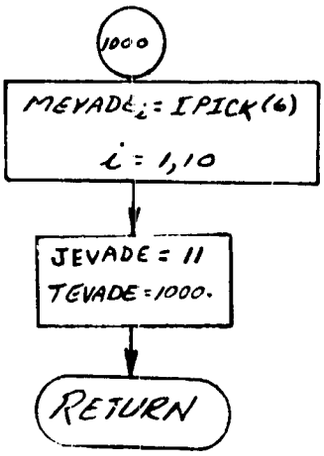
ROLE1 and ROLE2 have selected each vehicle's role. When an evasive role is requested, EVADE1 and EVADE2 define a specific evasive tactic for vehicles 1 and 2. Tactics may be selected in random or ordered fashion. An override to another specified role is possible by use of FIXEDR or FIXEDR2. The analyst may also specify a minimum elapsed time between tactic changes.

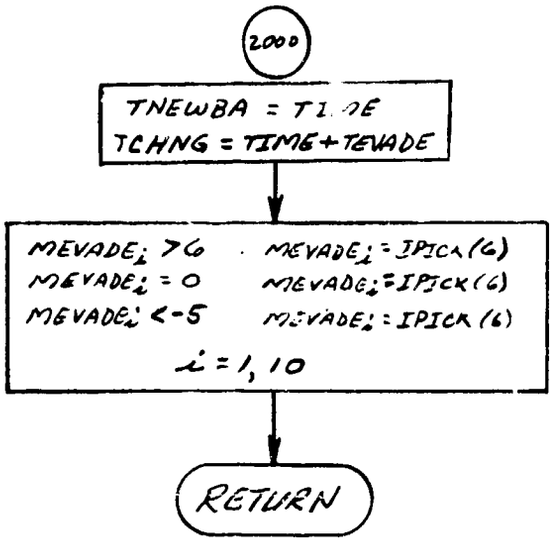
Remarks:

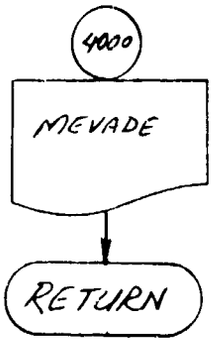
A flow chart for EVADE1 is presented. EVADE2 is identical except for use of vehicle 2 COMMON blocks and auxiliary routines.

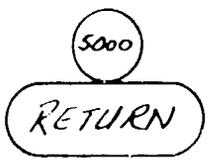
EVADE1

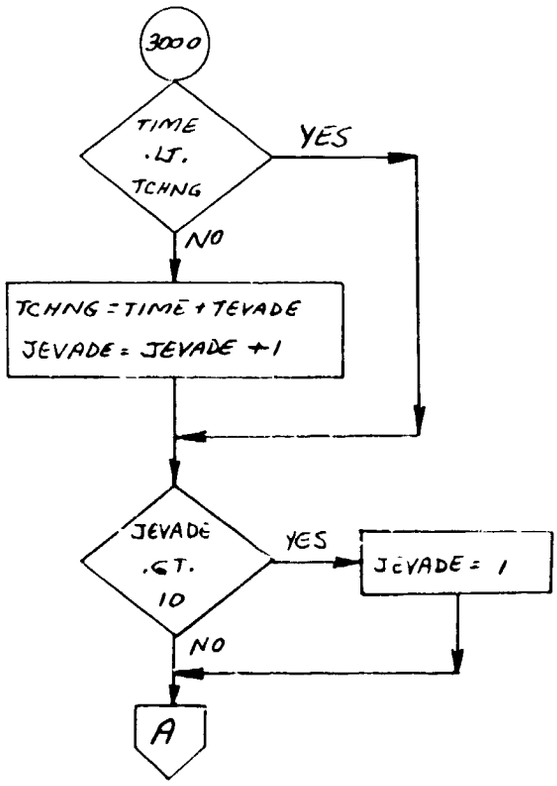

IENTRY
1000
2000
3000
4000
5000
6000
7000
8000

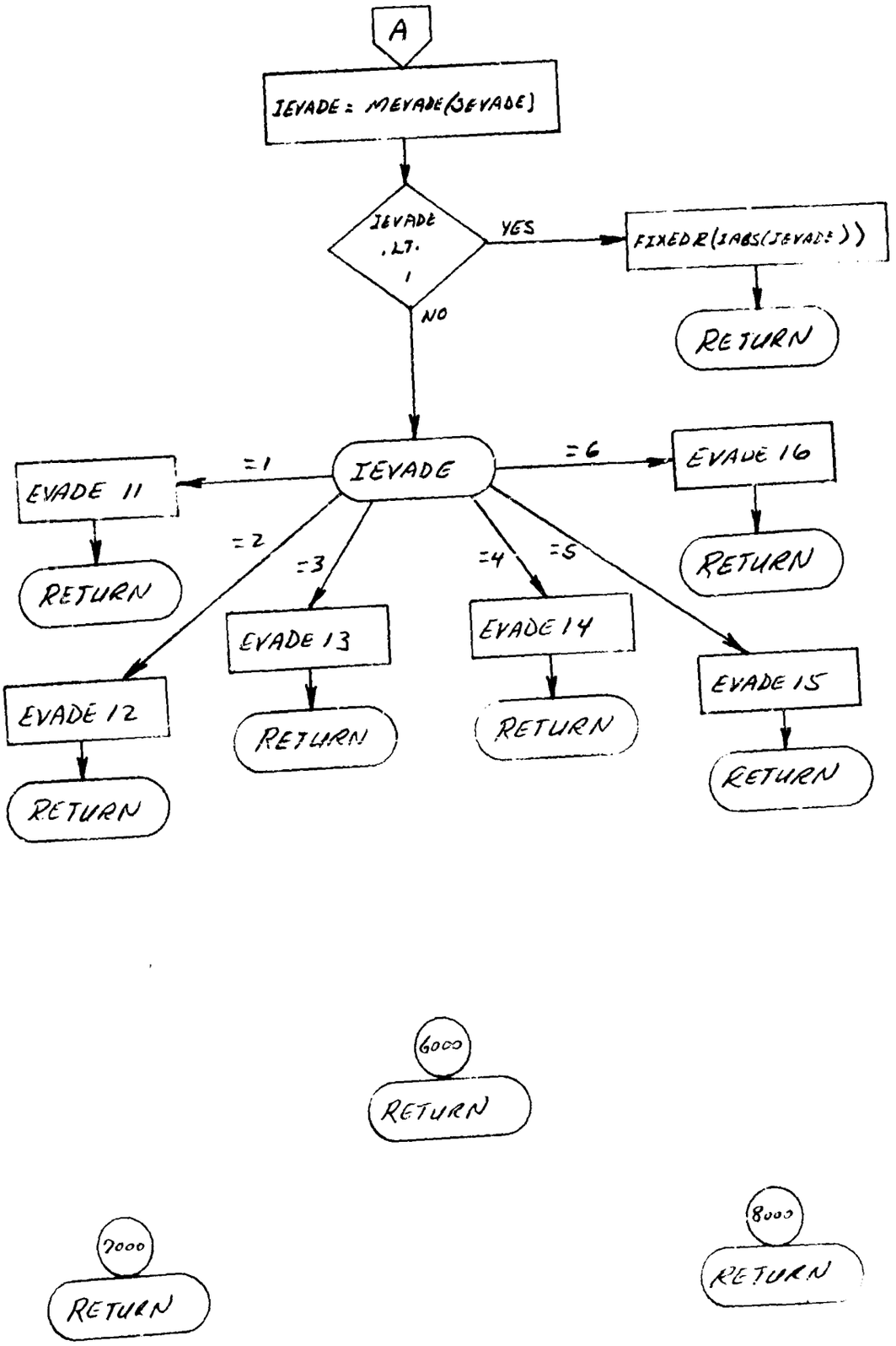

1000
MEVADE_i = IPICK(6)
i = 1,10
JEVADE = 11
TEVADE = 1000.
RETURN


2000
TNEWBA = TIME
TCHNG = TIME + TEVADE
MEVADE_i > 6 MEVADE_i = IPICK(6)
MEVADE_i = 0 MEVADE_i = IPICK(6)
MEVADE_i < -5 MEVADE_i = IPICK(6)
i = 1,10
RETURN


4000
MEVADE
RETURN


5000
RETURN


3000
TIME .LT. TCHNG
YES
NO
TCHNG = TIME + TEVADE
JEVADE = JEVADE + 1
JEVADE .GT. 10
YES
JEVADE = 1
NO
A


A
IEVADE = MEVADE(JEVADE)
IEVADE .LT. 1
YES
FIXED R(IABS(JEVADE))
RETURN
NO
IEVADE
=1
=2
=3
=4
=5
=6
EVADE 11
EVADE 12
EVADE 13
EVADE 14
EVADE 15
EVADE 16
RETURN
RETURN
RETURN
RETURN
RETURN
RETURN
6000
RETURN
7000
RETURN
8000
RETURN

68. OFFEN1 and OFFEN2 - Offensive Tactic Routine

Purpose:

To select each vehicle's offensive tactic.

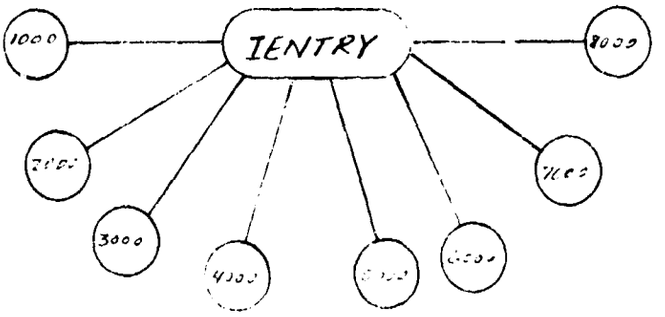
Method:

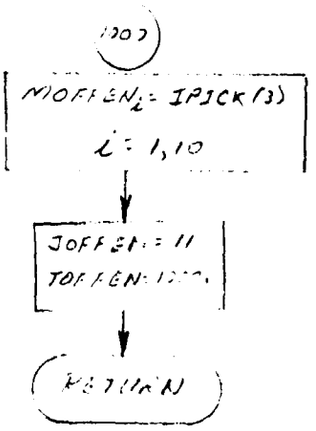
ROLE1 and ROLE2 have selected each vehicle's role. When an offensive role is requested, OFFEN1 and OFFEN2 define a specific offensive tactic for vehicles 1 and 2, respectively. Offensive tactics may be selected in random or ordered manner. An override to another specified role is also possible by use of FIXEDR or FIXEDR2. The analyst may also specify a minimum elapsed time between tactic changes.

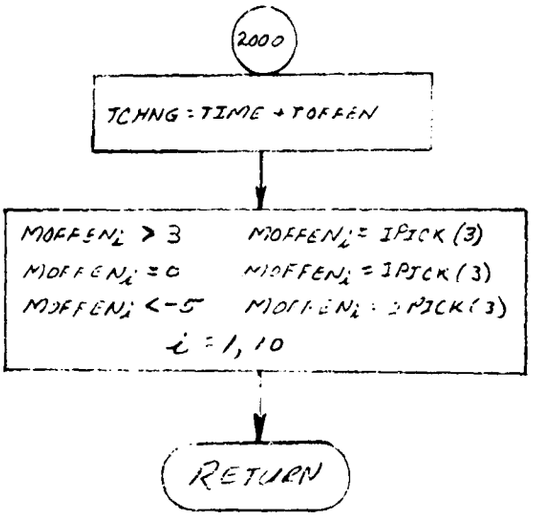
Remarks:

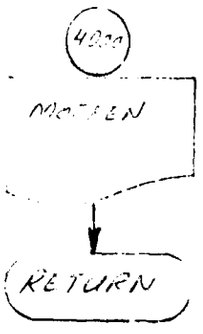
A flow chart for OFFEN1 is presented. OFFEN2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

OFFEN1

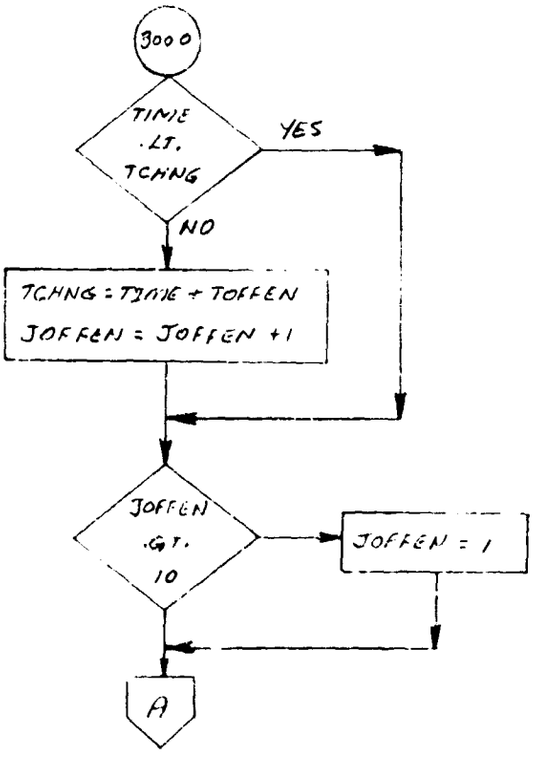

1000
IENTRY
3000

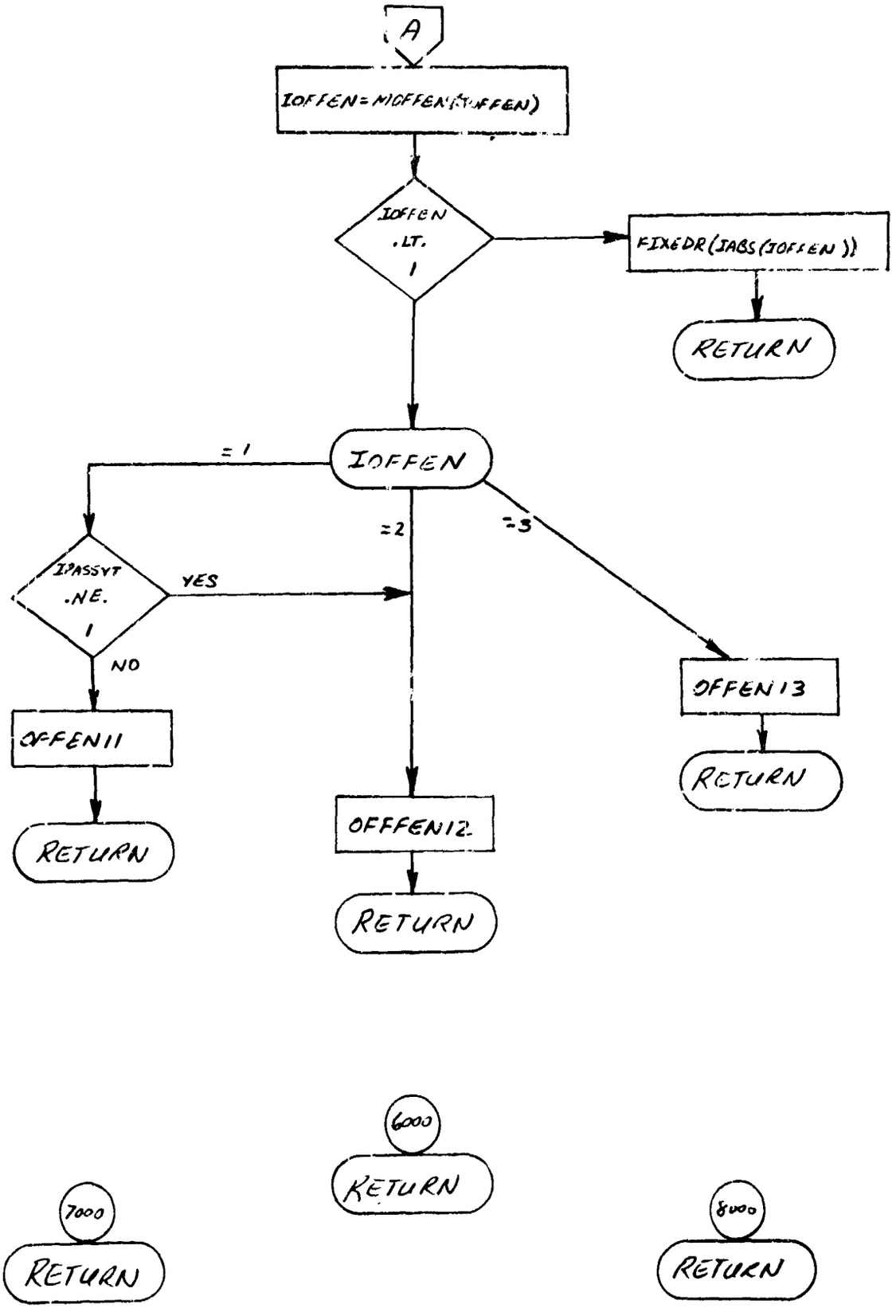

MOFFEN$_i$ = IPICK(3)
i = 1,10
JOFFEN = 11
RETURN


2000
TCHNG = TIME + TOFFEN
MOFFEN$_i$ > 3
MOFFEN$_i$ = IPICK(3)
MOFFEN$_i$ = 0
MOFFEN$_i$ = IPICK(3)
MOFFEN$_i$ < -5
MOFFEN$_i$ = IPICK(3)
i = 1,10
RETURN


RETURN

5000
RETURN


3000
TIME .LT. TCHNG
YES
NO
TCHNG = TIME + TOFFEN
JOFFEN = JOFFEN + 1
JOFFEN .GT. 10
JOFFEN = 1
A


A
IOFFEN=MIOFFEN(IOFFEN)
IOFFEN .LT. 1
FIXEDR(IABS(IOFFEN))
RETURN
IOFFEN
=1
=2
=3
IPASSVT .NE. 1
YES
NO
OFFEN11
RETURN
OFFFEN12
RETURN
OFFEN13
RETURN
6000
RETURN
7000
RETURN
8000
RETURN

69. ATTAC1 and ATTAC2 - Attacking Tactics Routine

Purpose:

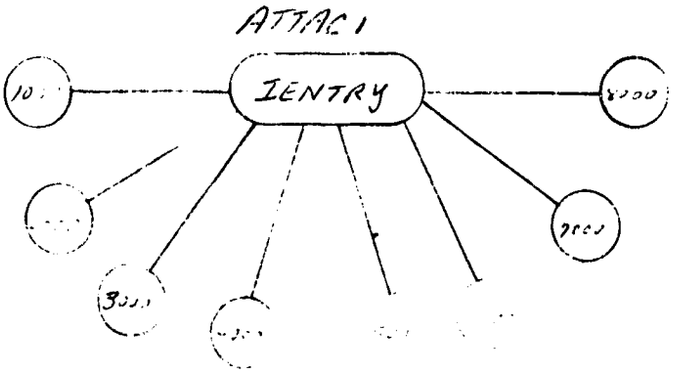
To select each vehicle's attacking tactic.

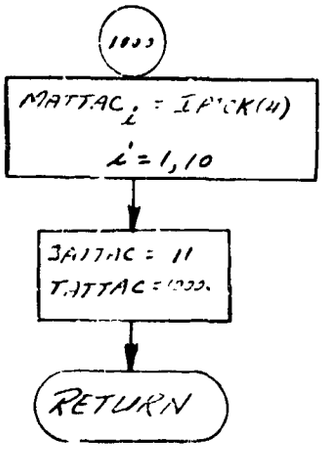
Method:

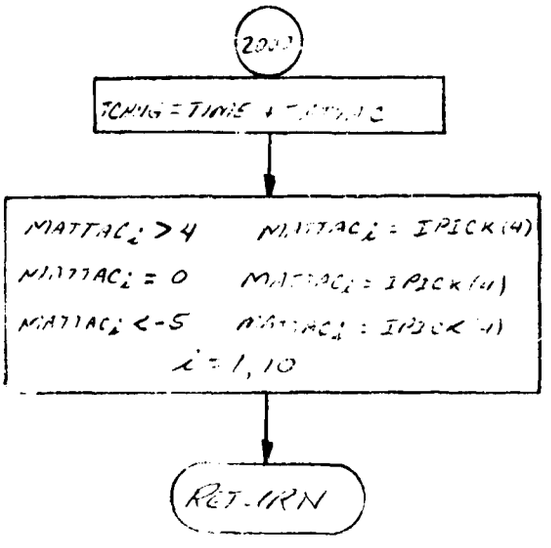
ROLE1 and ROLE2 has selected each vehicle's role. When an attacking role is requested, ATTAC1 and ATTAC2 define a specific attacking tactic for vehicles 1 and 2. Tactics may be selected in random or ordered manner. An override to another specified role is also possible by use of FIXEDR and FIXEDR2. The analyst may also specify a minimum elapsed time between tactic changes.

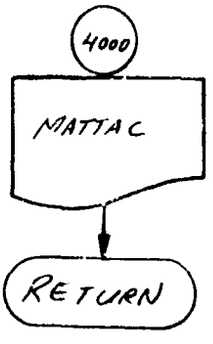
Remarks:

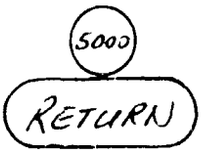
A flow chart for ATTAC1 is presented. ATTAC2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

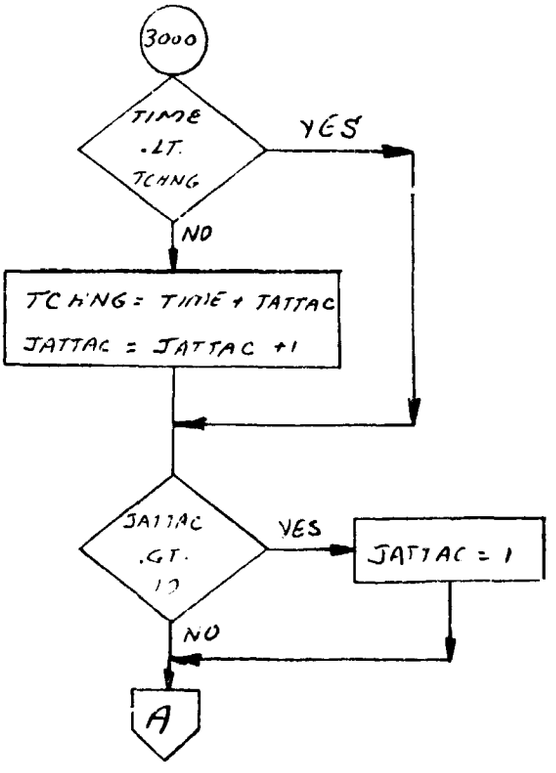

ATTAC 1
IENTRY

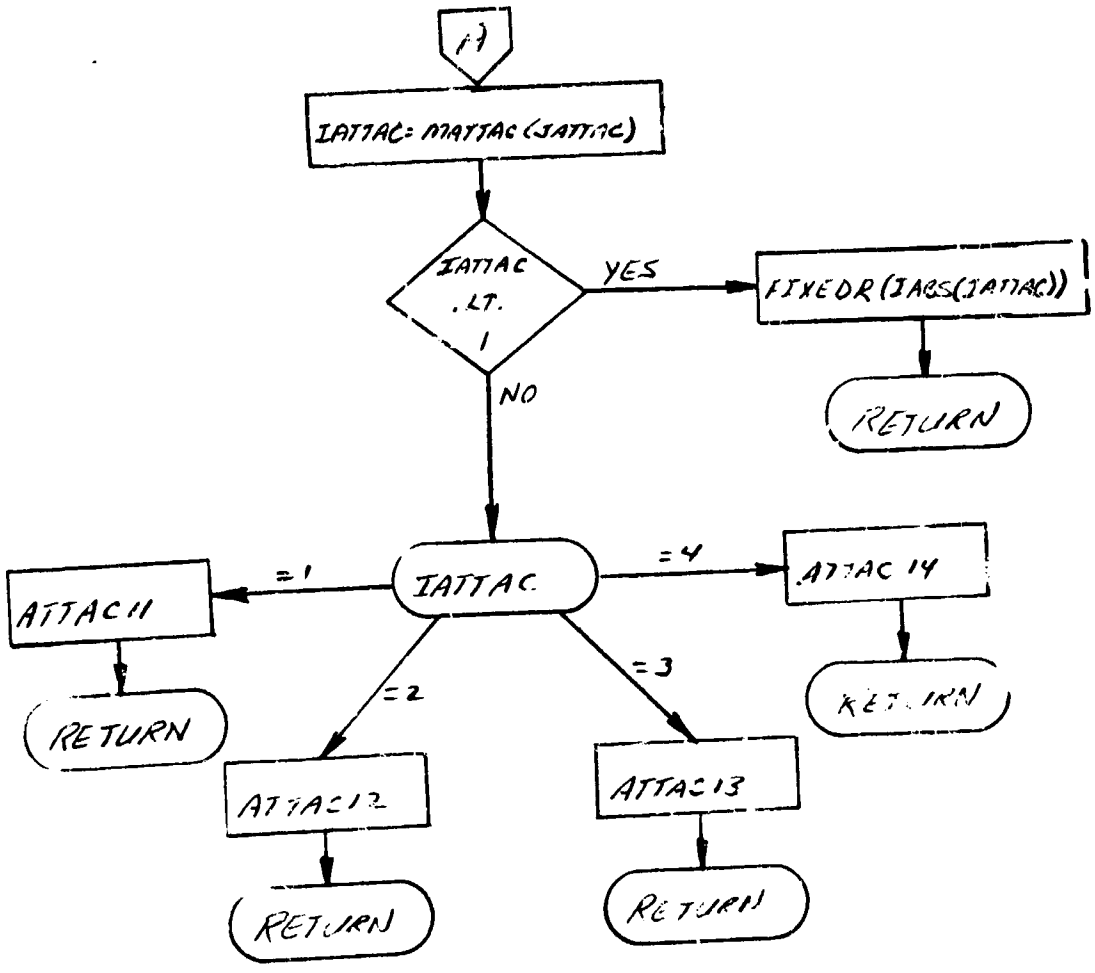

1000
MATTAC$_i$ = IPICK(4)
i = 1,10
JATTAC = 11
TATTAC = 1000.
RETURN

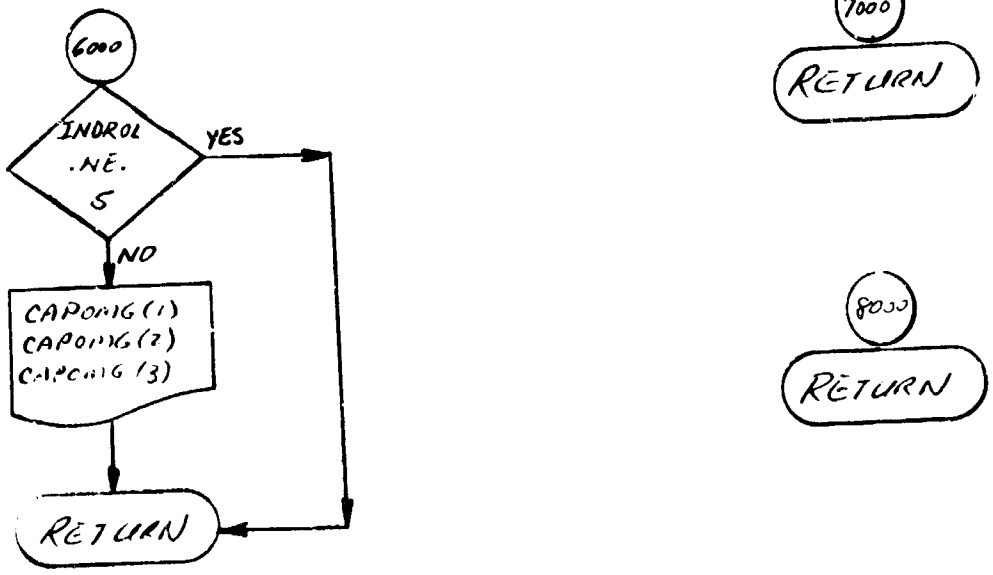

2000
TCHNG = TIME +
MATTAC$_i$ > 4 MATTAC$_i$ = IPICK(4)
MATTAC$_i$ = 0 MATTAC$_i$ = IPICK(4)
MATTAC$_i$ < -5 MATTAC$_i$ = IPICK(4)
i = 1,10
RETURN


4000
MATTAC
RETURN


5000
RETURN


3000
TIME .LT. TCHNG
YES
NO
TCHNG = TIME + JATTAC
JATTAC = JATTAC +1
JATTAC .GT. 10
YES
JATTAC = 1
NO
A


1
IATTAC= MATTAC (JATTAC)
IATTAC .LT. 1
YES
FIXEOR (IABS(JATTAC))
RETURN
NO
IATTAC
=1
=2
=3
=4
ATTAC11
RETURN
ATTAC12
RETURN
ATTAC13
RETURN
ATTAC 14
RETURN


6000
INDROL .NE. 5
YES
NO
CAPONIG (1)
CAPONIG (2)
CAPONIG (3)
RETURN
7000
RETURN
8000
RETURN

70. OALPBA and OALPBA2 - Subprogram for Instantaneous Control Vector Iteration

Purpose:

To define instantaneous angle-of-attack and bank-angle on the basis of local optimization and constraint criteria.

Method:

A local minimization criteria, $\phi(t)$, and local constraints criteria, $\psi_i(t)$, are created where

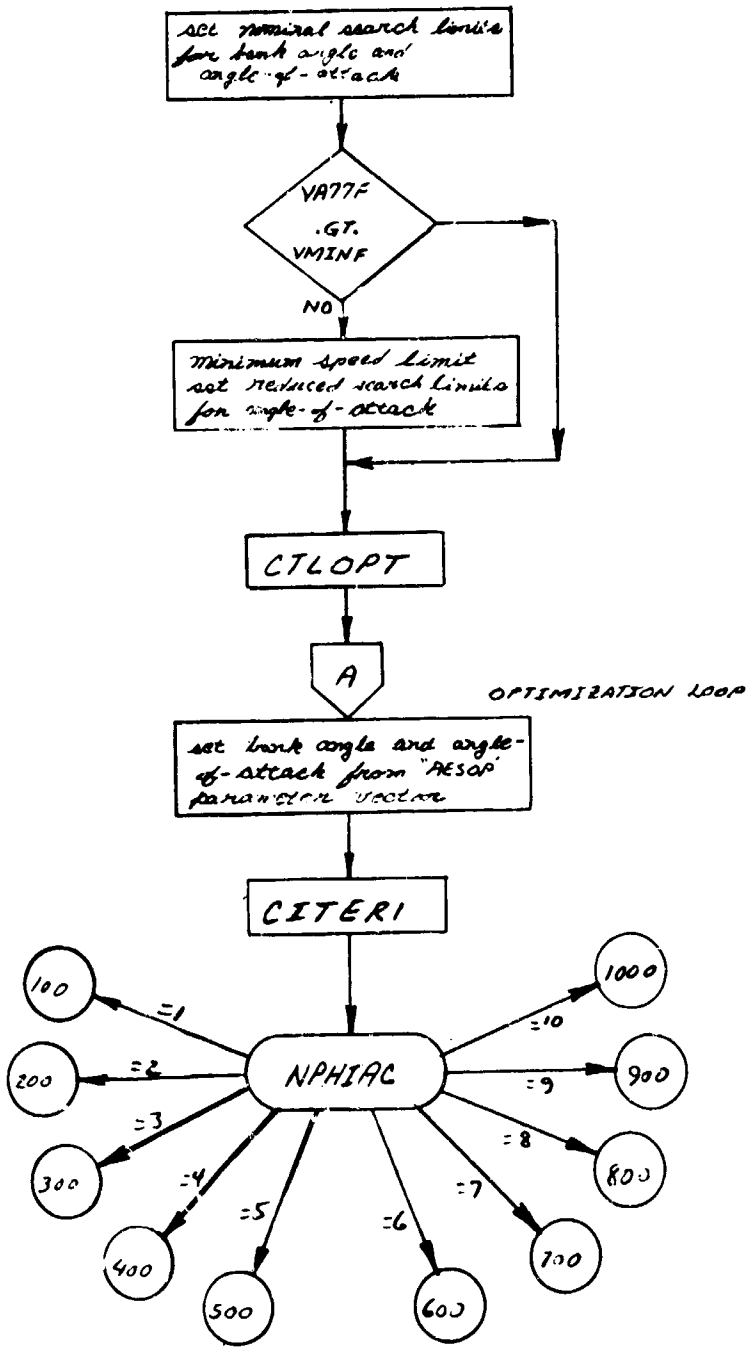
$$\phi(t) = \phi(\alpha, B_A) \text{ and } \psi_i(t) = \psi_i(\alpha, B_A)$$

An inner loop parameter optimization procedure, CTLOPT, is used to define the angle-of-attack and bank-angle combination which satisfies the resulting local (t = constant) optimization problem. The local minimization criteria involves a variety of combat guidance laws. Constraints include a minimum speed override on angle-of-attack, which reduces the angle-of-attack search range and any in-flight inequality constraint defined by IFCS which is directly affected by the control vector ($INDBNC_i$=1). Final steering and force vector errors are also computed in OALPBA and OALPBA2.

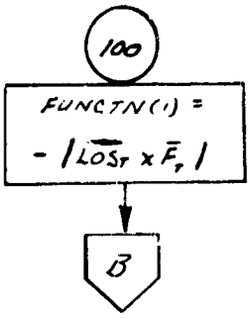
Remarks:

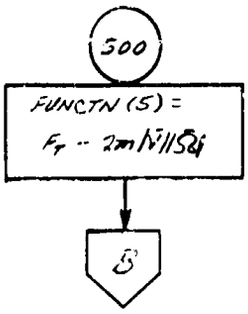
A flow chart for OALPBA is presented. OALPBA2 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

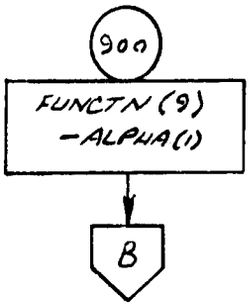
OALPBA

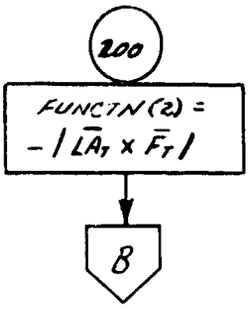

set nominal search limits for bank angle and angle-of-attack
VA77F .GT. VMINF
NO
Minimum speed limit set reduced search limits for angle-of-attack
CTLOPT
A
OPTIMIZATION LOOP
set bank angle and angle-of-attack from "AESOP" parameter vector
CITERI
NPHIAC
100
200
300
400
500
600
700
800
900
1000
=1
=2
=3
=4
=5
=6
=7
=8
=9
=10

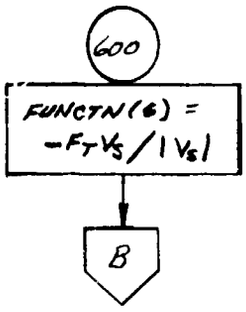
Compute performance function

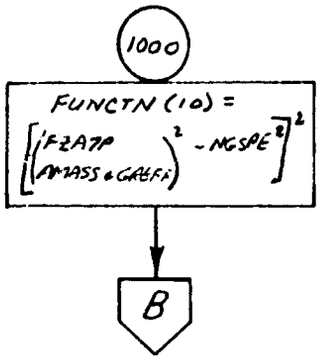


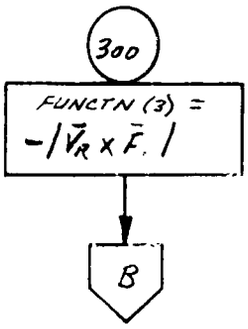


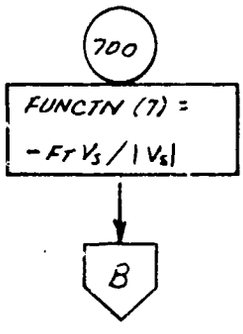


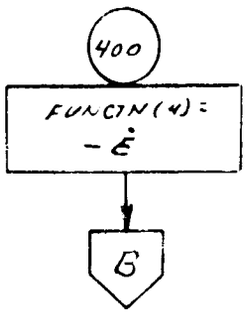


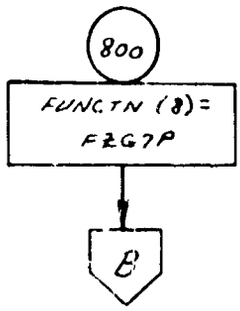


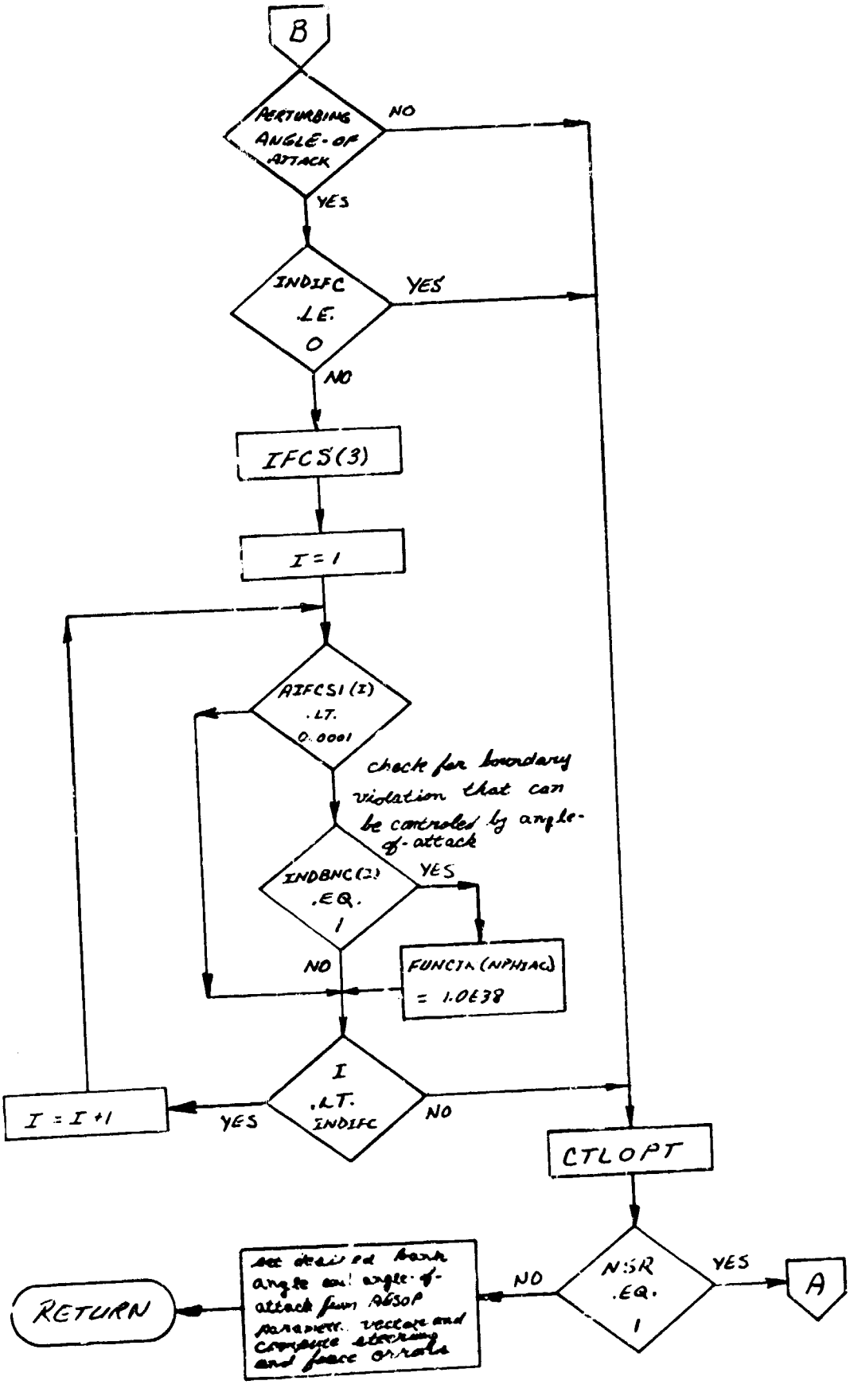











B
PERTURBING ANGLE-OF ATTACK
NO
YES
INDIFC .LE. 0
YES
NO
IFCS(3)
I = 1
AIFCS1(I) .LT. 0.0001
check for boundary violation that can be controled by angle-of-attack
INDBNC(2) .EQ. 1
YES
NO
FUNCTN(NPHIAC) = 1.0E38
I .LT. INDIFC
NO
YES
I = I + 1
CTLOPT
N:SR .EQ. 1
YES
A
NO
set desired bank angle and angle-of-attack from ABSOP parameter vectors and compute aileron and pitch errors
RETURN

71. HIHO and HIHO2 - N-Dimensional Table Call Routine

Purpose

To set up the NA array and Z location of tables with dimension from 3 to 5 as required by the calling sequence to NDTLU which is

CALL NDTLU (ND,NA, X, Z, XA, ZA, IE, NEXTR),

to make the call to NDTLU, and return the function value or data on a table read error.

Usage

Linkage to the subroutine is made via the statement

CALL HIHO (N, LOCZ, NX1, NX2, NX3, NX4, X1ARG, X2ARG, X3ARG, X4ARG, A)

where,

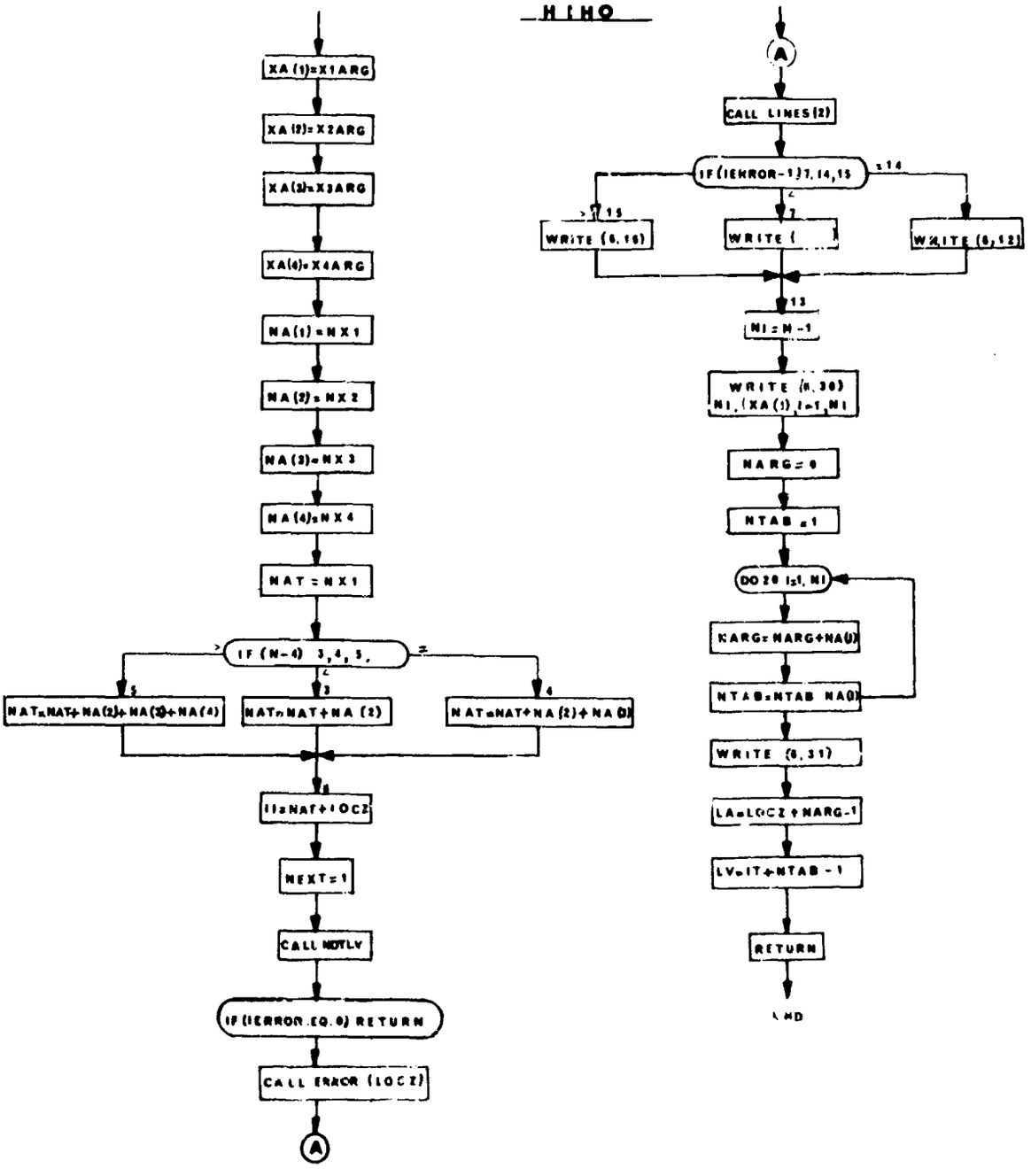
N	= Dimension of table look-up. When A = f(X) N = 2.
LOCZ	= Location of the first value in the table.
NX1 to X1ARG	= Location of number of points in the X1 to X4 arrays of independent variable values.
X1ARG to X4ARG	= Name of X1 to X4 argument or a dummy location if N < 5.
A	= Location of the dependent variable.

Subroutines Called

NDTLU, ERROR, LINES
Normal FORTRAN I/O routines

Remarks:

A flow chart for HIHO is presented. HIHO2 is identical except for the use of vehicle 2 COMMON blocks.


XA(1)=X1ARG
XA(2)=X2ARG
XA(3)=X3ARG
XA(4)=X4ARG
NA(1)=NX1
NA(2)=NX2
NA(3)=NX3
NA(4)=NX4
NAT=NX1
IF (N-4) 3,4,5,
NAT=NAT+NA(2)+NA(3)+NA(4)
NAT=NAT+NA(2)
NAT=NAT+NA(2)+NA(3)
LL=NAT+LOCZ
NEXT=1
CALL NOTLV
IF (IERROR.EQ.0) RETURN
CALL ERACR (LOCZ)
A
HIHO
A
CALL LINES(2)
IF(IERROR-1)17,14,15
WRITE (6,16)
WRITE (
WRITE (6,12)
13
NI=N-1
WRITE (6,30) NI,(XA(I),I=1,NI)
NARG=0
NTAB=1
DO 20 I=1,NI
NARG=NARG+NA(I)
NTAB=NTAB+NA(I)
WRITE (6,31)
LA=LOCZ+NARG-1
LV=IT+NTAB-1
RETURN
END

72. **TMTX - Transformation Matrix Routine**

Purpose

To calculate the transformation matrix, M given the angles of transformation Θ_1, Θ_2, Θ_3 and the desired axes about which rotation is to occur.

Method

$$M = \prod_{i=1}^{3} A_{4-i}$$

where A_i is a matrix determined from angle Θ_i. If $\Theta_i = 0$, $A_i = I$ (identity matrix)

A_i is defined as follows:

$$A_i = \begin{bmatrix} 1 & 0 & 0 \\ 0 & C_{\Theta_i} & S_{\Theta_i} \\ 0 & -S_{\Theta_i} & C_{\Theta_i} \end{bmatrix} = \text{X axis rotation matrix when } K_i = 1$$

$$A_i = \begin{bmatrix} C_{\Theta_i} & 0 & -S_{\Theta_i} \\ 0 & 1 & 0 \\ S_{\Theta_i} & 0 & C_{\Theta_i} \end{bmatrix} = \text{Y axis rotation matrix when } K_i = 2$$

$$A_i = \begin{bmatrix} C_{\Theta_i} & S_{\Theta_i} & 0 \\ -S_{\Theta_i} & C_{\Theta_i} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \text{Z axis rotation matrix when } K_i = 3$$

Usage

The transformation matrix routine is entered by the statement:

CALL TMTX (Θ_1, K_1, Θ_2, K_2, Θ_3, K_3, M, V) where:

1. Θ_1, Θ_2, Θ_3 define the transformation angles
2. K_1, K_2, K_3, as indicated above, determine the axis to which the corresponding Θ_i refers.

3. V is a 1-dimensional array containing past values of the following:

   Θ_1, $\text{Sin}\Theta_1$, $\text{COS}\Theta_1$, Θ_2, $\text{Sin}\Theta_2$, $\text{COS}\Theta_2$, Θ_3, $\text{Sin}\Theta_3$, $\text{COS}\Theta_3$, (in that order)

4. M is the 3 x 3 array where the resultant transformation matrix is to be stored.

V contains the last computed values of each Θ_i, $\text{Sin}\Theta_i$. If the current Θ_i presented by the call statement is such that

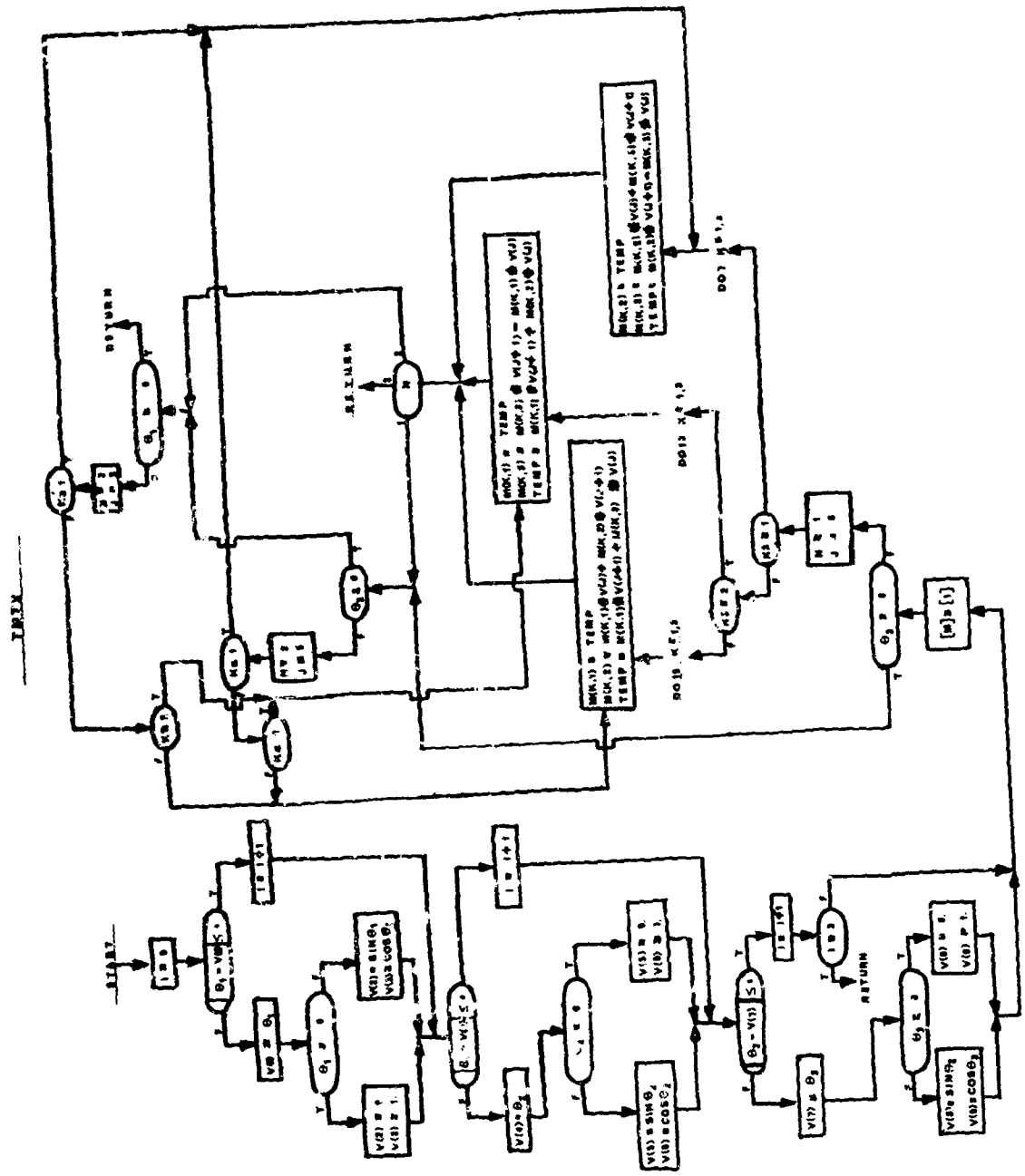
$$\left|\Theta_{\text{new } i} - \Theta_{\text{old } i}\right| \leq E, \quad E = 1.E\text{-}6$$

then $\text{Sin}\Theta_i$, $\text{COS}\Theta_i$ are not recomputed. If <u>all</u> Θ_i's are such that $\left|\Theta_{\text{new } i} - \Theta_{\text{old } i}\right| \leq E$, then M is not recomputed at all. <u>Any</u> recomputed values are stored in the V array.

To avoid possible errors in the E-test, V(1), V(4), and V(7) should be initialized to angles which would never be encountered in the user's program <u>before</u> the first entry into TMTX.

<u>Note</u>: For any given transformation, it is assumed that the order of rotation about the 3 axes will not change.

When $\Theta_i = 0$, K is ignored.



73. TRNPOS - A 3 x 3 Matrix Transpose Routine

Purpose

To transpose a 3 x 3 matrix A to obtain the 3 x 3 matrix A'.

Method

The resulting transposed matrix is stored in a separate array. All elements of A must be stored in the normal FORTRAN sense (i.e., columnwise).

Usage

The transpose of a matrix A is obtained by the statement,

```
CALL TRNPOS (A,B)
```

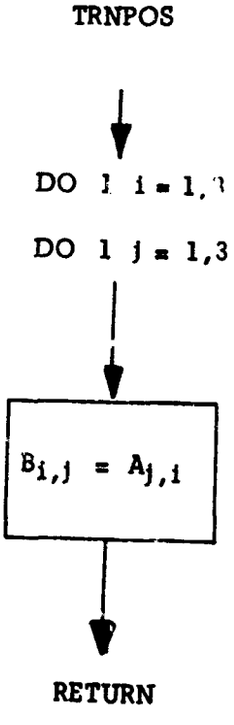
where,

A = The variable name of the 3 x 3 matrix A.

B = The variable name of the 3 x 3 matrix A'.

Remarks

This routine calls no other routines.



74. MULT31 - A Matrix Multiplication Routine

Purpose

To post-multiply a 3 x 3 matrix by a 3 x 1 matrix.

Method

The result of $[A][B] = [C]$ is computed using single precision floating point arithmetic. All elements must be stored in the normal FORTRAN sense (i.e., columnwise).

Usage

The matrix multiplication is obtained by the statement:

CALL MULT31 (A, B, C)

where,

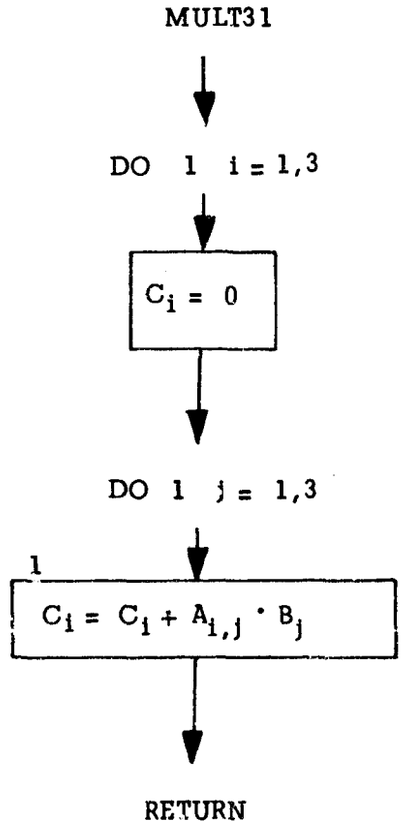
A = Array name of the 3 x 3 matrix $[A]$

B = Array name of the 3 x 1 matrix $[B]$

C = Array name of the resulting 3 x 1 matrix $[C]$

Remarks

This routine calls no other routine.



75. HETS and HETS2 - Heating Computations

Purpose:

To monitor a characteristic structural temperature and/or aerodynamic heating rate.

Method:

This subprogram has all the standard entry points of EXE.

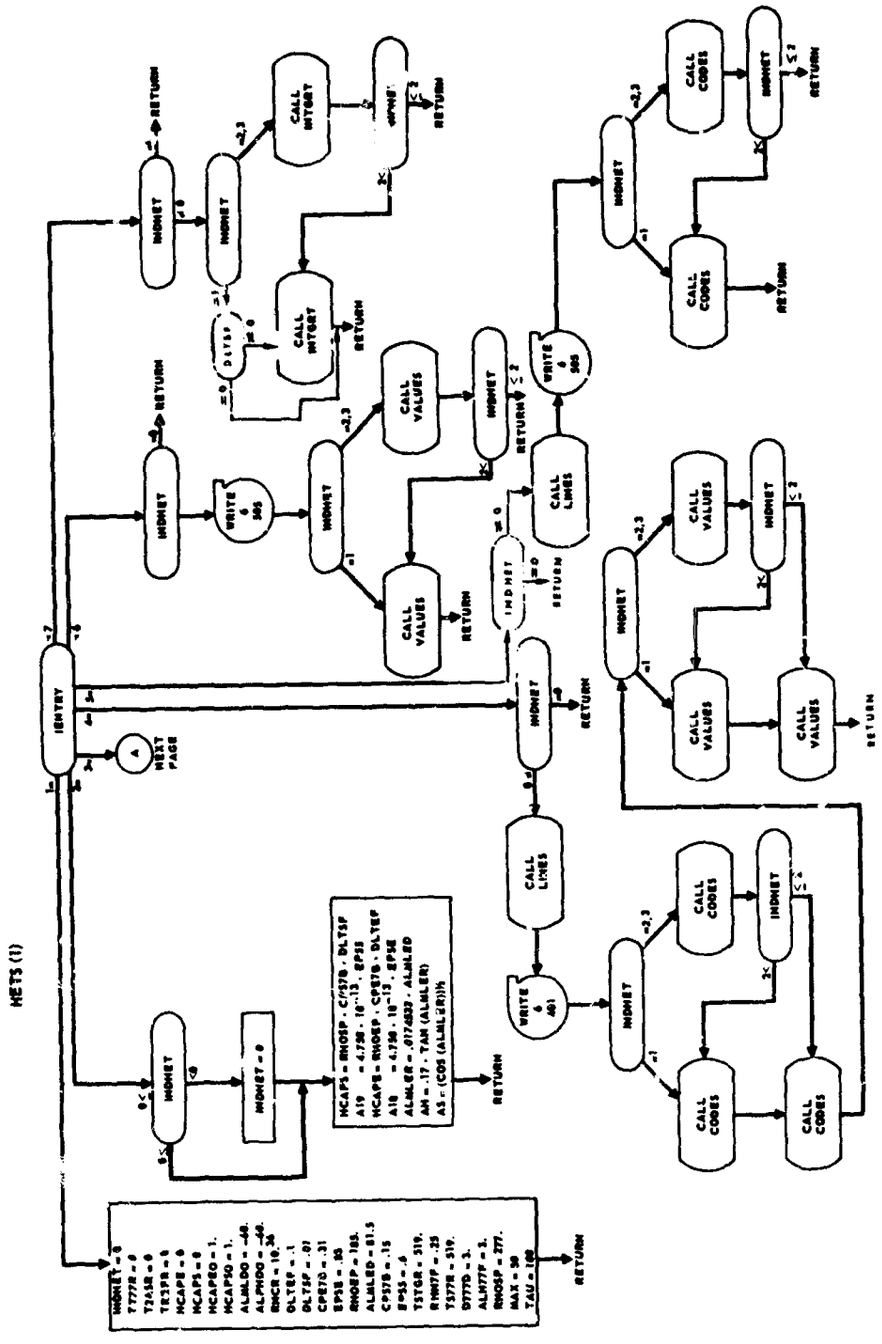
INDHET = 0:	No computations made.
INDHET = 1:	Wedge skin temperature computed.
INDHET = 2:	Hemispherical nose temperature computed.
INDHET = 3:	Both wedge skin and hemispherical nose temperatures computed.
DLTSF = 0:	Equilibrium temperature computed for wedge skin. Transient temperature computed for hemispherical nose.
DLTSF $\neq$ 0:	Transient temperatures computed for both wedge skin and hemispherical nose.

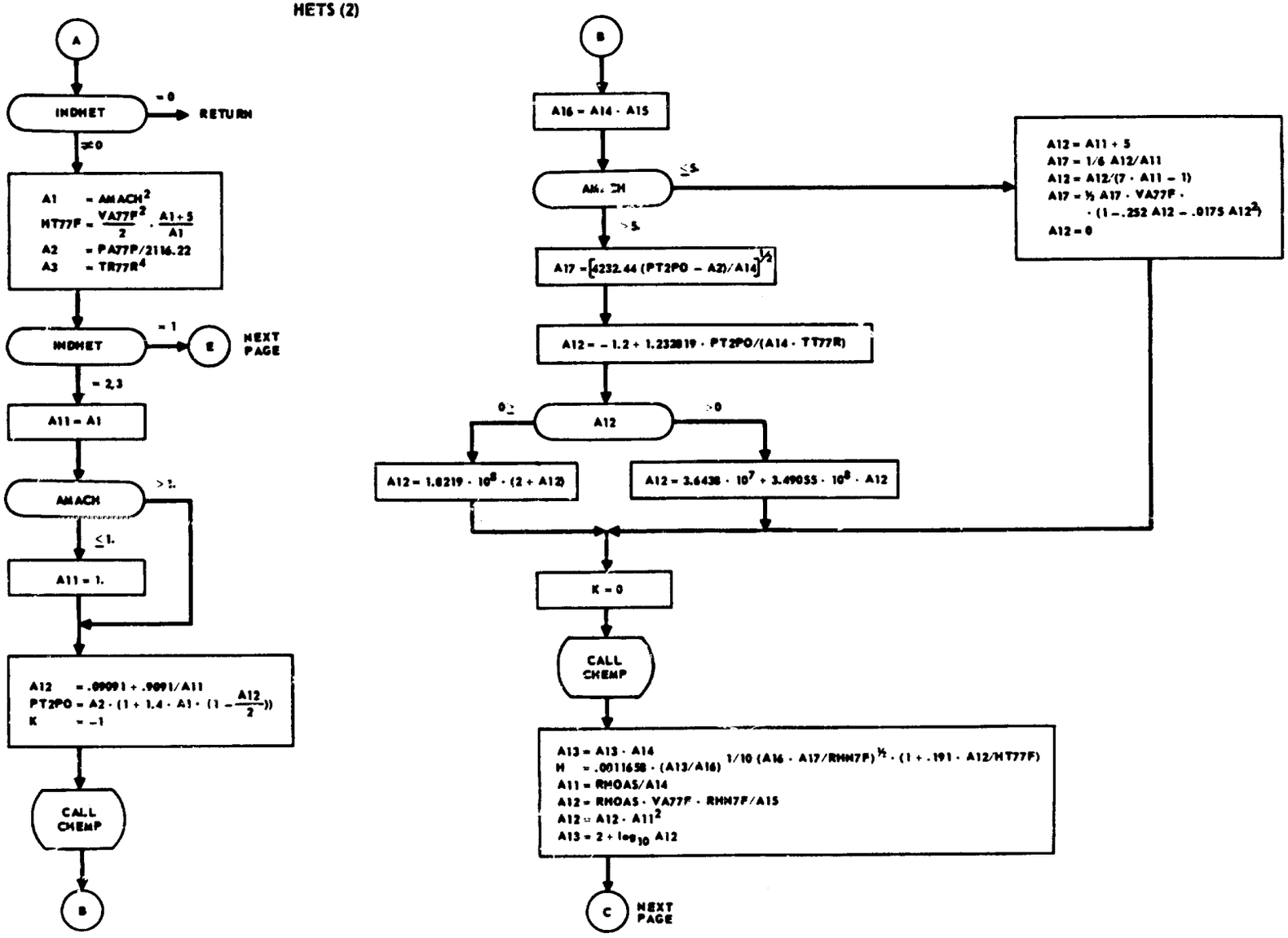
Remarks:

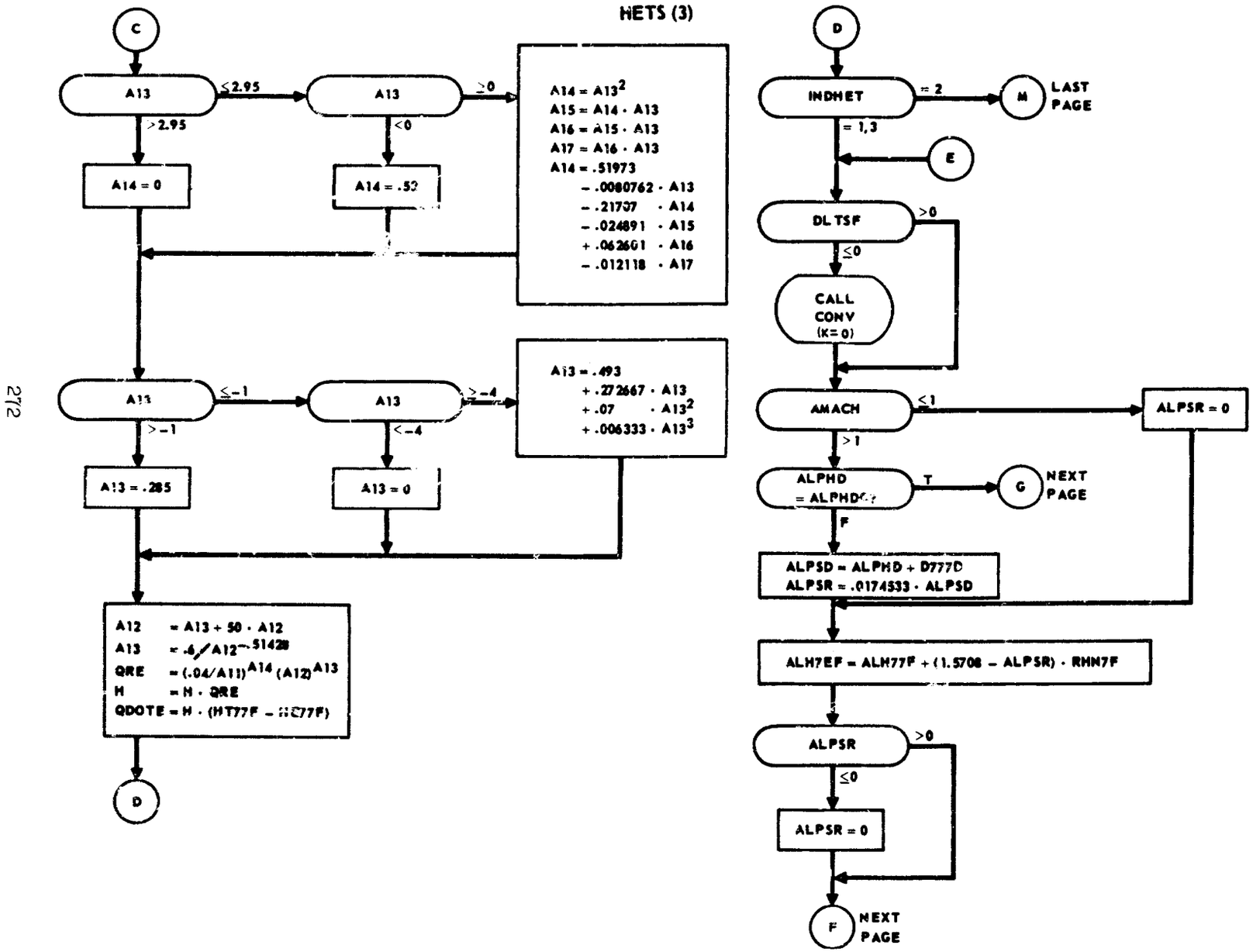
The following quantities are computed at HETS(3):

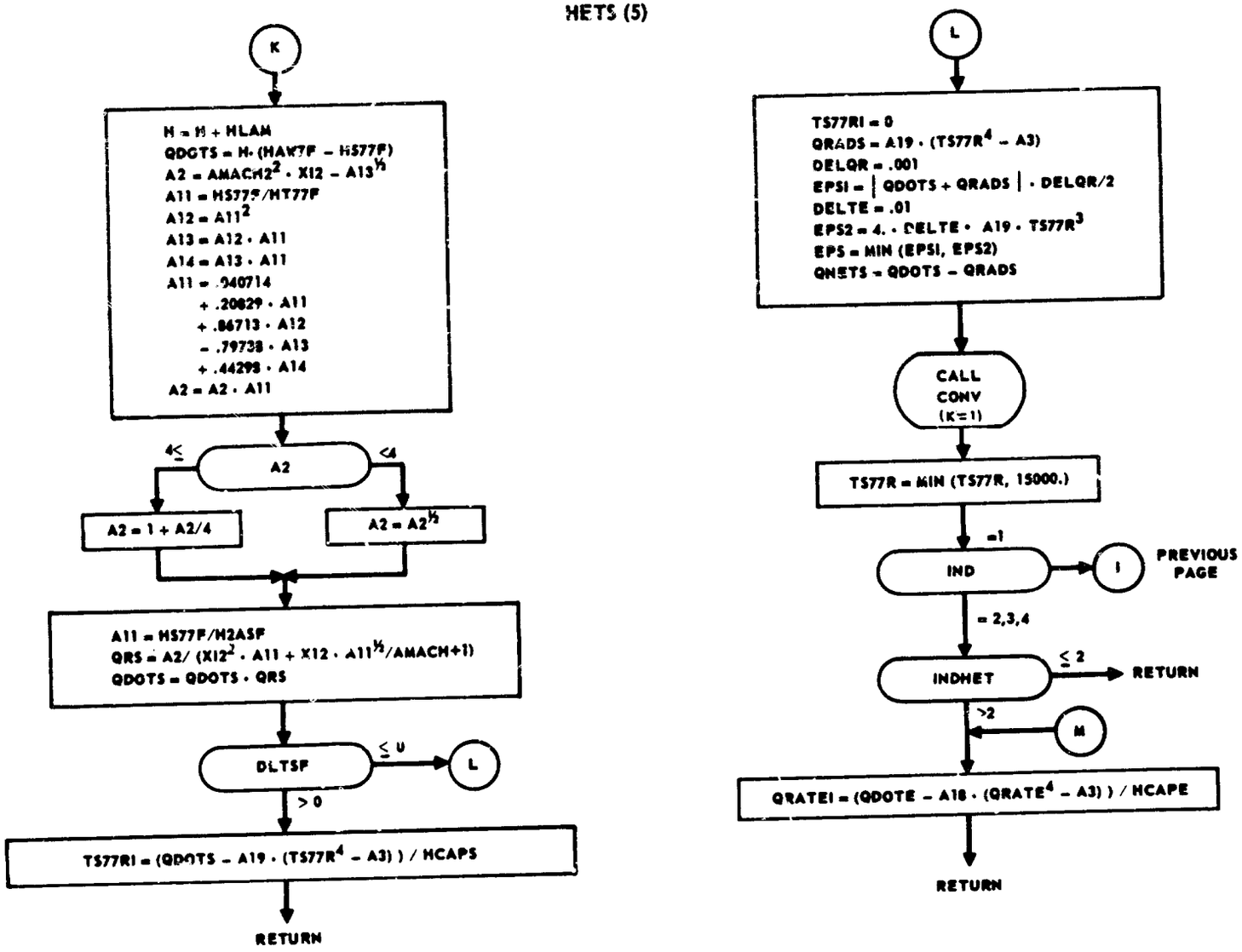
QDOTS	Convective heating rate of the wedge skin.
TS77R1	Rate of change of wedge skin temperature (set equal to zero if DLTSF = 0).
TS77R	Wedge skin temperature.
QDOTE	Convective heating rate of the hemispherical nose.
TSTGR1	Rate of change of hemispherical nose temperature.
TSTGR	Hemispherical nose temperature.
RN2	Reynolds number at edge of boundary layer.
RNCR	Critical Reynold's number at edge of boundary layer.

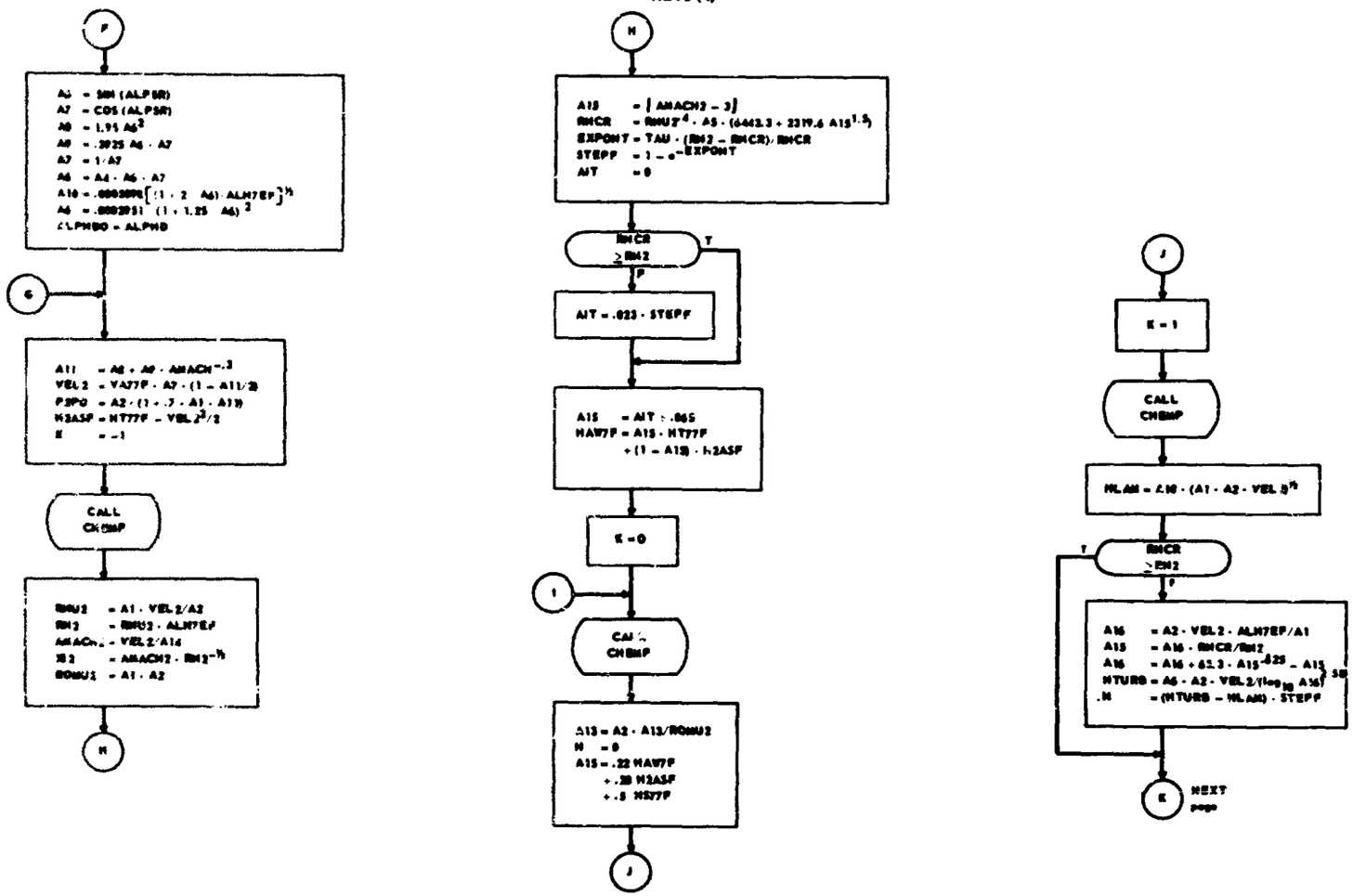
A flow chart for HETS is presented. HETS2 is identical to HETS except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

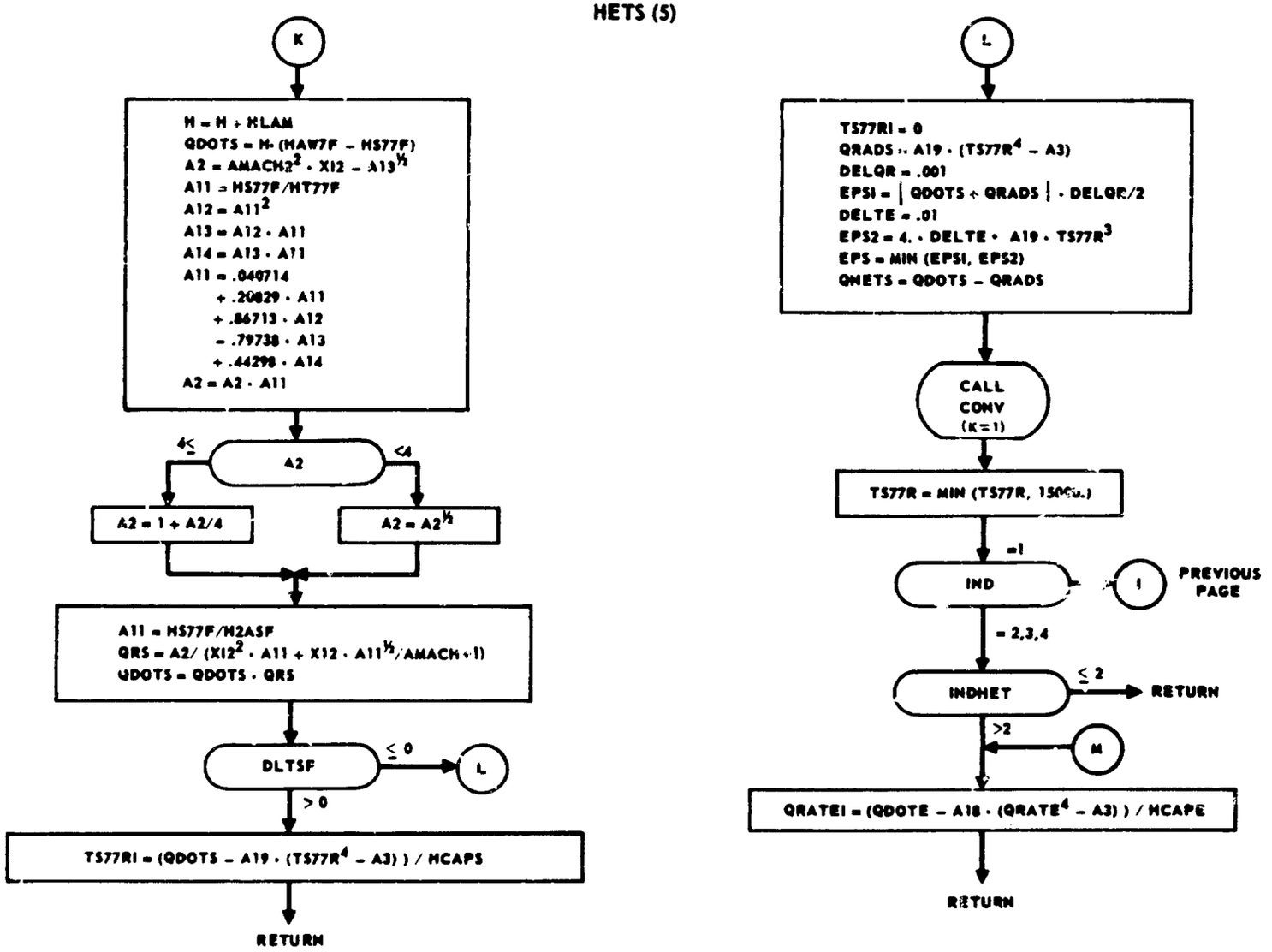

HETS (1)
RETURN
CALL VALUES
CALL LINES
CALL CODES
CALL INTGRT
WRITE 6 305
WRITE 6 401
NEXT PAGE


A
INDHET
= 0
RETURN
≠0
A1 = AMACH²
HT77F = VA77F²/2 · (A1+5)/A1
A2 = PA77P/2116.22
A3 = TR77R⁴
INDHET
= 1
E
NEXT PAGE
= 2,3
A11 = A1
AMACH
> 1.
≤ 1.
A11 = 1.
A12 = .09091 + .9091/A11
PT2PO = A2 · (1 + 1.4 · A1 · (1 − A12/2))
K = −1
CALL CHEMP
B
B
A16 = A14 · A15
AM. CH
≤ 5.
> 5.
A12 = A11 + 5
A17 = 1/6 A12/A11
A12 = A12/(7 · A11 − 1)
A17 = ½ A17 · VA77F · · (1 − .252 A12 − .0175 A12²)
A12 = 0
A17 = [4232.44 (PT2PO − A2)/A14]^½
A12 = − 1.2 + 1.232819 · PT2PO/(A14 · TT77R)
A12
< 0
> 0
A12 = 1.8219 · 10⁸ · (2 + A12)
A12 = 3.6438 · 10⁷ + 3.49055 · 10⁸ · A12
K = 0
CALL CHEMP
A13 = A13 · A14
H = .0011658 · (A13/A16)^1/10 (A16 · A17/RHN7F)^½ · (1 + .191 · A12/HT77F)
A11 = RHOAS/A14
A12 = RHOAS · VA77F · RHN7F/A15
A12 = A12 · A11²
A13 = 2 + log10 A12
C
NEXT PAGE


HETS (3)
C
A13
≤2.95
>2.95
A13
≥0
<0
A14 = 0
A14 = .52
A14 = A13^2
A15 = A14 · A13
A16 = A15 · A13
A17 = A16 · A13
A14 = .51973
− .0080762 · A13
− .21707 · A14
− .024891 · A15
+ .062601 · A16
− .012118 · A17
A13
≤−1
>−1
A13
≥−4
<−4
A13 = .285
A13 = 0
A13 = .493
+ .272667 · A13
+ .07 · A13^2
+ .006333 · A13^3
A12 = A13 + 50 · A12
A13 = .6/A12^.51428
QRE = (.04/A11)^A14 (A12)^A13
H = H · QRE
QDOTE = H · (HT77F − HE77F)
D
D
INDHET
= 2
M
LAST PAGE
= 1,3
E
DLTSF
>0
≤0
CALL CONV (K=0)
AMACH
≤1
ALPSR = 0
>1
ALPHD = ALPHD
T
G
NEXT PAGE
F
ALPSD = ALPHD + D777D
ALPSR = .0174533 · ALPSD
ALH7EF = ALH77F + (1.5708 − ALPSR) · RHN7F
ALPSR
>0
≤0
ALPSR = 0
F
NEXT PAGE


K
H = H + HLAM
QDOTS = H· (HAW?F − HS77F)
A2 = AMACH2² · X12 − A13½
A11 = HS77F/HT77F
A12 = A11²
A13 = A12 · A11
A14 = A13 · A11
A11 = .940714
+ .20829 · A11
+ .86713 · A12
− .79738 · A13
+ .44298 · A14
A2 = A2 · A11
≥4
A2
<4
A2 = 1 + A2/4
A2 = A2½
A11 = HS77F/H2ASF
QRS = A2/ (X12² · A11 + X12 · A11½/AMACH+1)
QDOTS = QDOTS · QRS
DLTSF
≤ 0
L
> 0
TS77RI = (QDOTS − A19 · (TS77R⁴ − A3)) / HCAPS
RETURN
L
TS77RI = 0
QRADS = A19 · (TS77R⁴ − A3)
DELQR = .001
EPSI = | QDOTS + QRADS | · DELQR/2
DELTE = .01
EPS2 = 4. · DELTE · A19 · TS77R³
EPS = MIN (EPSI, EPS2)
QNETS = QDOTS − QRADS
CALL
CONV
(K=1)
TS77R = MIN (TS77R, 15000.)
=1
IND
1
PREVIOUS
PAGE
= 2,3,4
INDHET
≤ 2
RETURN
>2
M
QRATEI = (QDOTE − A18 · (QRATE⁴ − A3)) / HCAPE
RETURN


F
A5 = SIN (ALPSR)
A7 = COS (ALPSR)
A8 = 1.95 A6^2
A9 = .3925 A6 · A7
A7 = 1/A7
A4 = A4 · A5 · A7
A10 = .0003896[(1 + 2 A6) ALH7EP]^½
A6 = .0002951 (1 + 1.25 A6)^2
ALPHBO = ALPHB
G
A11 = A8 + A9 · AMACH^-2
VEL2 = VA77F · A7 · (1 − A11/2)
P3PO = A2 · (1 + .7 · A3 · A11)
H2ASF = HT77F − VEL2^2/2
E = −1
CALL CHSMP
RNU2 = A1 · VEL2/A2
RN2 = RNU2 · ALH7EP
AMACH2 = VEL2/A10
R2 = AMACH2 · RN2^-½
RQNU2 = A1 · A2
H
H
A15 = ⌊AMACH2 − 3⌋
RNCR = RNU2^.4 · A5 · (6442.3 + 2319.6 A15^1.5)
EXPONT = TAU · (RN2 − RNCR)/RNCR
STEPF = 1 − e^−EXPONT
AIT = 0
RNCR ≥ RN2
T
F
AIT = .023 · STEPF
A15 = AIT + .865
HAW7F = A15 · HT77F + (1 − A15) · H2ASF
E = 0
I
CALL CHSMP
A13 = A2 · A13/RQNU2
N = 0
A15 = .22 HAW7F + .28 H2ASF + .5 HS77F
J
J
E = 1
CALL CHSMP
HLAM = A10 · (A1 · A2 · VEL2)^½
RNCR ≥ RN2
T
F
A16 = A2 · VEL2 · ALH7EP/A1
A13 = A16 · RNCR/RN2
A16 = A16 + 65.3 · A15^.425 − A15
HTURB = A6 · A2 · VEL2/(log10 A16)^2.58
.N = (HTURB − HLAM) · STEPF
E
NEXT page


K
H = H + HLAM
QDOTS = H· (HAW7F − HS77F)
A2 = AMACH2² · X12 − A13½
A11 = HS77F/HT77F
A12 = A11²
A13 = A12 · A11
A14 = A13 · A11
A11 = .040714
+ .20829 · A11
+ .86713 · A12
− .79738 · A13
+ .44298 · A14
A2 = A2 · A11
≥4
A2
<4
A2 = 1 + A2/4
A2 = A2½
A11 = HS77F/H2ASF
QRS = A2/ (X12² · A11 + X12 · A11½/AMACH+1)
QDOTS = QDOTS · QRS
DLTSF
≤ 0
L
> 0
TS77RI = (QDOTS − A19 · (TS77R⁴ − A3)) / HCAPS
RETURN
L
TS77RI = 0
QRADS = A19 · (TS77R⁴ − A3)
DELQR = .001
EPSI = | QDOTS + QRADS | · DELQR/2
DELTE = .01
EPS2 = 4. · DELTE · A19 · TS77R³
EPS = MIN (EPSI, EPS2)
QNETS = QDOTS − QRADS
CALL
CONV
(K=1)
TS77R = MIN (TS77R, 150
=1
IND
1
PREVIOUS
PAGE
= 2,3,4
INDHET
≤ 2
RETURN
>2
M
QRATEI = (QDOTE − A18 · (QRATE⁴ − A3)) / HCAPE
RETURN

76. TFFS and TFFS2 - Single Engine Thrust and Fuel Flow

Purpose:

The single engine thrust and fuel flow program provides the means of introducing the engine thrust and propellant flow rate data. It corrects the thrust for atmospheric effects and resolves the thrust vector into its components.

Usage:

Linkage to TFFS is accomplished via the general statement:

CALL TFFS (IENTRY)

where IENTRY is a fixed point variable.

IENTRY = 1

This performs the pre-data initialization. At this entry the subscripts for all tables are computed and the following data is initialized.

INDTFF = 0 = thrust option indicator

N = 1. = throttle setting

BURNRS = 2. = number of identical engines

In addition, the data initialization of TFFM when its IENTRY is 1 is performed under this entry.

IENTRY = 2

This performs the post-data initialization. At this entry the following data is initialized:

$\dot{m}$ = 0 = rate of change of vehicle mass

m_o = m = mass of vehicle

m_f = 0 = fuel used

T_x = 0 = thrust component along x-axis

T_y = 0 = thrust component along y-axis

T_z = 0 = thrust component along z-axis

T = 0 = thrust after atmospheric effects correction

T_{VAC} = 0 = vacuum thrust of engine

$\dot{m}_f$ = 0 = fuel flow rate

IENTRY = 3

At this entry the thrust components and fuel flow are computed. If INDTFF = 0 no computations are performed. For values of INDTFF of 1 through 5 the following computations are performed:

$$T = MAX \left\{ \begin{matrix} 0 \\ T_{VAC} - P\ A_e \end{matrix} \right\} = \text{total corrected thrust}$$

$$\dot{m}_t = -\bar{\dot{m}}_t = \text{total rate of change of vehicle mass}$$

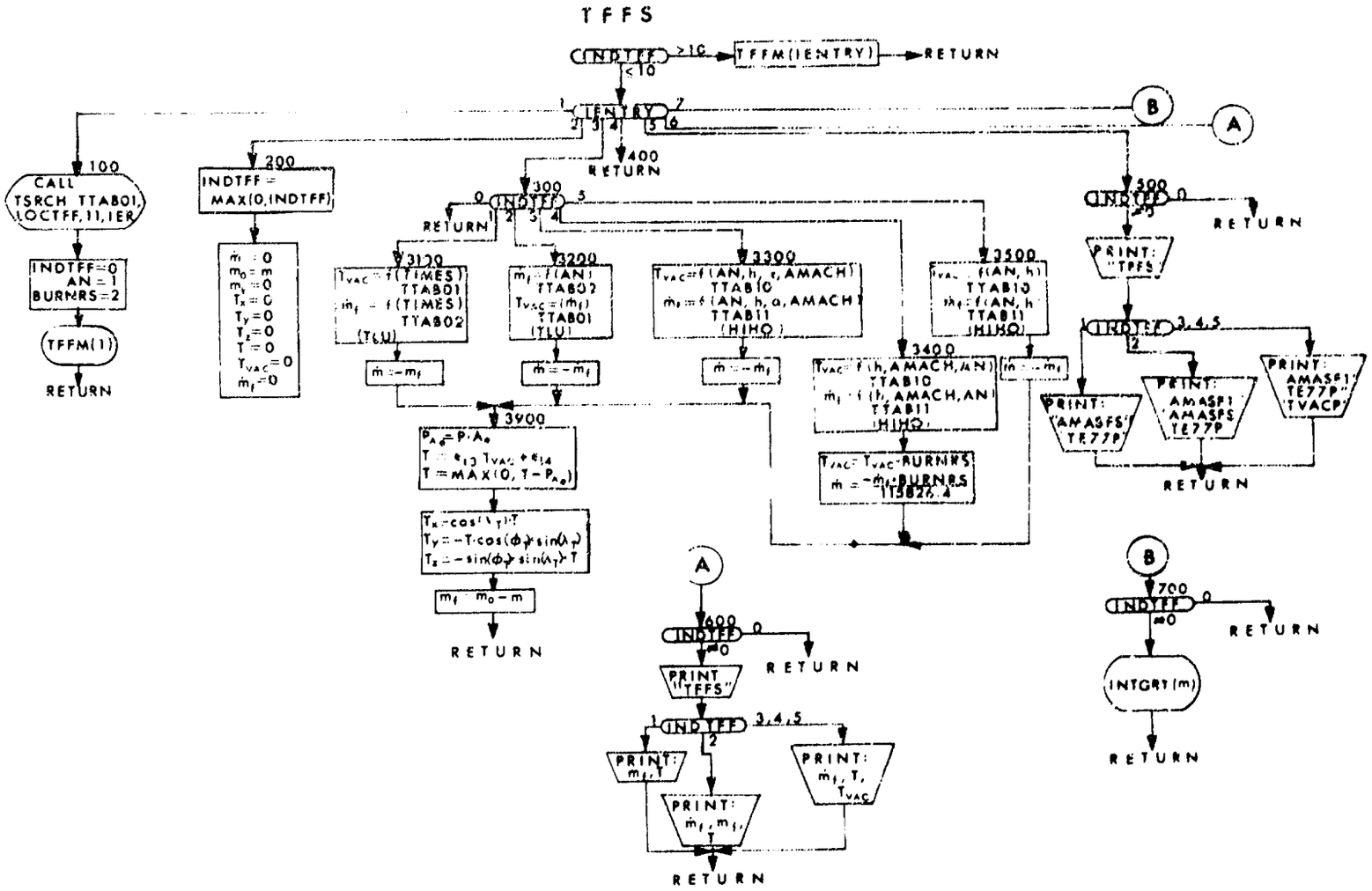
T_{VAC} and $\dot{m}_t$ are input as tabular data. The functional relationships for the various options are as follows:

Value of INDTFF	Option	Functional Relationship
1	Single Engine Noncontrolled Thrust	$\bar{\dot{m}}_t = f(\tau^s)$ $T_{VAC} = f(\tau^s)$
2	Single Engine Controlled Thrust	$\bar{\dot{m}}_t = f(N)$ $T_{VAC} = f(\dot{m}_t)$
3	Single Engine Air Breather	$\bar{\dot{m}}_t = f(N,h,\alpha,M_N)$ $T_{VAC} = f(N,h,\alpha,M_N)$
4	Fractional or Multiple Identical Engine	$\bar{\dot{m}}_t = f(h,M_N,N)$ $T_{VAC} = f(h,M_N,N)$
5	Simplified Single Engine	$\bar{\dot{m}}_t = f(N,h)$ $T_{VAC} = f(N,h)$

NOTES:

While it is possible to change from the single engine option (TFFS) in a given stage to the multiengine option (TFFM) in a later stage, the opposite is not possible.

TFFS



INDTFF
>10
TFFM(IENTRY)
RETURN
<10
IENTRY
100
CALL TSRCH TTAB01, LOCTFF, 11, IER
INDTFF=0
AN=1
BURNRS=2
TFFM(1)
RETURN
200
INDTFF = MAX(0,INDTFF)
300
INDTFF
RETURN
400
RETURN
3100
3200
3300
3400
3500
3900
500
PRINT: "TFFS"
PRINT: "AMASFS" TE77P
PRINT: AMASF1 AMASFS TE77P
PRINT: AMASF1 TE77P TVACP
600
PRINT "TFFS"
PRINT: m_f, T
PRINT: $\dot{m}_f$, m_f, T
PRINT: $\dot{m}_f$, T, T_{VAC}
700
INTGRT(m)

77. SACS and SACS2 - Aerodynamic Routines

Purpose:

To compute the aerodynamic forces, side force (SIDEP), drag force (DRAGP) and lift force (ALIFTP). SACS is used for vehicle 1; SACS2, for vehicle 2.

Usage:

Linkage is made via the general statement:

CALL SACS (IENTRY)

where IENTRY is a fixed point number.

IENTRY = 1

At this entry curve read subscripts are found by calling subroutine TSRCH and the following variables are initialized

	INDAER	=	0
	INDBAD	=	0
L	ALIFTP	=	0
D	DRAGP	=	0
Y	SIDEP	=	0
C_D	CDMNU	=	0
C_Y	CYMNU	=	0
C_L	CLMNU	=	0
$h_{aero_{max}}$	AMAXH	=	10^6
	AMAXA	=	300000.
	AINCD	=	1.
	CPAK1	=	0
	CPAK2	=	0
	INDAO1	=	0
	INDAO2	=	0
	INDAO3	=	0
	INDAO4	=	0

INDA07 = 0

INDA10 = 0

INDA11 = 0

INDA12 = 0

INDA13 = 0

INDA14 = 0

INDA17 = 0

INDA24 = 0

INDA25 = 0

INDA26 = 0

INDA27 = 0

INDA28 = 0

INDA31 = 0

INDA80 = 0

INDA90 = 0

INDA91 = 0

INDA92 = 0

IENTRY = 2

Checks to see if INDAER is less than 0 and if so sets INDAER to 0.

IENTRY = 3

If INDAER is 0 no computation. If INDAER is nonzero a check is made on altitude (HGC7F). If altitude is greater than AMAXH then the following variables are set

ALIFTP = 0

DRAGP = 0

SIDEP = 0

If altitude is less than AMAXH a test is made on the data input value (AMAXA). If altitude (HGC7F) is greater than AMAXA then the following variables are set before ALIFTP, DRAGP and SIDEP are computed.

CA = CDMNU CYA = CYMNU CL = CLMNU

If the altitude is less than the input value AMAXA the following variables are set.

CA = 0 CYA = 0 CN = 0

at this time a test is made on the INDAER for the option chosen.

INDAER = 1

If INDA90 is 0 then CN is unchanged;

If INDA90 = 1

CN = f (ALPHD,AMACH).

If INDA90 = 2

CN = f (ALPHD,AMACH,HGC7F)

If INDA91 is 0 then CA is unchanged, if not

CA = f (|CN|,AMACH).

CYA is set to 0.

If INDA01 is 0 then DELCA is set to 0, if INDA01 is nonzero then

DELCA = f (AMACH).

CA = f (CA + DELCA).

If INDA02 is 0 then CPAK1 is unchanged, if INDA02 is nonzero then

CPAK1 = f (AMACH).

If INDA03 is 0 CPAK2 is unchanged, if INDA03 is nonzero then

CPAK2 = f (AMACH).

Cockpit and pitch angle are computed as auxiliary computations as follows:

CPA7D = Cockpit α = CPAK1 . α + CPAK2

PA77D = Pitch angle = γ + α cos B_A - AINCD

After this computation ALIFTP, DRAGP and SIDEP are computed.

INDAER = 2

RN = VA77F/ANUA7F where

RN = Reynolds Number

VA77F = Velocity, V_A

ANUA7F = Kinematic Viscosity, ν

If INDA80 is 0, CA_0 = CAVAH = 0, if INDA80 is nonzero then

CA_0 = CAVAH = f(RN, AMACH)

If any of the following indicators are nonzero, then the respective coefficients are computed as a function of Mach number.

INDA01	CAALPH	=	CA_α	=	f(AMACH)
INDA02	CAALSQ	=	$CA_\alpha{}^2$	=	f(AMACH)
INDA03	CABETA	=	CA_β	=	f(AMACH)
INDA04	CABTSQ	=	$CA_\beta{}^2$	=	f(AMACH)
INDA07	CAALBT	=	$CA_{\alpha\beta}$	=	f(AMACH
INDA10	CN1ZER	=	CN_0	=	f(AMACH)
INDA11	CN1ALP	=	CN_α	=	f(AMACH)
INDA12	CN1ASQ	=	$CN_\alpha{}^2$	=	f(AMACH)
INDA13	CN1BET	=	CN_β	=	f(AMACH)
INDA14	CN1BSQ	=	$CN_\beta{}^2$	=	f(AMACH)
INDA17	CN1ALB	=	$CN_{\alpha\beta}$	=	f(AMACH)
INDA24	CYZERO	=	CY_0	=	f(AMACH)
INDA25	CYALPH	=	CY_α	=	f(AMACH)
INDA26	CYALSQ	=	$CY_\alpha{}^2$	=	f(AMACH)
INDA27	CYBETA	=	CY_β	=	f(AMACH)
INDA28	CYBTSQ	=	$CY_\beta{}^2$	=	f(AMACH)
INDA31	CYALBT	=	$CY_{\alpha\beta}$	=	f(AMACH)

From these coefficients CA, CN and C_{YA} are computed as:

$$CA = |\alpha| CA_{\alpha} + CA_{0} + \alpha^{2} CA_{\alpha 2} + |\beta| CA_{\beta} + \beta^{2} CA_{\beta 2} + CA_{\alpha\beta} |\alpha| |\beta|$$

$$CN = \alpha CN_{\alpha} + CN_{0} + \alpha |\alpha| CN_{\alpha 2} + |\beta| CN_{\beta} + \beta^{2} CN_{\beta 2} + |\beta| \alpha CN_{\alpha\beta}$$

$$CYA = |\alpha| CY_{\alpha} + CY_{0} + CY_{\alpha 2} \alpha^{2} + \beta CY_{\beta} + |\beta| \beta CY_{\beta 2} + |\alpha| \beta CY_{\alpha\beta}$$

After this computation ALIFTP, DRAGP and SIDEP are computed.

INDAER = 3

If INDA90, INDA91 and INDA92 are 0, CA, CN and CYA respectively are unchanged but if either indicator is nonzero, the coefficients are computed as follows:

If	INDA90	$\neq$	0	CA	=	f(ALPHD,BETAD,AMACH)
	INDA91	$\neq$	0	CN	=	f(ALPHD,BETAD,AMACH)
	INDA92	$\neq$	0	CYA	=	f(ALPHD,BETAD,AMACH)

The routine then computes lift, drag and side force.

INDAER = 4

If INDA90, INDA91 and INDA92 are 0, then CA, CN and CYA are unchanged. If any of the indicators are nonzero, the coefficients are computed as follows:

If	INDA90	$\neq$	0	CA	=	f(AMACH,ALPHD)
	INDA91	$\neq$	0	CN	=	f(AMACH,ALPHD)
	INDA92	$\neq$	0	CYA	=	f(ALPHD,BETAD,AMACH)

With these coefficients lift, drag and sideslip are computed.

INDAER = 5

If any of the indicators listed are set to 0 the respective coefficient is unchanged. If any indicators are nonzero the coefficients are computed as follows:

If	INDA10	$\neq$	0	CAMIN	=	f(AMACH)
	INDA11	$\neq$	0	ELPRIM	=	f(AMACH)
	INDA12	$\neq$	0	ZK	=	f(AMACH)
	INDA13	$\neq$	0	CLZ	=	f(AMACH)

INDA80	$\neq$	0	CN	=	f(ALPHD,AMACH)
INDA01	$\neq$	0	DELCA	=	f(AMACH)
INDA02	$\neq$	0	CPAK1	=	f(AMACH)
INDA03	$\neq$	0	CPAK2	=	f(AMACH)

$CA = CAMIN + ELPRIM\ (CN-CLZ)^2 + ZK(CN-CLZ)^6 + DELCA$

$CYA = 0$

Cockpit alpha and pitch angle are computed as auxiliary computation.

CPA7D = Cockpit α = CPAK1α+CPAK2

PA77D = Pitch angle = $\gamma + \alpha \cos B_A$-AINCD

with the above computation lift, drag and sideslip are computed.

INDAER = 6

If INDA90 and INDA91 are 0, CA and CN are unchanged. If INDA90 and/or INDA91 are not zero the coefficients are computed as follows:

$A = (\alpha^2 + \beta^2)^{1/2}$

INDA90	$\neq$	0	CA	=	f(A.AMACH,HGC7F)
INDA91	$\neq$	0	CN	=	f(A,AMACH,HGC7F)

$CN = CN \frac{|\alpha|}{\alpha}$

INDAER = 7

If INDA80 and/or INDA10 is zero then CA and CN are unchanged.

If INDA80 $\neq$ 0 then

CA = f(HGC7F,ALPHD)

If INDA10 $\neq$ 0, then

CN = f(ALPHD)

LIFT, DRAG, SIDE FORCE

If INDBAD is $\neq$ 0, the following computations are made.

CD = CA CY = CYA CL = CN

If INBAD = 0, the following computations are made.

$$\begin{bmatrix} -CD \\ CY \\ -CL \end{bmatrix} = \begin{bmatrix} uvw \end{bmatrix} \begin{bmatrix} -CA \\ CYA \\ -CN \end{bmatrix}$$

With these coefficients lift, drag and side force are computed as follows:

LIFT	=	ALIFTP	= q S(ϵ_1CL+ ϵ_2)
DRAG	=	DRAGP	= q S(ϵ_3CD+ ϵ_4)
SIDE FORCE	=	SIDEP	= q S(ϵ_5CY+ ϵ_6)

q = dynamic pressure

ϵ_1, ϵ_2, ϵ_3, ϵ_4, ϵ_5, and ϵ_6 are input error constants

S = reference area.

IENTRY = 4

Not used.

IENTRY = 5

A test is made on INDAER if 0, no computation. If INDAER is equal to 2 the following codes are printed.

CAVAH CL CD CY L7D

If INDAER = 1, the following codes are printed

CL CD CY L7D PA77D CPA7D

If INDAER = 5, the following codes are printed

CL CD CY L7D PA77D CPA7D

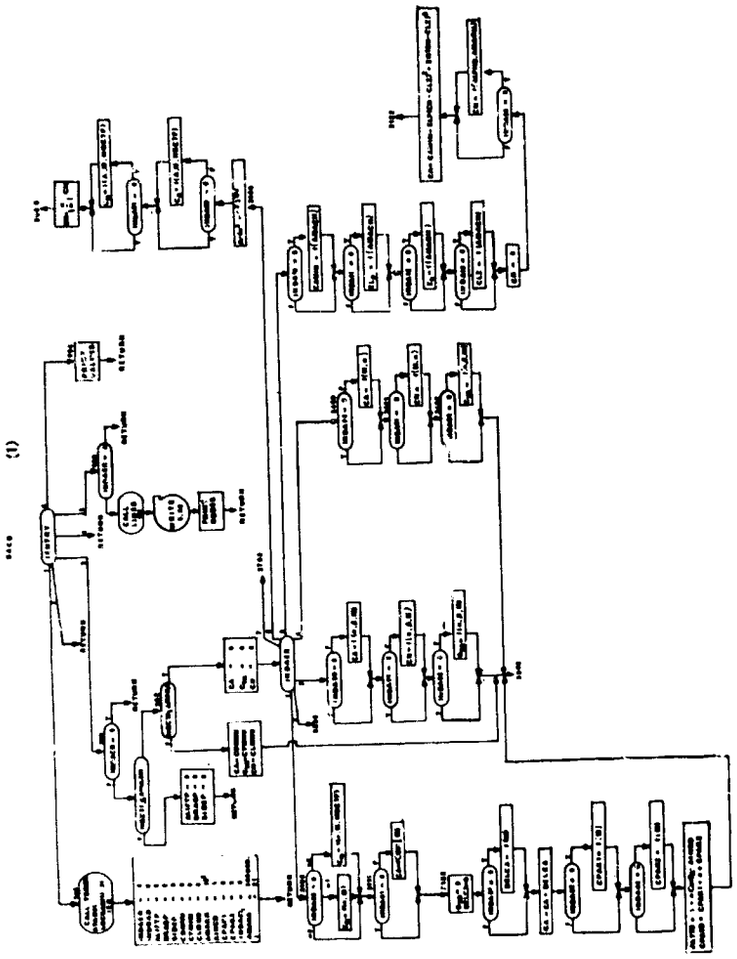
For any other value of INDAER ≠ 0, the following codes are printed

CL CD CY L7D

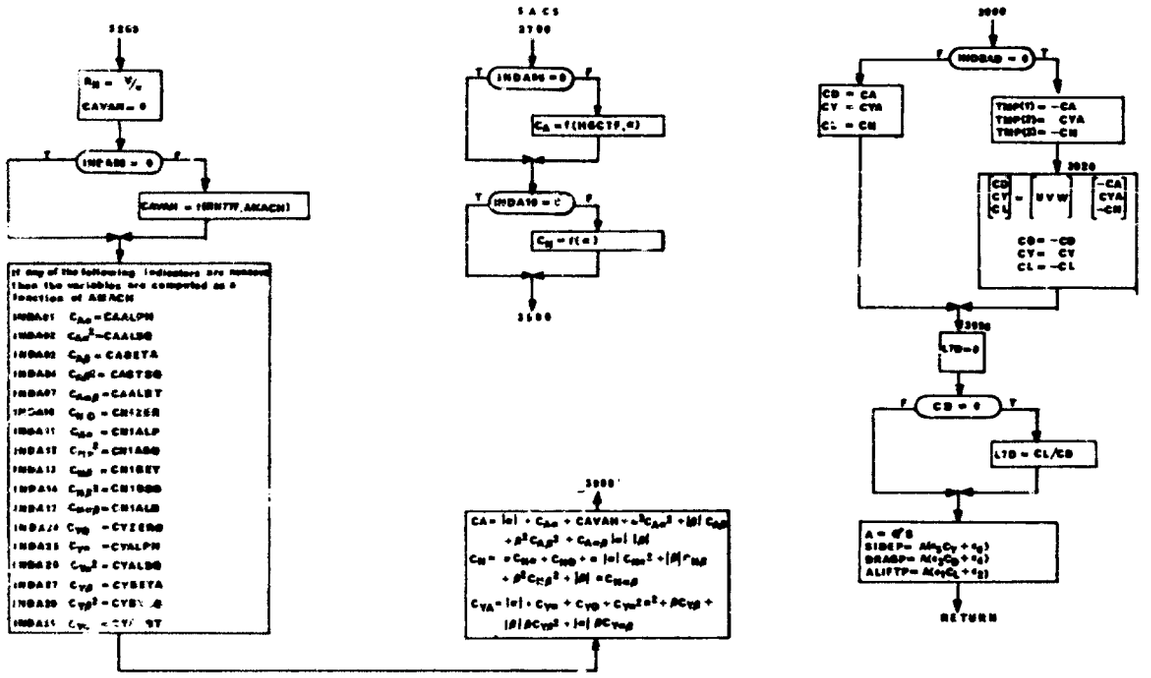
IENTRY = 6

With this entry the corresponding values to the code in entry 5 are printed.

CL	=	Lift force coefficient
CD	=	Drag force coefficient
CY	=	Aerodynamic side force coefficient
L7D	=	This will be 0 if CD is 0, if not L7D = CL/CD
PA77D	=	$\gamma + \alpha \cos B_A$ -AINCD
CPA7D	=	CPAK1·α + CPAK2



(2)



3265
CAVAN = 0
CAVAN = f(RNTF, AMACH)
If any of the following indicators are nonzero then the variables are computed as a function of AMACH
INDA01 $C_{A\alpha}$ = CAALPH
INDA02 $C_{A\alpha}^2$ = CAALSQ
INDA03 $C_{A\beta}$ = CABETA
INDA04 $C_{A\beta}^2$ = CABTSQ
INDA07 $C_{A\alpha\beta}$ = CAALBT
INDA10 C_{N0} = CN1ZER
INDA11 $C_{N\alpha}$ = CN1ALP
INDA12 $C_{N\alpha}^2$ = CN1ASQ
INDA13 $C_{N\beta}$ = CN1BET
INDA14 $C_{N\beta}^2$ = CN1BSQ
INDA17 $C_{N\alpha\beta}$ = CN1ALB
INDA24 C_{Y0} = CYZERO
INDA25 $C_{Y\alpha}$ = CYALPH
INDA26 $C_{Y\alpha}^2$ = CYALSQ
INDA27 $C_{Y\beta}$ = CYBETA
INDA28 $C_{Y\beta}^2$ = CYBTSQ
INDA31 $C_{Y\alpha}$ = CYALBT
SACS
3700
INDA06 = 0
C_A = f(HGCTF, α)
INDA10 = 0
C_N = f(α)
3500
3000
$CA = |\alpha| + C_{A\alpha} + CAVAN + \alpha^2 C_{A\alpha}^2 + |\beta| C_{A\beta} + \beta^2 C_{A\beta}^2 + C_{A\alpha\beta} |\alpha| |\beta|$
$C_N = \alpha C_{N\alpha} + C_{N0} + \alpha |\alpha| C_{N\alpha}^2 + |\beta| C_{N\beta} + \beta^2 C_{N\beta}^2 + |\beta| \alpha C_{N\alpha\beta}$
$C_{YA} = |\alpha| + C_{Y\alpha} + C_{Y0} + C_{Y\alpha}^2 \alpha^2 + \beta C_{Y\beta} + |\beta| \beta C_{Y\beta}^2 + |\alpha| \beta C_{Y\alpha\beta}$
3000
INDBAD = 0
CD = CA
CY = CYA
CL = CN
TMP(1) = −CA
TMP(2) = CYA
TMP(3) = −CN
3020
CD = −CD
CY = CY
CL = −CL
3030
L7D = 0
CD = 0
L7D = CL/CD
A = q*S
SIDEP = A(α3CY + α4)
DRAGP = A(α3CD + α4)
ALIFTP = A(α1CL + α2)
RETURN

78. LATS and LATS2 - Geodetic-Geocentric Conversion

Purpose:

To compute the geocentric latitude in terms of the geodetic latitude and vice versa. LATS is used for vehicle 1; LATS2, for vehicle 2.

Usage:

CALL LATS (IENTRY, PHIC, PHIG, TMP)

IENTRY = 1	compute geocentric latitude in terms of geodetic latitude
= 2	compute geodetic latitude in terms of geocentric latitude
PHIC	the geocentric latitude (degrees)
PHIG	the geodetic latitude (degrees)
TMP	an array of dimension ≥ 7 for temporary calculations

The COMMON variables HCC7F, RP77F, RE77F, R777F are used also.

Method:

IENTRY = 1: h, R_p, R_e, ϕ_g given

$$\phi^1 = \tan^{-1}\left[\left(\frac{R_p}{R_E}\right)^2 \tan\phi_g\right]$$

$$R_{\phi 1} = \frac{R_E R_p}{\sqrt{(R_p\cos\phi^1)^2 + (R_E\sin\phi^1)^2}}$$

$$\phi_c = \tan^{-1}\left[\frac{R_{\phi 1}\sin\phi^1 + h\sin\phi_g}{R_{\phi 1}\cos\phi^1 + h\cos\phi_g}\right]$$

IENTRY = 2: R, R_p, R_E, ϕ_c given

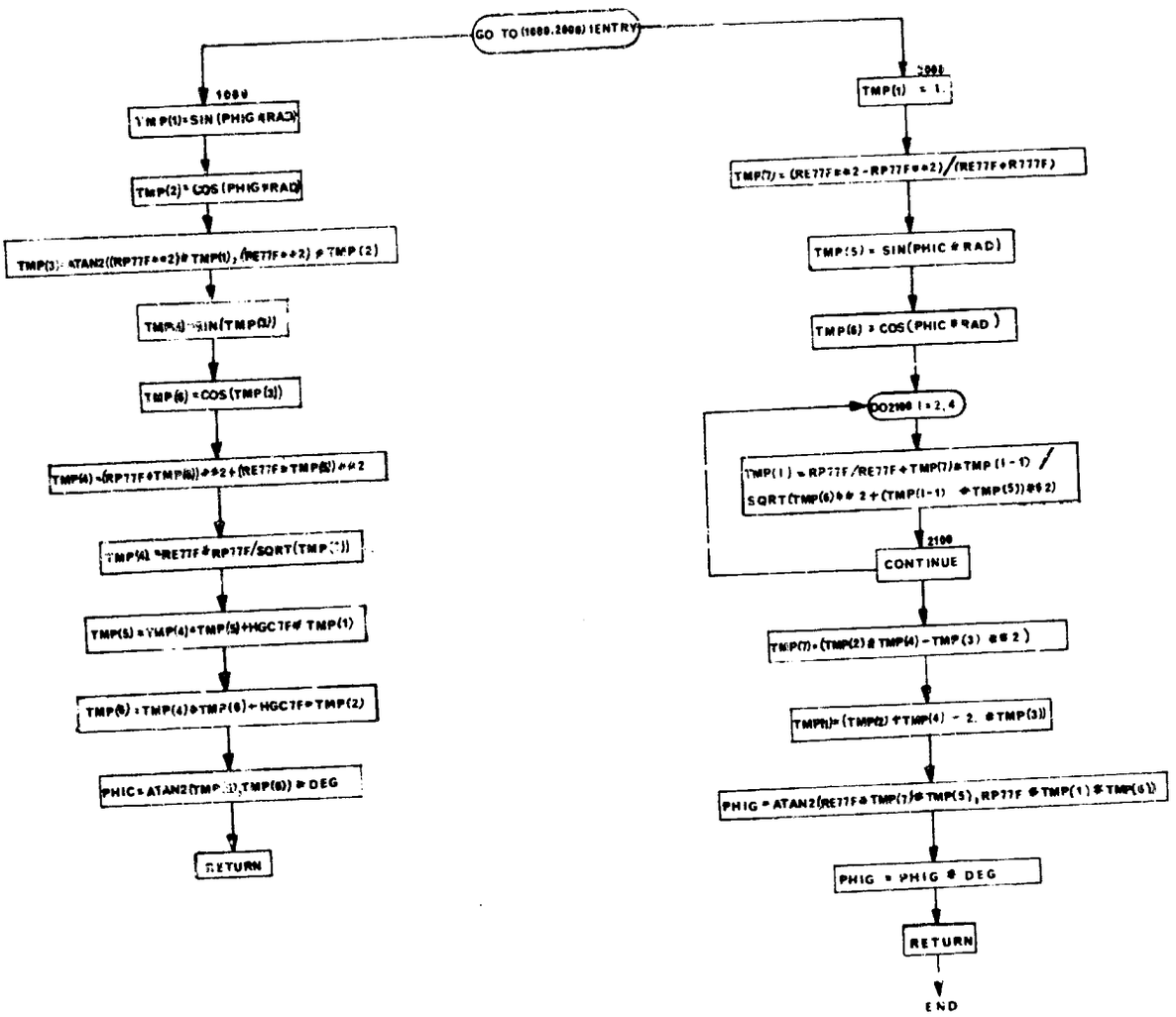
$$U_o = 1$$

$$U_n = \left(\frac{R_p}{R_E}\right) + \left[\frac{R_E^2 - R_p^2}{R_E + R}\right]\left[\frac{U_{n-1}}{\sqrt{\cos^2\phi_c + (U_{n-1}\sin\phi_c)^2}}\right] \quad n = 1,..,3$$

$$U_* = \frac{U_1 U_3 - U_2^2}{U_1 + U_3 - 2U_2}$$

$$\phi_G = \tan^{-1}\left[\frac{R_E}{R_p} U_*\tan\phi_c\right]$$

LATS


GO TO (1000,2000) IENTRY
1000
TMP(1)=SIN (PHIG*RAD)
TMP(2)=COS (PHIG*RAD)
TMP(3)=ATAN2((RP77F**2)*TMP(1),(RE77F**2)*TMP(2)
TMP(4)=SIN(TMP(3))
TMP(5)=COS(TMP(3))
TMP(6)=(RP77F*TMP(4))**2+(RE77F*TMP(5))**2
TMP(4)=RE77F*RP77F/SQRT(TMP(6))
TMP(5)=TMP(4)*TMP(5)+HGC7F*TMP(1)
TMP(6)=TMP(4)*TMP(6)+HGC7F*TMP(2)
PHIC=ATAN2(TMP(5),TMP(6))*DEG
RETURN
2000
TMP(1)=1.
TMP(7)=(RE77F**2-RP77F**2)/(RE77F*R777F)
TMP(5)=SIN(PHIC*RAD)
TMP(6)=COS(PHIC*RAD)
DO2100 I=2,4
TMP(I)=RP77F/RE77F*TMP(7)*TMP(I-1)/
SQRT(TMP(6)**2+(TMP(I-1)*TMP(5))**2)
2100
CONTINUE
TMP(7)=(TMP(2)*TMP(4)-TMP(3)**2)
TMP(1)=(TMP(2)*TMP(4)-2.*TMP(3))
PHIG=ATAN2(RE77F*TMP(7)*TMP(5),RP77F*TMP(1)*TMP(6))
PHIG=PHIG*DEG
RETURN
END

79. ATMS and ATMS2 - Atmosphere Selector

Purpose:

To enable the user to select which atmosphere (1959 or the 1962 atmosphere) that will be used by the program.

Usage:

CALL ATMS (HGC7F)

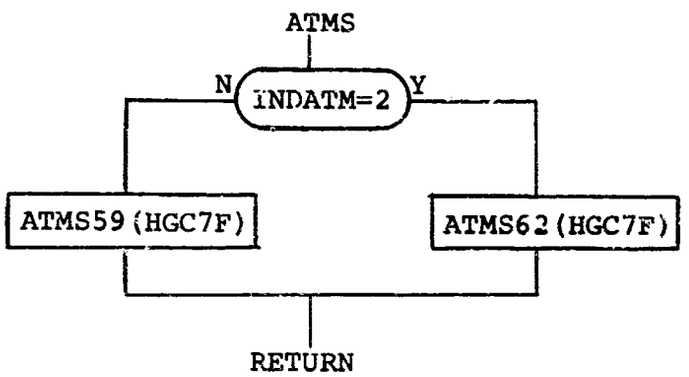
HGC7F = The initial altitude that will be used by ATMS59 or ATMS62.

INDATM = 2 selection of the 1962 atmosphere.

INDATM ≠ 2 selection of the 1959 atmosphere.

Remarks

ATMS59 or ATMS62 is called only by DIFEQ1. A flow chart for ATMS is presented. ATMS2 is identical except for the use of vehicle 2 COMMON blocks and subroutines.



80. GVSP and GVSP2 - Gravitational Routine

Purpose

To compute the components of gravitational acceleration within a local geocentric coordinate system.

Usage

CALL GVSP

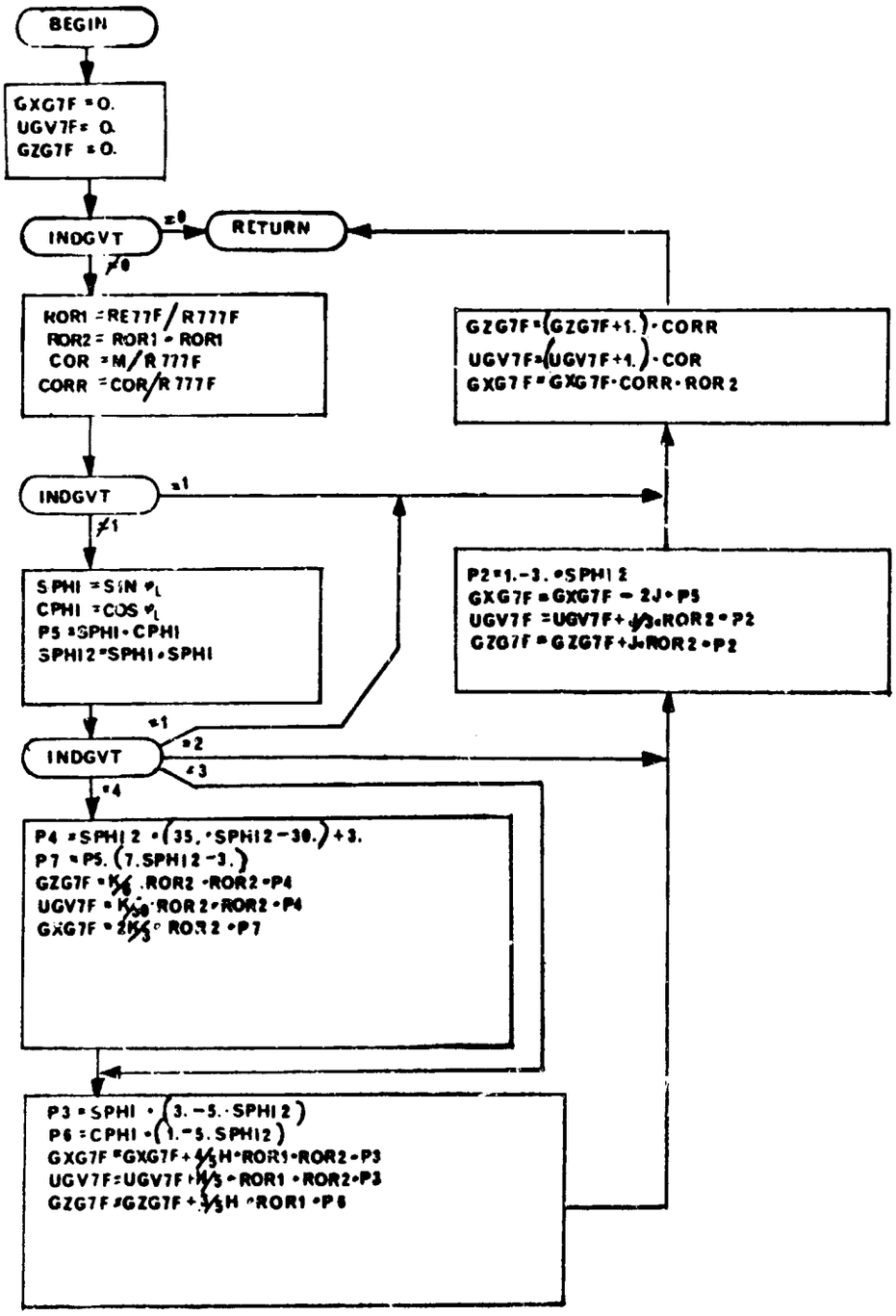
The equations used were adapted from the gravity potential equation. The number of spherical harmonics included in the computation is determined by inputing an integral value from 0 to 4 for INDGVT. The nominal value for INDGVT is 4.

The equations pertain to the planet Earth; however, it is possible to use these same equations for any other planet. For this reason, the values of the coefficients are programmed as an input to the program so that the applicable coefficients may be inserted for the planet under consideration. The following give the necessary planet change as well as the nominal values for the planet Earth.

RE777	$= R_e$	$= 20926428.$ ft.	= equatorial radius
RP77F	$= R_p$	$= 20855965.$ ft.	= polar radius
AMU	$= \mu$	$= 1.407698 \times 10^{16}$ ft^3/sec^2	= gravity constant
AKGRAV	$= K$	$= 6.37 \times 10^{-6}$	= 3rd spherical harmonic
HG	$= H$	$= 6.04 \times 10^{-6}$	= 2nd spherical harmonic
AJG	$= J$	$= 1.62341 \times 10^{-3}$	= 1st spherical harmonic

Remarks:

A flow chart for GVSP is presented. GVSP2 is identical except for the use of vehicle 2 COMMON blocks.


BEGIN
GXG7F = 0.
UGV7F = 0.
GZG7F = 0.
INDGVT
=0
RETURN
≠0
ROR1 = RE77F / R777F
ROR2 = ROR1 * ROR1
COR = M / R777F
CORR = COR / R777F
GZG7F = (GZG7F+1.) * CORR
UGV7F = (UGV7F+1.) * COR
GXG7F = GXG7F * CORR * ROR2
INDGVT
=1
≠1
SPHI = SIN φL
CPHI = COS φL
P5 = SPHI * CPHI
SPHI2 = SPHI * SPHI
P2 = 1. - 3. * SPHI2
GXG7F = GXG7F - 2J * P5
UGV7F = UGV7F + J/3 * ROR2 * P2
GZG7F = GZG7F + J * ROR2 * P2
INDGVT
=1
=2
≠3
=4
P4 = SPHI2 * (35. * SPHI2 - 30.) + 3.
P7 = P5 * (7. SPHI2 - 3.)
GZG7F = K/8 * ROR2 * ROR2 * P4
UGV7F = K/30 * ROR2 * ROR2 * P4
GXG7F = 2K/5 * ROR2 * P7
P3 = SPHI * (3. - 5. * SPHI2)
P6 = CPHI * (1. - 5. SPHI2)
GXG7F = GXG7F + 1/3 H * ROR1 * ROR2 * P3
UGV7F = UGV7F + 1/3 * ROR1 * ROR2 * P3
GZG7F = GZG7F + 1/3 H * ROR1 * P6

81. ANITR and ANITR2 - Throttle Dependent Derivative and Thrust Vector Calculation

Purpose:

ANITR and ANITR2 compute that portion of the derivative calculation which directly depends on the throttle setting and thrust angles.

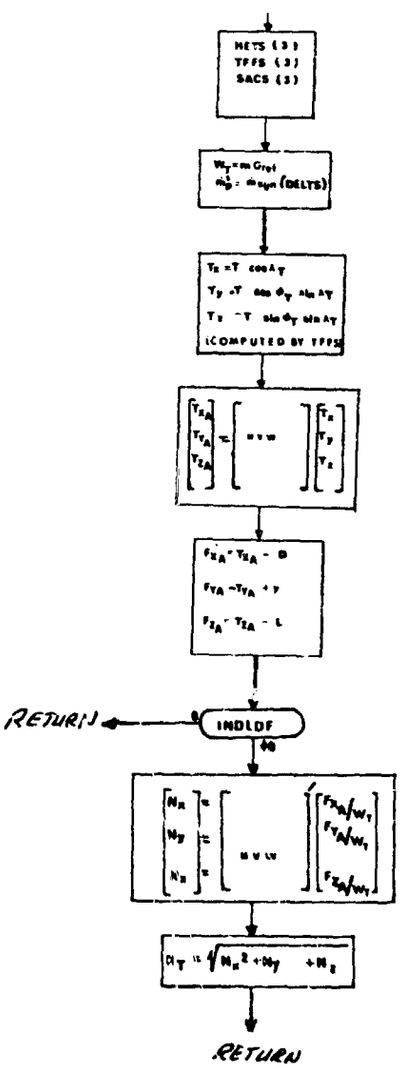
Method:

Propulsion system effects are computed from TFFS and TFFS2. A thrust vector transformation accounts for the possibility of thrust vector control. Load factors are optionally computed. Heating and aerodynamic routines HETS, HETS2, SACS, and SACS2 are called from ANITR and ANITR2 in anticipation of subsequent requirements to account for propulsion system interaction on aerodynamics and heating.

Remarks:

A flow chart for ANITR is provided. ANITR2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.

ANITR



NOT REPRODUCIBLE

82. BAITR and BAITR2 - Bank Angle Dependent Derivative Calculation

Purpose:

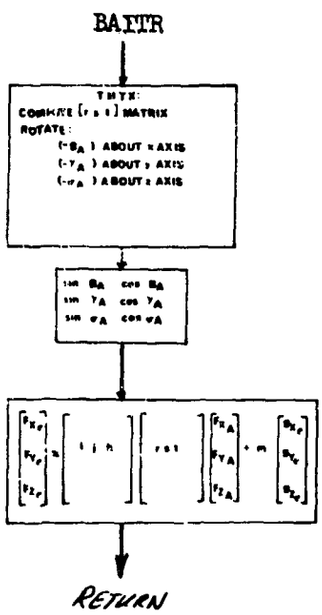
To carry out that portion of the derivative calculation which is directly dependent on the bank angle.

Method:

Computes the (r,s,t) matrix through TMTX or TMTX2 then computes total force components in the rotating earth-centered coordinate system, (X_E, Y_E, Z_E)

Remarks:

A flow chart for BAITR is provided. BAITR2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



83. ASRCH and ASRCH2 - Directory Search Routines for BCD Characters

Purpose:

To provide a BCD word look up from a subscript.

Method:

Given a subscript the routine will search the directory for the BCD word corresponding to that subscript. If the subscripts do not compare the BCD name is set to blank and return to the Calling Program is made.

Usage:

Entry is made to the routine with the following statement:

```
CALL ASRCH (LOC1,SYM1)
```

where,

LOC1 = subscript being searched.

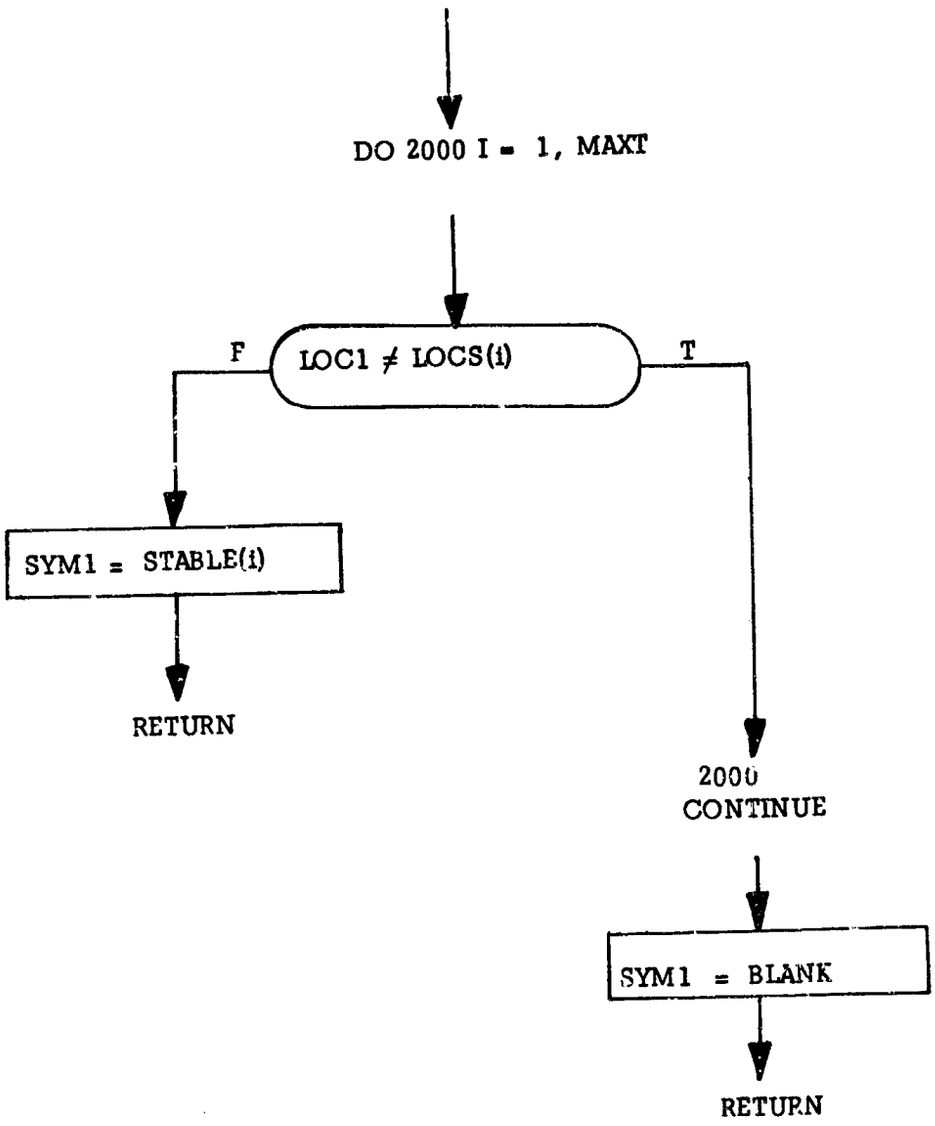
SYM1 = The variable name into which the routine is to store the corresponding BCD name.

No other routines are called from this routine.

Remarks:

A flow chart for ASRCH is presented. ASRCH2 is identical except for use of vehicle 2 COMMON blocks.

ASRCH


DO 2000 I = 1, MAXT
F
LOC1 ≠ LOCS(i)
T
SYM1 = STABLE(i)
RETURN
2000
CONTINUE
SYM1 = BLANK
RETURN

84. GRIDXY - Paper Plot Grid Routine

Purpose:

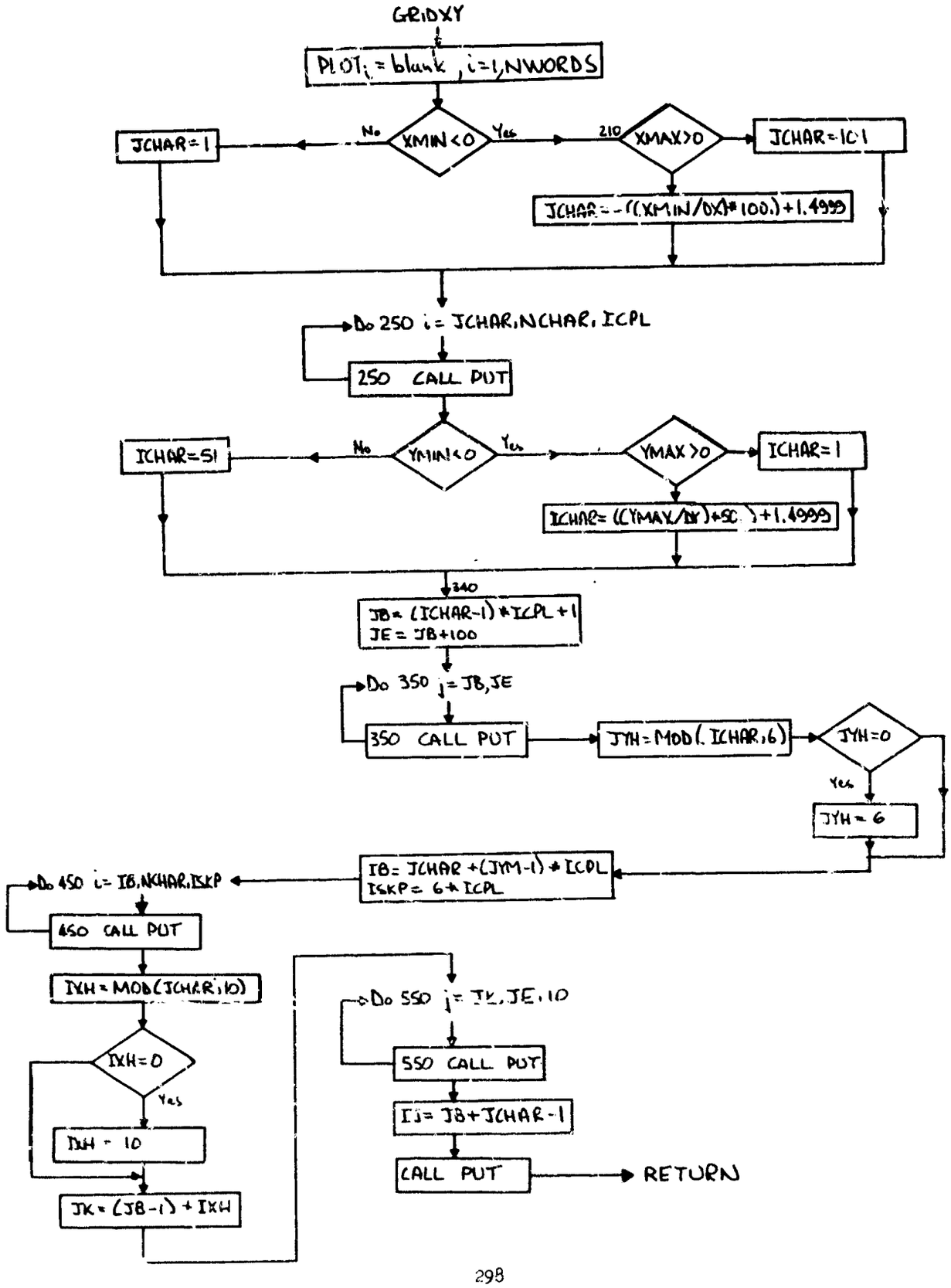
To set up the grid for paper plots

Usage:

Call GRIDXY (PLOT, XMIN, XMAX, YMIN, YMAX, DX, DY)

Where

PLOT	is an array into which the plot characters and grid characters are placed to produce the finished plot. The finished plot will contain 51 lines of 102 characters per line.
XMIN	minimum x value for the grid
XMAX	maximum x value for the grid
YMIN	minimum y value for the grid
YMAX	maximum y value for the grid
DX	number of divisions on x axis
DY	number of divisions on y axis


GRIDXY
PLOT_i = blank , i=1,NWORDS
XMIN<0
No
Yes
JCHAR=1
210
XMAX>0
JCHAR=101
JCHAR=-((XMIN/DX)*100.)+1.4999
Do 250 i = JCHAR,NCHAR, ICPL
250 CALL PUT
YMIN<0
No
Yes
ICHAR=51
YMAX>0
ICHAR=1
ICHAR=((YMAX/DY)+50)+1.4999
340
JB=(ICHAR-1)*ICPL+1
JE=JB+100
Do 350 j = JB,JE
350 CALL PUT
JYH=MOD(.ICHAR,6)
JYH=0
Yes
JYH=6
IB=JCHAR+(JYM-1)*ICPL
ISKP=6*ICPL
Do 450 i = IB,NCHAR,ISKP
450 CALL PUT
IXH=MOD(JCHAR,10)
IXH=0
Yes
IXH=10
JK=(JB-1)+IXH
Do 550 j = JK,JE,10
550 CALL PUT
I1=JB+JCHAR-1
CALL PUT
RETURN

85. PLCPTS - Paper Plot Point Placing Program

Purpose:

To place plotted points on the grid formed by GRIDXY.

Usage:

Call PLCPTS (PLOT, X, Y, NPTS, IPATH, IPLOT, NFRAME, IFLAT, XMIN, YMAX, XP, YP)

Where

PLOT is an array into which the plot characters and grid characters are placed to produce the finished plot. The finished plot will contain 51 lines of 102 characters per line.

X is the array that contains the values of the independent variable to be plotted

Y is the array that contains the values of the dependent variable to be plotted

NPTS number of points to be plotted

IPATH not used

IPLOT subscript used to select plot symbol

NFRAME frame number to be printed at the bottom of the plot

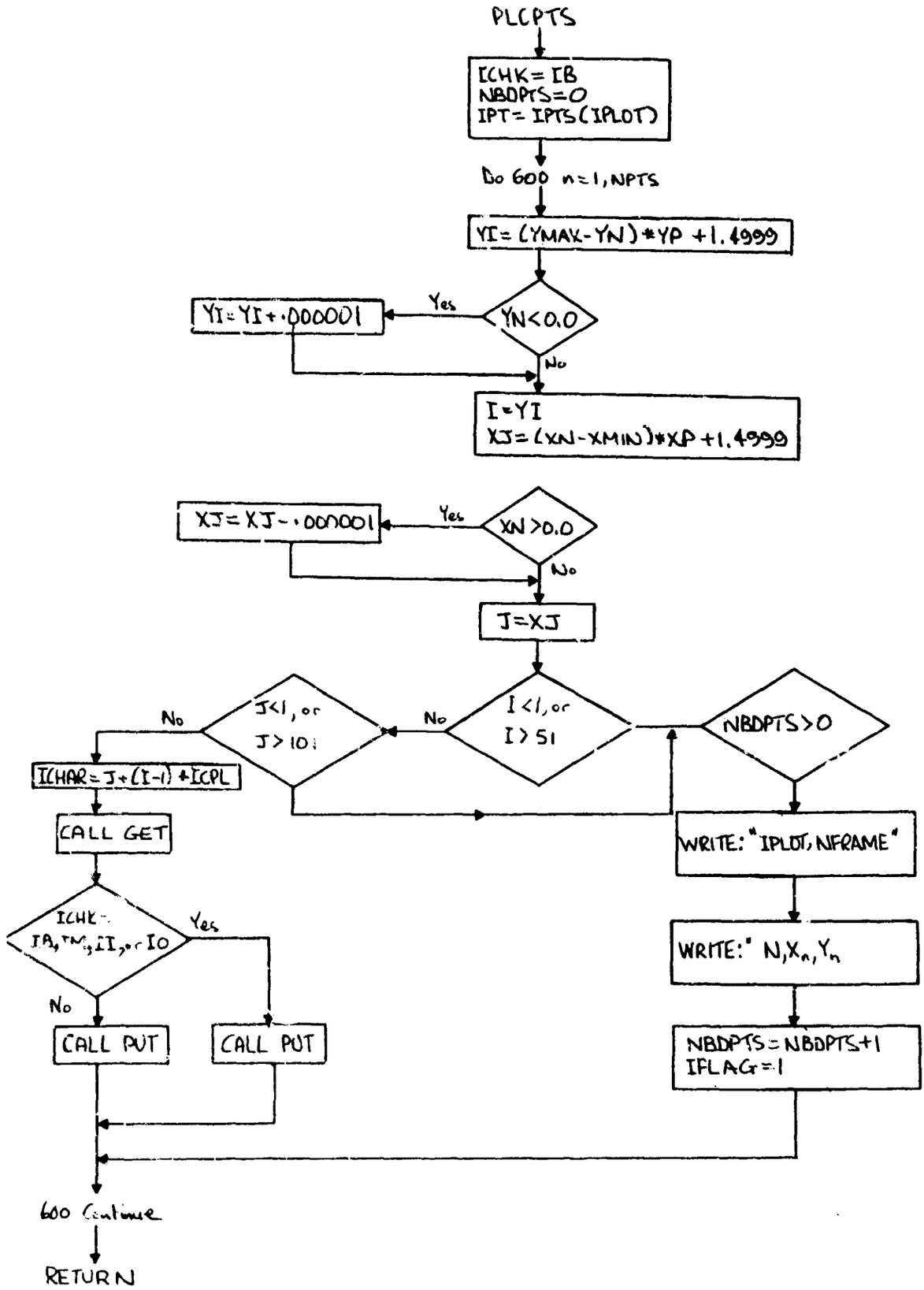
IFLAG flag set equals 1 if points have fallen off the plotting area

XMIN minimum x value

XMAX maximum y value

XP x plot increment

YP y plot increment


PLCPTS
ICHK = IB
NBDPTS = 0
IPT = IPTS(IPLOT)
Do 600 n=1,NPTS
YI = (YMAX-YN)*YP +1.4999
YN<0.0
Yes
YI = YI + .000001
No
I = YI
XJ = (XN-XMIN)*XP +1.4999
XN>0.0
Yes
XJ = XJ - .000001
No
J = XJ
I<1, or I>51
No
J<1, or J>10
No
ICHAR = J + (I-1) * ICPL
CALL GET
ICHK = IA, TM, II, or IO
Yes
No
CALL PUT
CALL PUT
600 Continue
RETURN
NBDPTS>0
WRITE: "IPLOT, NFRAME"
WRITE: " N, Xn, Yn
NBDPTS = NBDPTS+1
IFLAG = 1

86. IPICK - Random Tactic Selector

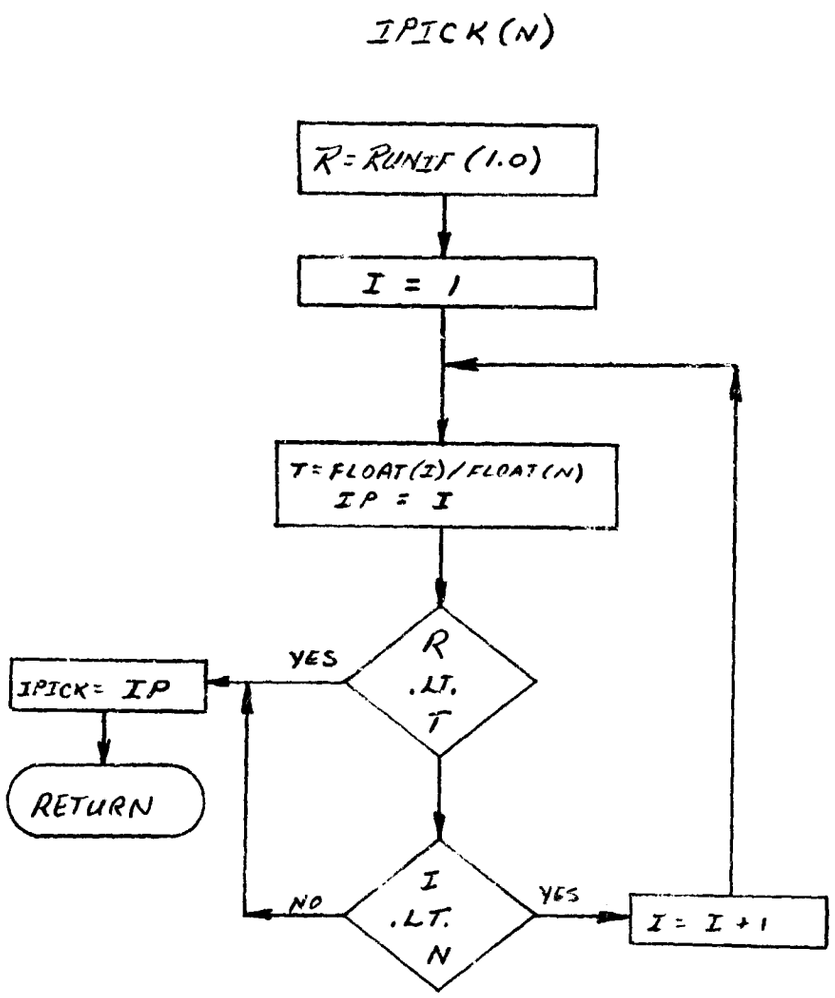
Purpose:

Subroutine IPICK nominally fills both vehicle 1 and vehicle 2 role selection tables with random arrays of tactics.

Usage:

CALL IPICK(N)

where N is the number of random tactics to be selected.



87. DEFEN11 and DEFEN21 - First Defensive Tactic

Purpose:

To steer each vehicle along a path giving a maximum rate of turn. The tactic is a hard turn in the vertical plane.

Remarks:

A flow chart for DEFEN11 is presented; DEFEN12 is identical except for use of vehicle 2 COMMON blocks.

DEFEN12 and DEFEN22 - Second Defensive Tactic

Purpose:

DEFEN12 and DEFEN22 are ENTRY points in DEFEN11 and DEFEN22, respectively. The tactic is a hard turn *into* the opponent at an altitude dependent bank-angle. A flow chart for DEFEN12 is presented. DEFEN22 is identical except for use of vehicle 2 COMMON blocks.

DEFEN13 and DEFEN23 - Third Defensive Tactic

Purpose:

DEFEN13 and DEFEN23 are ENTRY points in DEFEN11 and DEFEN21, respectively. The tactic restates the target's line-of-sight vector at a maximum possible rate. Local angle-of-attack and bank-angle perturbations determine control values which maximize the vector product magnitude,

$$|\overline{LOS}_T \times \overline{F}_T|$$

where $\overline{F}_T$ is the total force vector, and $\overline{LOS}_T$ is the *target's* line-of-sight vector. A flow chart for DEFEN13 is presented. DEFEN23 is identical except for the use of vehicle 2 COMMON blocks and auxiliary routines.

DEFEN14 and DEFEN24 - Fourth Defensive Tactic

Purpose:

DEFEN14 and DEFEN24 are ENTRY points in DEFEN11 and DEFEN21, respectively. The tactic rotates the *target's* lead-pursuit vector at a maximum rate. Local angle-of-attack and bank-angle perturbations determine control values which maximize the vector product magnitude

$$|\overline{LA}_T \times \overline{F}_T|$$

where $\overline{F}_T$ is the total force vector, and $\overline{LA}_T$ is the *target's* lead-pursuit vector. A flow chart for DEFEN14 is presented. DEFEN24 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

DEFEN15 and DEFEN25 - Fifth Defensive Tactic

Purpose:

DEFEN15 and DEFEN25 are ENTRY points in DEFEN11 and DEFEN21, respectively. The tactic rotates a proportional vector, $\bar{V}_R$, which is a linear combination of *target's* lead-pursuit angle and *target's* line-of-sight vectors at a maximum possible rate. Local angle-of-attack and bank-angle perturbations determine control values which maximize the vector product magnitude

$$|\bar{V}_R \times \bar{F}_T|$$

Here, $\bar{F}_T$ is the total force, and $\bar{V}_R$ is the *target's* reference vector.

Remarks:

A flow chart for DEFEN15 is presented. DEFEN25 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

DEFEN16 and DEFEN26 - Sixth Defensive Tactic

Purpose:

DEFEN16 and DEFEN26 are ENTRY points in DEFEN11 and DEFEN21, respectively. The tactic performs a "Split-S" under an opponent. The initial maneuver through the vertical is a simple Split-S. Once through the vertical has been accomplished, the reference vehicle maintains the Split-S but banks underneath the opponent.

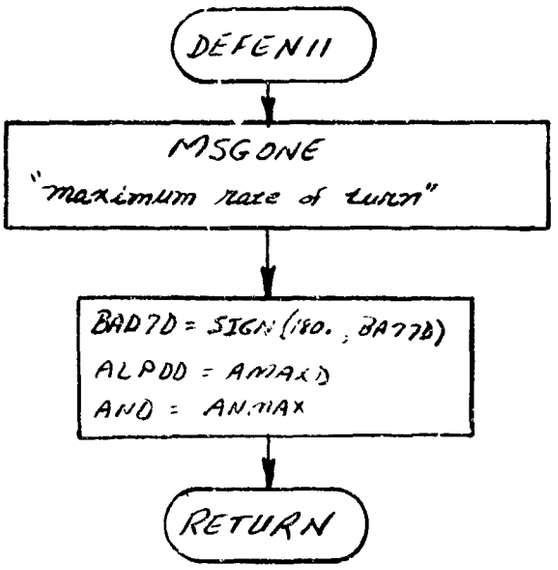
Remarks:

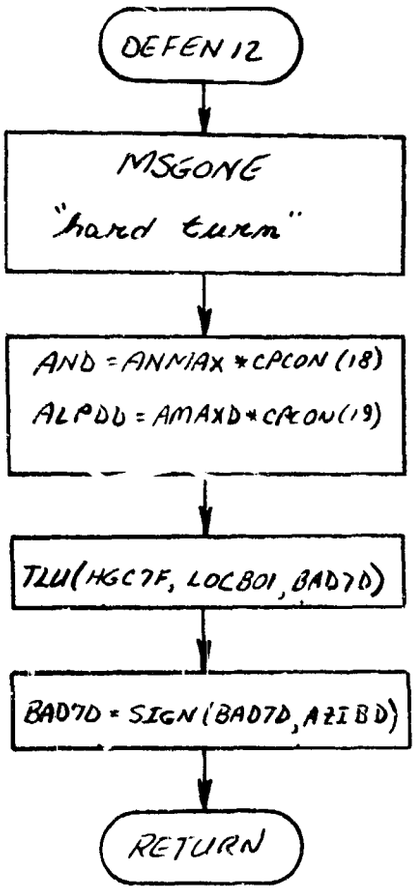
A flow chart for DEFEN16 is presented. DEFEN26 is identical except for use of vehicle 2 COMMON blocks and auxiliary routines.

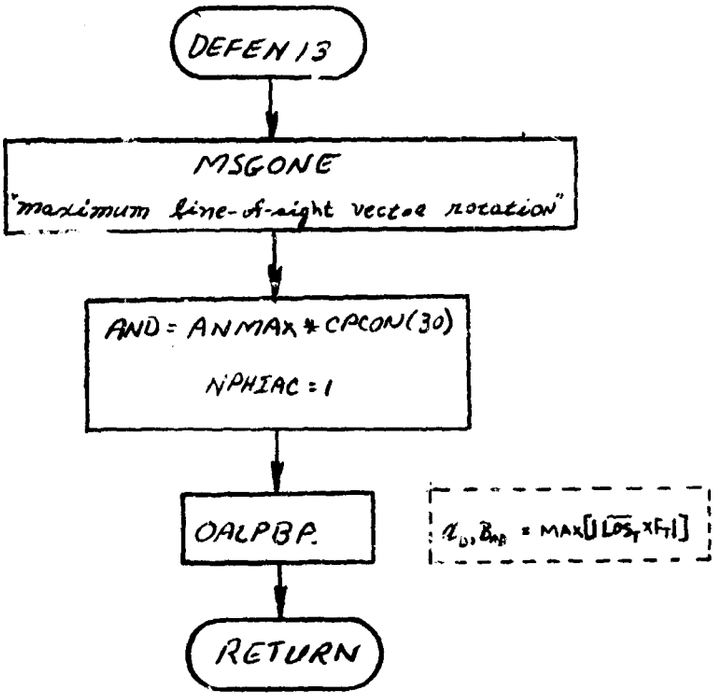
DEFEN17 and DEFEN27 - Seventh Defensive Tactic

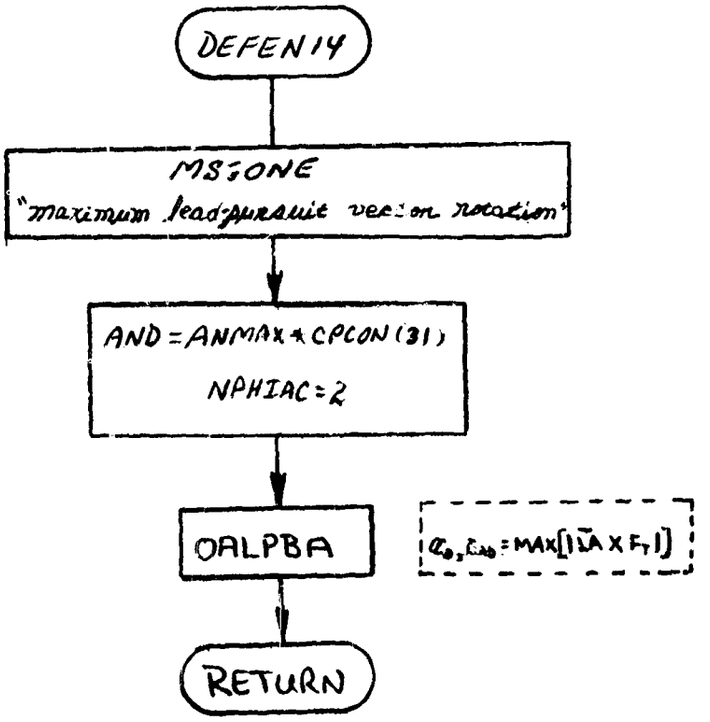
Purpose:

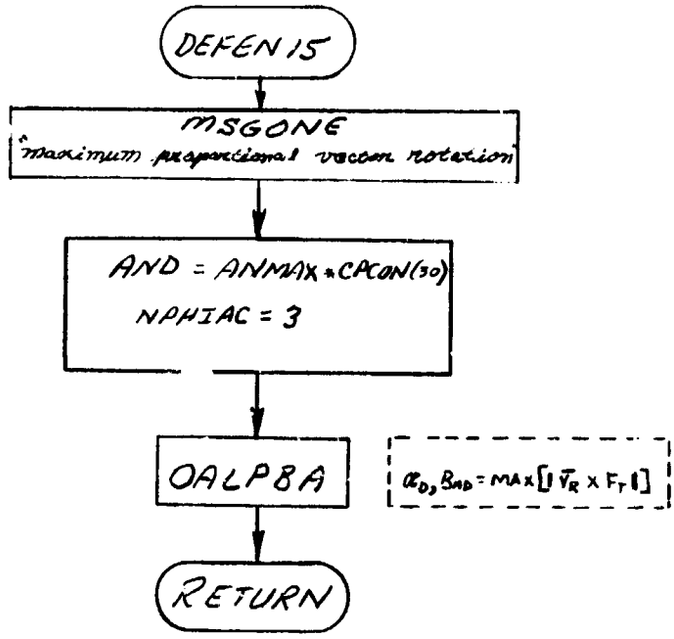
DEFEN17 and DEFEN27 are ENTRY points in DEFEN11 and DEFEN21, respectively. The tactic develops a maximum specific energy path in a vertical plane. Angle-of-attack,which maximizes $\dot{E}$ at time ΔT ahead along the predicted flight path,is determined by local control perturbation. The tactic includes logic to force a transfer to supersonic conditions. A flow chart forDEFEN17 is presented. DEFEN27 is identical except for use of vehicle 2 COMMON blocks and auxiliary routines.

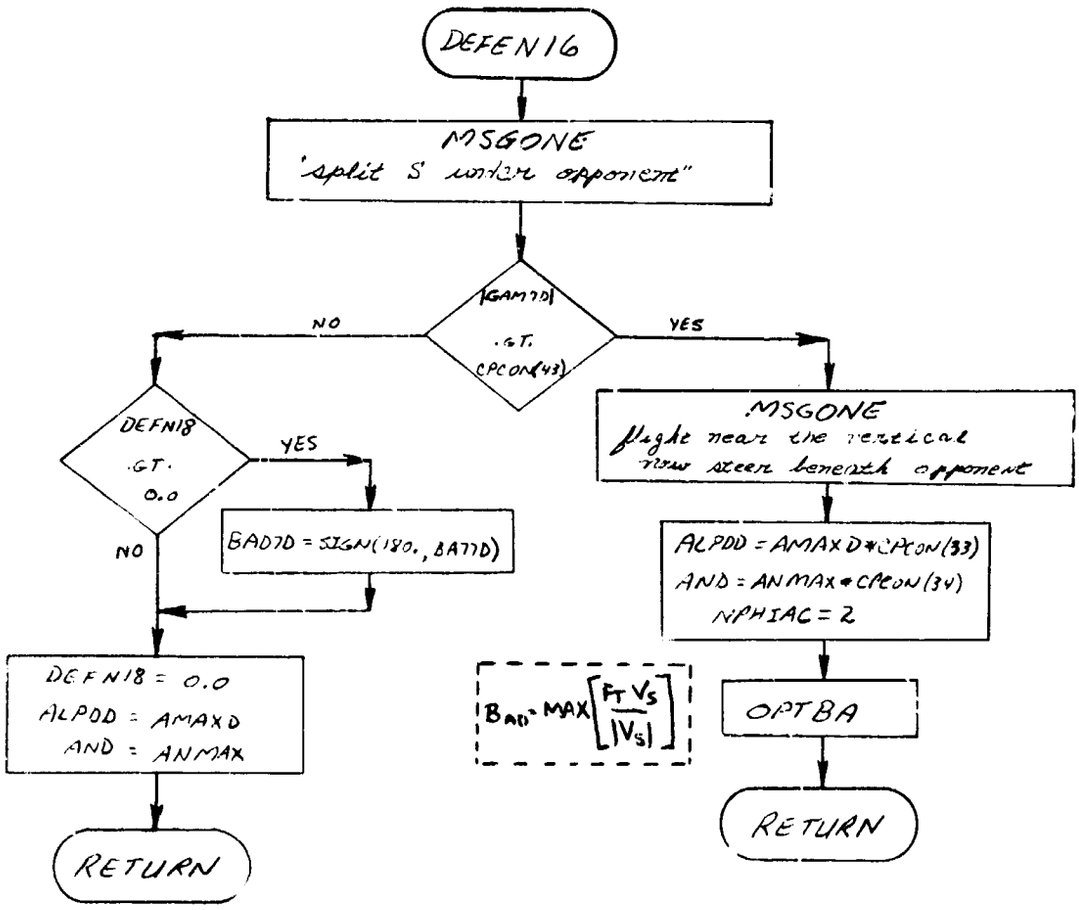

DEFEN11
MSGONE
"maximum rate of turn"
BAD7D = SIGN(180., BA77B)
ALPDD = AMAXD
AND = ANMAX
RETURN

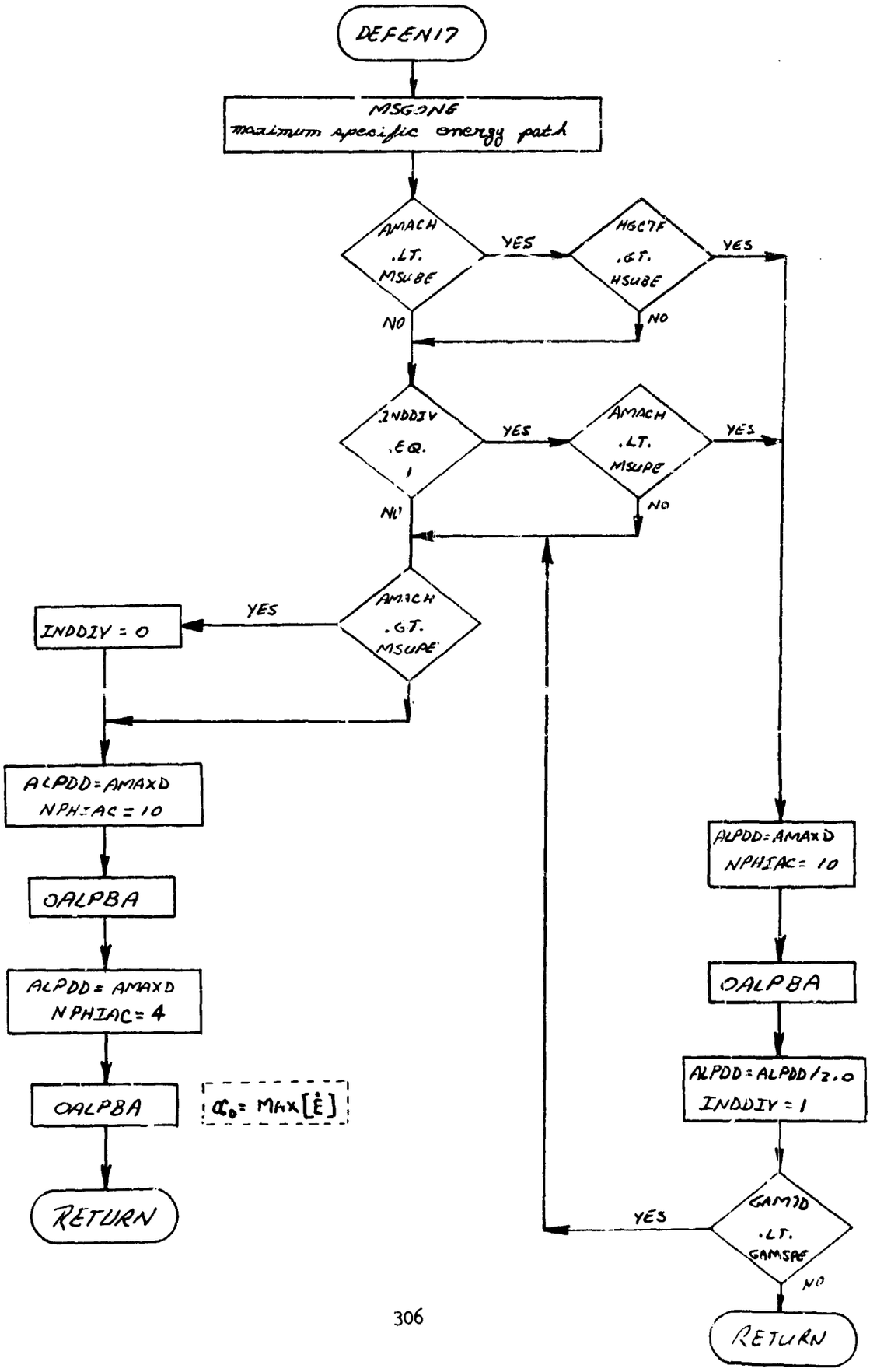

DEFEN12
MSGONE
"hard turn"
AND = ANMAX * CPCON(18)
ALPDD = AMAXD * CPCON(19)
TLU(HGC7F, LOCB01, BAD7D)
BAD7D = SIGN(BAD7D, AZIBD)
RETURN


DEFEN13
MSGONE
"maximum line-of-sight vector rotation"
AND = ANMAX * CPCON(30)
NPHIAC = 1
OALPBP
RETURN


DEFEN14
MSGONE
"maximum lead-pursuit vector rotation"
AND = ANMAX * CPCON(31)
NPHIAC = 2
OALPBA
RETURN


DEFEN15
MSGONE
"maximum proportional vector rotation"
AND = ANMAX * CPCON(30)
NPHIAC = 3
OALPBA
$\alpha_D, B_{AD} = MAX[|\bar{V}_R \times F_T|]$
RETURN


DEFEN16
MSGONE
"split S under opponent"
|GAM7D| .GT. CPCON(43)
NO
YES
DEFN18 .GT. 0.0
YES
NO
BAD7D = SIGN(180., BA77D)
DEFN18 = 0.0
ALPDD = AMAXD
AND = ANMAX
RETURN
MSGONE
flight near the vertical
now steer beneath opponent
ALPDD = AMAXD * CPCON(33)
AND = ANMAX * CPCON(34)
NPHIAC = 2
$B_{AD} = MAX\left[\frac{F_T V_S}{|V_S|}\right]$
OPTBA
RETURN


DEFEN17
MSGONE
maximum specific energy path
AMACH .LT. MSUBE
YES
HGCTF .GT. HSUBE
YES
NO
NO
INDDIV .EQ. 1
YES
AMACH .LT. MSUPE
YES
NO
NO
AMACH .GT. MSUBE
YES
INDDIV = 0
ALPDD=AMAXD
NPHIAC = 10
OALPBA
ALPDD = AMAXD
NPHIAC = 4
OALPBA
RETURN
α_0 = MAX[$\dot{E}$]
ALPDD=AMAXD
NPHIAC = 10
OALPBA
ALPDD = ALPDD/2.0
INDDIV = 1
GAMTD .LT. GAMSAE
YES
NO
RETURN

88 FIXEDR and FIXEDR2 - Fixed Role Selection Routine

Purpose:

To provide a means for overriding the combat role selection logic.

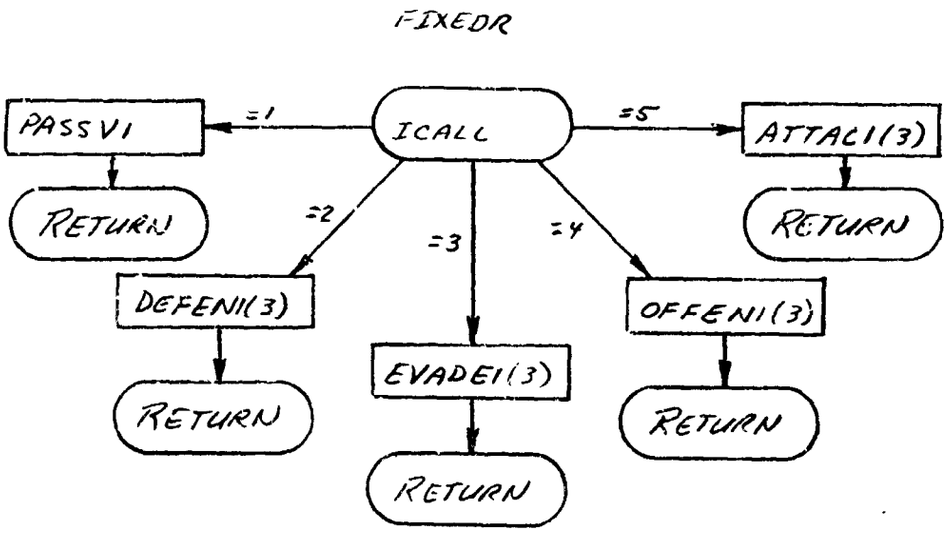
Method:

When the tactic selection indicator, $MOFFEN_i$ is negative, the role selection is overridden,and the role is selected on the basis of $MOFFEN_i$ as follows:

$MOFFEN_i$ = -1, Passive
= -2, Defensive
= -3, Evasive
= -4, Offensive
= -5, Attacking

Remarks:

A flow chart for FIXEDR is presented. FIXEDR2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



89. EVADE11 and EVADE21 - First Evasive Tactic

Purpose:

To steer each vehicle into a hard pull-up maneuver. A flow chart for EVADE11 is presented. EVADE21 is identical except for use of vehicle 2 COMMON blocks.

EVADE12 and EVADE22 - Second Evasive Tactic

Purpose:

EVADE12 and EVADE22 are ENTRY points in EVADE11 and EVADE21, respectively. They steer each vehicle in a random weaving maneuver. A flow chart for EVADE12 is presented. EVADE22 is identical except for use of vehicle 2 COMMON blocks.

EVADE13 and EVADE23 - Third Evasive Tactic

Purpose:

EVADE12 and EVADE23 are ENTRY points in EVADE11 and EVADE21. They steer each vehicle into a Split-S. A flow chart for EVADE13 is presented. EVADE23 is identical except for the use of vehicle 2 COMMON blocks.

EVADE14 and EVADE24 - Fourth Evasive Tactic

Purpose:

EVADE14 and EVADE24 are ENTRY points in EVADE11 and EVADE21. They steer each vehicle in a random sequence of hard turns. The direction of the turns is changed at intervals of TNEWBA. A flow chart for EVADE14 is presented. EVADE24 is identical except for use of vehicle 2 COMMON blocks.

EVADE15 and EVADE25 - Fifth Evasive Tactic

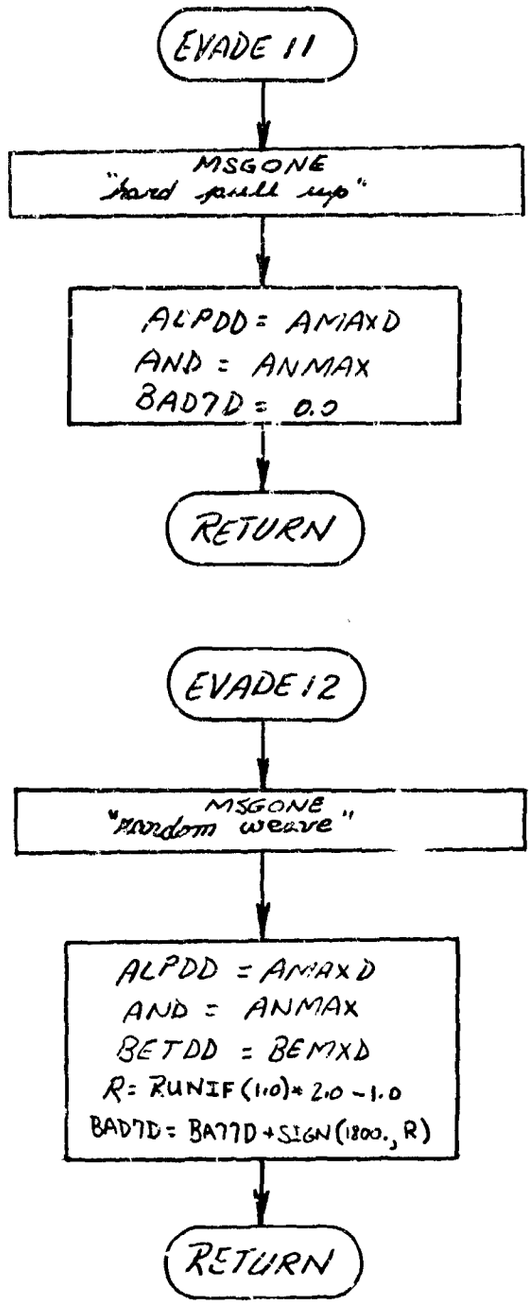
Purpose:

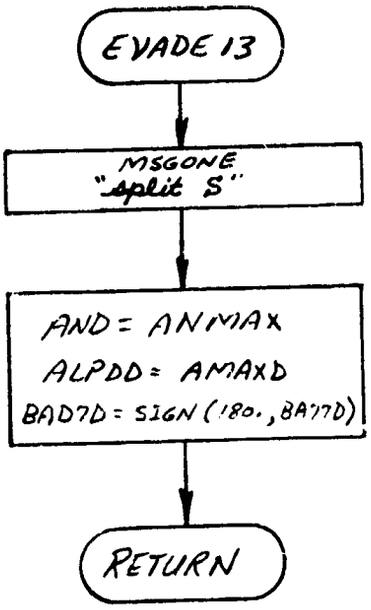
EVADE15 and EVADE25 are ENTRY points in EVADE11 and EVADE21. They steer each vehicle towards a vertical dive. At $\gamma = 90 - \alpha_{max}$, the maneuvers become zero lift dives. A flow chart for EVADE15 is presented. EVADE25 is identical except for use of vehicle 2 COMMON blocks.

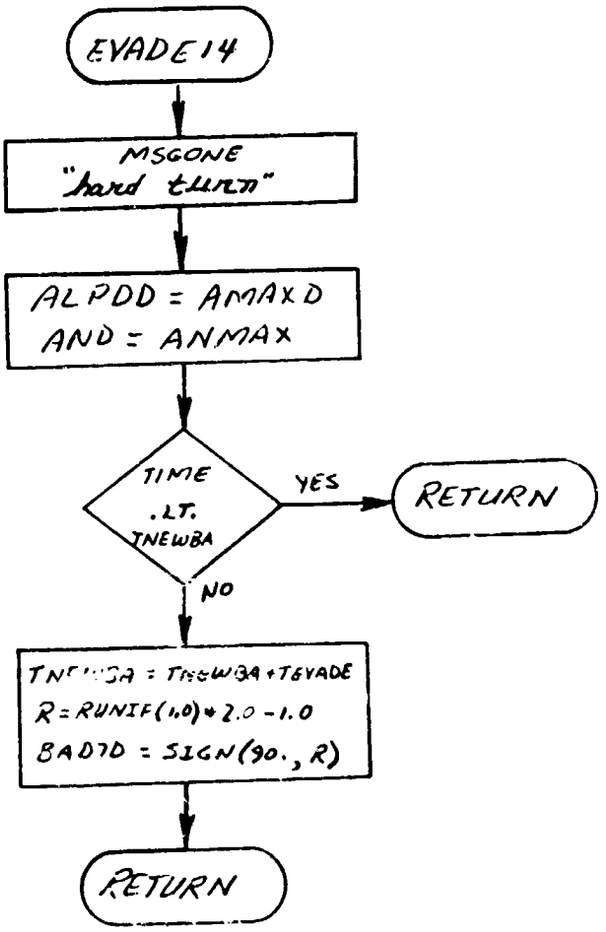
EVADE16 and EVADE26 - Sixth Evasive Tactic

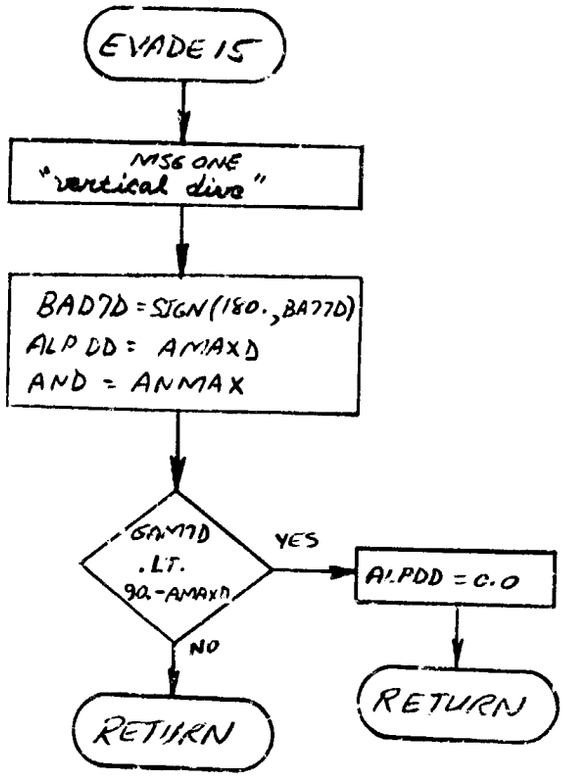
Purpose:

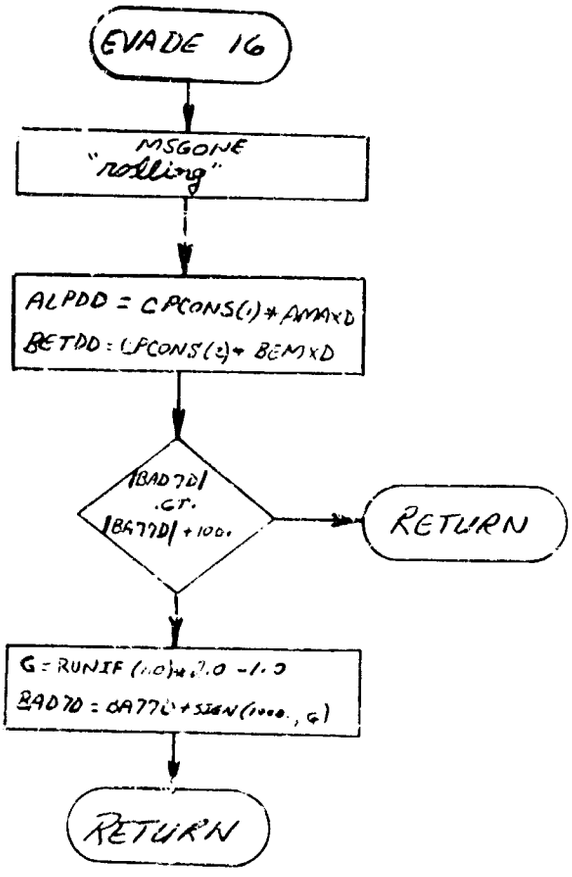
EVADE16 and EVADE26 are entry points in EVADE11 and EVADE21, respectively. They steer each vehicle into a random rolling maneuver with maximum angle-of-attack and sideslip. A flow chart for EVADE16 is presented. EVADE26 is identical except for use of vehicle 2 COMMON blocks.


EVADE 11
MSGONE
"hard pull up"
ALPDD = AMAXD
AND = ANMAX
BADTD = 0.0
RETURN
EVADE 12
MSGONE
"random weave"
ALPDD = AMAXD
AND = ANMAX
BETDD = BEMXD
R = RUNIF(1.0) * 2.0 - 1.0
BADTD = BATTD + SIGN(1800., R)
RETURN


EVADE 13
MSGONE
"split S"
AND = ANMAX
ALPDD = AMAXD
BADTD = SIGN (180., BATTD)
RETURN


EVADE 14
MSGONE
"hard turn"
ALPDD = AMAXD
AND = ANMAX
TIME
.LT.
TNEWBA
YES
RETURN
NO
TNEWBA = TNEWBA + TEVADE
R = RUNIF(1.0) * 2.0 - 1.0
BADTD = SIGN(90., R)
RETURN


EVADE 15
MSG ONE
"vertical dive"
BADTD = SIGN(180., BATTD)
ALPDD = AMAXD
AND = ANMAX
GAMTD .LT. 90.-AMAXD
YES
ALPDD = 0.0
NO
RETURN
RETURN


EVADE 16
MSGONE
"rolling"
ALPDD = CPCONS(1) * AMAXD
BETDD = CPCONS(2) * BEMXD
|BADTD| .GT. |BATTD| + 100.
RETURN
G = RUNIF (1.0) * 2.0 - 1.0
BADTD = BATTD + SIGN (1000., G)
RETURN

90. OFFEN11 and OFFEN21 - Lag-Pursuit Offensive Tactic

Purpose:

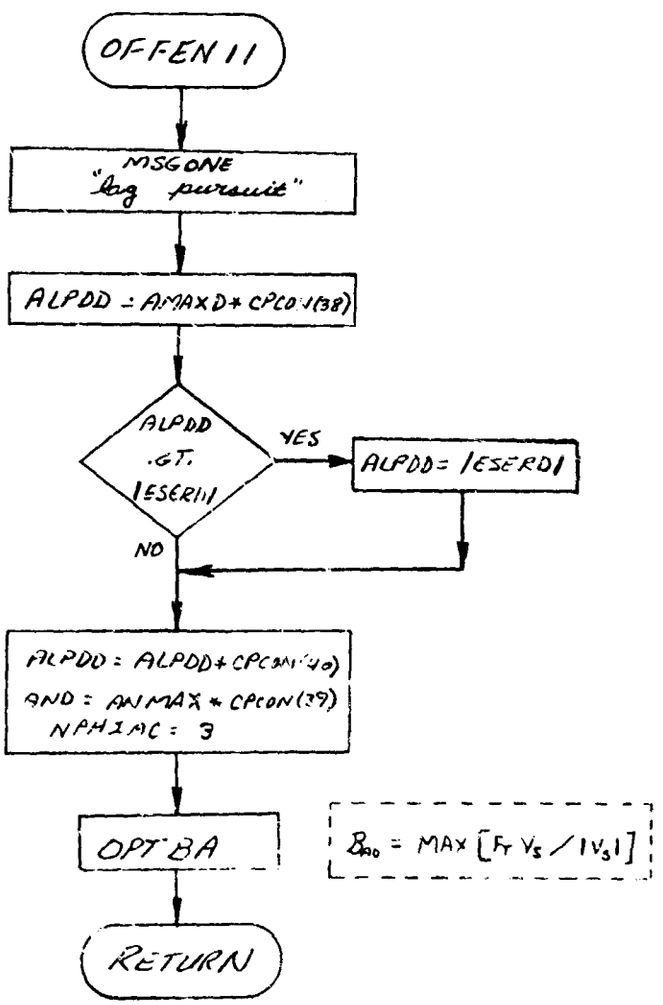
To steer each vehicle along a lag-pursuit course.

Method:

When the first offensive tactic is selected, a vehicle flies a lag-pursuit course determined by force maximization along the lag-pursuit vector. Instantaneous bank-angle perturbation defines the desired control variable values.

Remarks:

A flow chart for OFFEN11 is presented. OFFEN21 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.



91. OFFEN12 and OFFEN22 - Lead-Pursuit Offensive Tactic

Purpose:

To steer each vehicle along a lead-pursuit course.

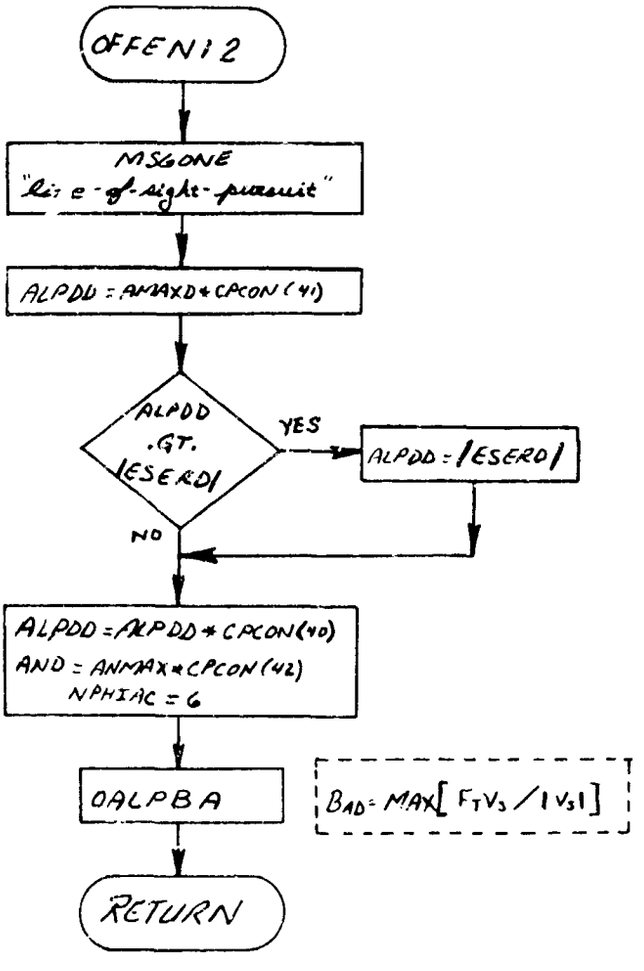
Method:

When the second offensive tactic is selected, a vehicle flies a lead-pursuit course determined by force maximization along the lead-pursuit vector. Instantaneous angle-of-attack and bank-angle perturbations define the desired control variable values.

Remarks:

A flow chart for OFFEN21 is presented. OFFEN22 is identical except for use of vehicle COMMON blocks and auxiliary subroutines

OFFEN12 and OFFEN22 are ENTRY points in OFFEN11 and OFFEN12.



92. OFFEN13 and OFFEN23 - Reference Vector Offensive Tactic

Purpose:

To steer each vehicle along a reference vector course which is a linear combination of the lead-angle and line-of-sight vectors.

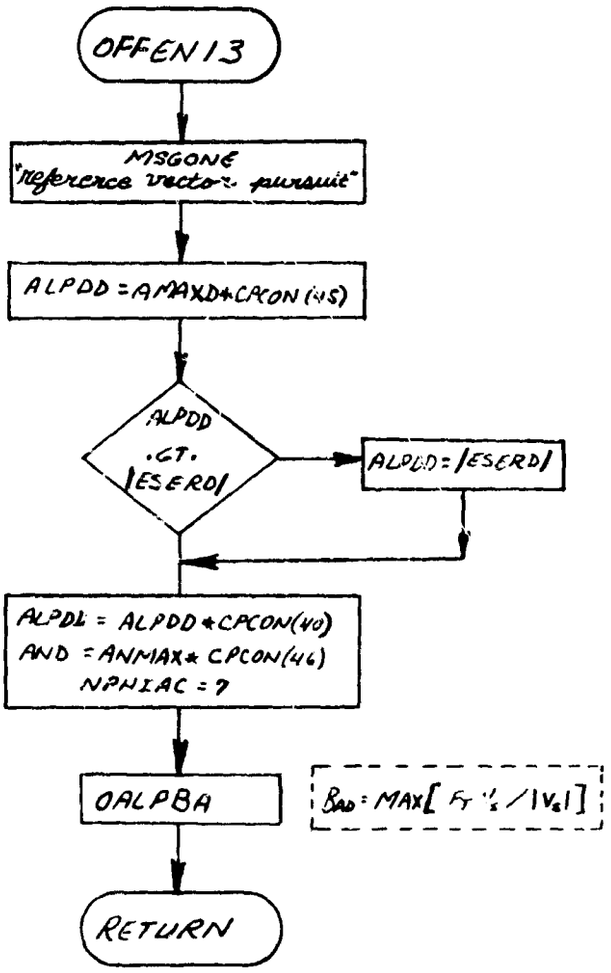
Method:

When the first offensive tactic is selected, a vehicle flies a reference vector pursuit course determined by force maximization along the reference vector. Instantaneous angle-of-attack and bank-angle perturbations define the desired control variable values.

Remarks:

A flow chart for OFFEN13 is presented. OFFEN23 is identical except for use of vehicle 2 COMMON blocks and auxiliary subroutines.

OFFEN13 and OFFEN23 are entry points in OFFEN11 and OFFEN21.



93. ATTAC11 and ATTAC21 - First Attacking Tactic

Purpose:

To steer each vehicle along a path which simultaneously tracks the firing point and eliminates any steering errors.

Method:

When the first attacking tactic is selected, a vehicle steers in a manner calculated to eliminate steering errors and track the firing point. Instantaneous bank-angle perturbations define the desired bank-angle; angle-of-attack is determined by subsequent minimization of the difference between force magnitude and force magnitude required.

Remarks:

A flow chart for ATTAC11 is presented; ATTAC21 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.

94. ATTAC12 and ATTAC22 - Second Attacking Tactic

Purpose:

ATTAC12 and ATTAC22 are ENTRY points in ATTAC11 and ATTAC21, respectively. The second attacking tactic is identical to the second offensive tactic.

ATTAC13 and ATTAC23 - Third Attacking Tactic

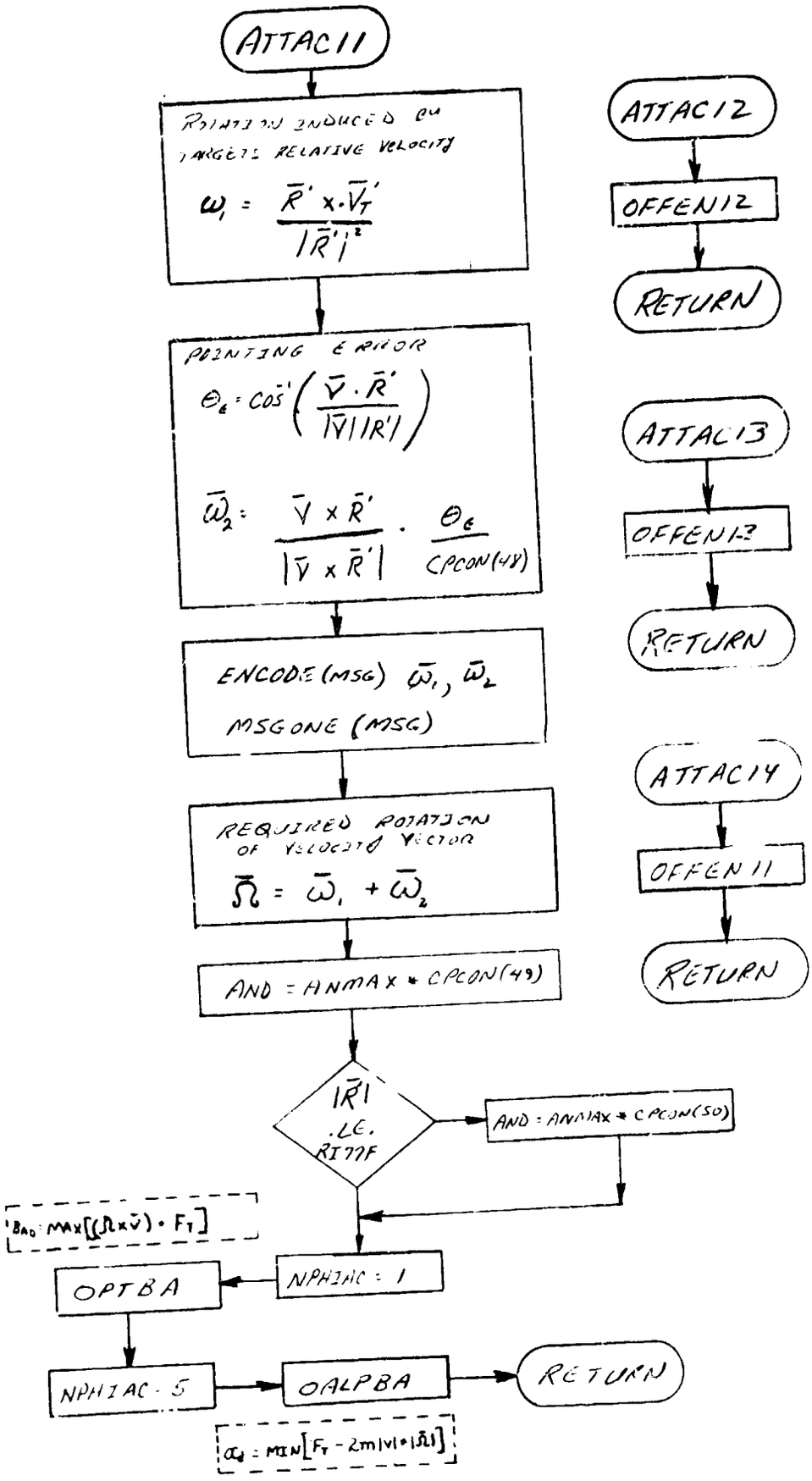
Purpose:

ATTAC13 and ATTAC23 are ENTRY points in ATTAC11 and ATTAC21, respectively. The third attacking maneuver is identical to the third offensive maneuver.

ATTAC14 and ATTAC24 - Fourth Attacking Tactic

Purpose:

ATTAC14 and ATTAC24 are ENTRY points in ATTAC11 and ATTAC21, respectively. The fourth attacking maneuver is identical to the first offensive maneuver.


ATTAC11
ROTATION INDUCED BY TARGET'S RELATIVE VELOCITY
$\omega_1 = \frac{\bar{R}' \times \bar{V}_T'}{|\bar{R}'|^2}$
PRINTING ERROR
$\Theta_E = \cos^{-1}\left(\frac{\bar{V} \cdot \bar{R}'}{|\bar{V}||R'|}\right)$
$\bar{\omega}_2 = \frac{\bar{V} \times \bar{R}'}{|\bar{V} \times \bar{R}'|} \cdot \frac{\Theta_E}{CPCON(47)}$
ENCODE(MSG) $\bar{\omega}_1, \bar{\omega}_2$
MSGONE (MSG)
REQUIRED ROTATION OF VELOCITY VECTOR
$\bar{\Omega} = \bar{\omega}_1 + \bar{\omega}_2$
AND = ANMAX * CPCON(49)
$|\bar{R}|$ .LE. RIIIF
AND = ANMAX * CPCON(50)
B_{AD}: MAX[($\bar{\Omega} \times \bar{V}$) · F_T]
NPHIAC = 1
OPTBA
NPHIAC = 5
OALPBA
RETURN
α_d = MIN[F_T - 2m|V|·|$\bar{\Omega}$|]
ATTAC12
OFFEN12
RETURN
ATTAC13
OFFEN13
RETURN
ATTAC14
OFFEN11
RETURN

95. CTLOPT - Internal AESOP Optimization Loop

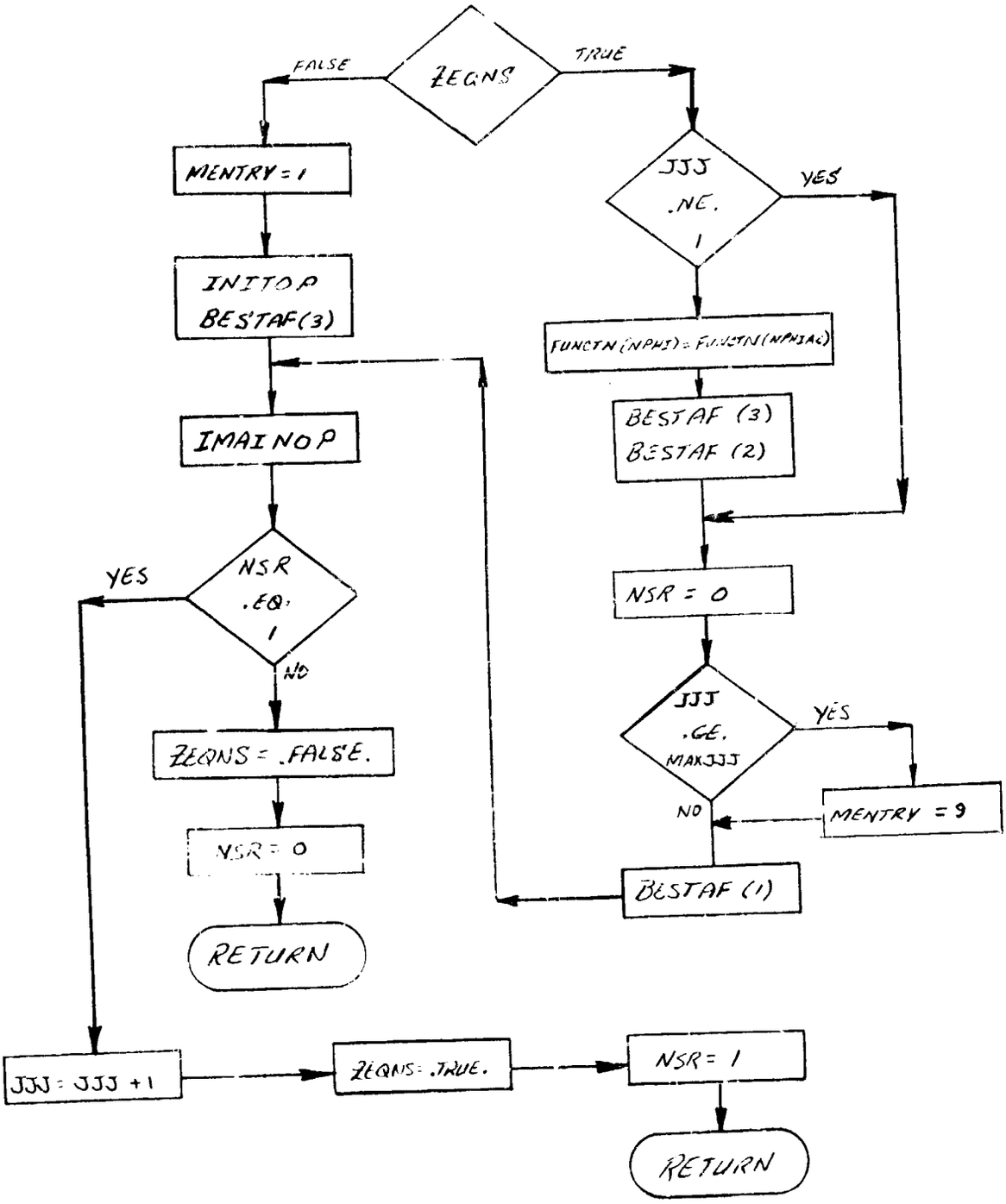
Purpose:

To provide an internal parameter optimization capability independent of the outer trajectory or design optimization loop.

Method:

A control program essentially identical to the AESOP control program is created. The outer parameter optimization data base is preserved on tape at trajectory commencement and is retrieved at trajectory termination. During the trajectory CTLOPT through IMAINOP permits independent access to the AESOP optimization routines. Optimization functions are defined by a variety of control programs embodying combat guidance laws and local inequality information. CTLOPT is used by either vehicle.

CTLOPT


ZEQNS
FALSE
TRUE
MENTRY = 1
INITOP
BESTAF(3)
IMAINOP
NSR .EQ. 1
YES
NO
ZEQNS = .FALSE.
NSR = 0
RETURN
JJJ = JJJ + 1
ZEQNS = .TRUE.
NSR = 1
RETURN
JJJ .NE. 1
YES
FUNCTN(NPHI) = FUNCTN(NPHIAC)
BESTAF (3)
BESTAF (2)
NSR = 0
JJJ .GE. MAXJJJ
YES
NO
MENTRY = 9
BESTAF (1)

96. NDTLU - N - Dimensional Table Look-up Routine

Purpose

To provide a method of linearly interpolating in a table of n independent variables.

Method

Given the arguments X(1), X(2), X(3),..., X(N-1), the routine computes Y = f(X(1)), X(2), X(3),..., X(N-1) by linear interpolation from a table. Extrapolation beyond the upper and lower limits is optional.

Usage

Entry is made via the statement:

CALL NDTLU (ND,NA,X,Z,XA,ZR,IE,NEXTR)

where

ND = Dimension of look-up when Y-f(X). ND = 2
NA = An array of length ND-1. Numbers of values of each table of X. The tables are listed by size, the largest being first.
X = Tables of each X in order.
Z = Function values. If A = f(X,Y,Z) the dependent variable array must be in the following order

Assume X = 4, Y = 3, Z = 2

Z(1) = f(X1,Y1,Z1)	Z(13) = f(X1,Y1,Z2)
Z(2) = f(X2,Y1,Z1)	Z(14) = f(X2,Y1,Z2)
Z(3) = f(X3,Y1,Z1)	Z(15) = f(X3,Y1,Z2)
Z(4) = f(X4,Y1,Z1)	Z(16) = f(X4,Y1,Z2)
Z(5) = f(X1,Y2,Z1)	Z(17) = F(X1,Y2,Z2)
Z(6) = f(X2,Y2,Z1)	Z(18) = f(X2,Y2,Z2)
Z(7) = f(X3,Y2,Z1)	Z(19) = f(X3,Y2,Z2)
Z(8) = f(X4,Y2,Z1)	Z(20) = f(X4,Y2,Z2)
Z(9) = f(X1,Y3,Z1)	Z(21) = f(X1,Y3,Z2)
Z(10) = f(X2,Y3,Z1)	Z(22) = f(X2,Y3,Z2)
Z(11) = f(X3,Y3,Z1)	Z(23) = f(X3,Y3,Z2)
Z(12) = f(X4,Y3,Z1)	Z(24) = f(X4,Y3,Z2)

ZR = Results

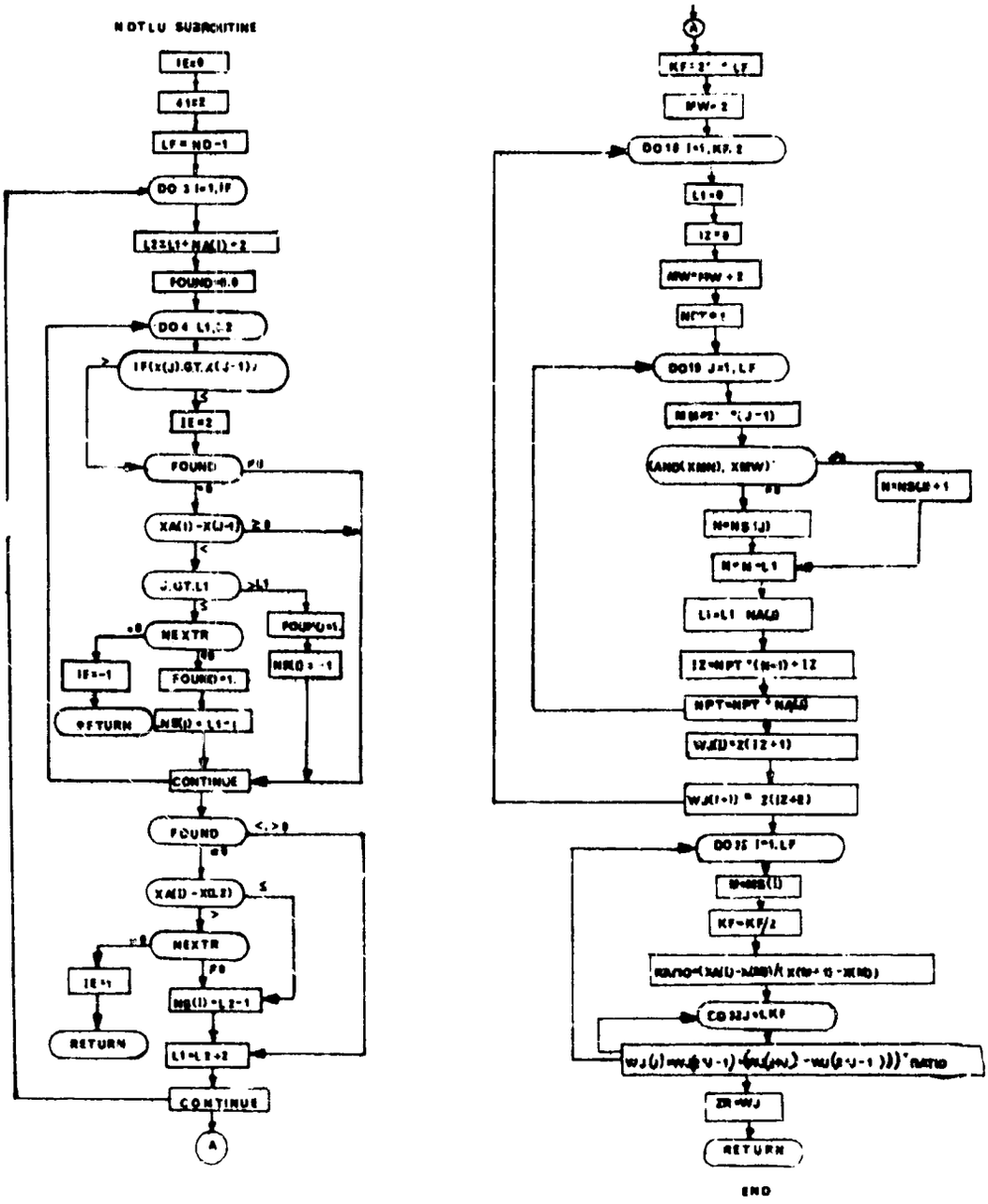
NEXTR = 1 Extrapolate
= 0 No extrapolation
IE = Error code
0 no error
-1 X array too small
1 X array too large
2 array not in ascending order

Let n be number of independent variables, then the table is called an "(n+1) dimensional table."

$$Z = f(x_1,\ldots,x_n)$$

Program

To use a table of dimension ≥ 3 and ≤ 5 a call to HIHO should be made with the list of arguments in the calling sequence in the same order as the independent variables are numbered.


NOTLU SUBROUTINE
IE=0
LF=ND-1
DO 3 I=1,IF
FOUND=0.0
DO 4 L1,L2
IF(X(J).GT.X(J-1))
IE=2
FOUND
XA(I)-X(J-1)
J.GT.L1
NEXTR
IE=-1
RETURN
FOUND=1.
CONTINUE
FOUND
XA(I)-XL(2)
NEXTR
IE=1
RETURN
CONTINUE
A
KF=2** LF
MW=2
DO 18 I=1,KF/2
L1=0
I2=0
MW=MW+2
DO 19 J=1,LF
N=NS(J)
N=N-L1
L1=L1+NA(J)
I2=NPT*(N-1)+I2
NPT=NPT*NA(J)
W(I)=Z(I2+1)
W(I+1)=Z(I2+2)
DO 35 I=1,LF
KF=KF/2
DO 33 J=1,KF
ZR=W(J)
RETURN
END

97. CHEMP and CHEMP2 - Chemical State Computation

Purpose:

To provide the thermodynamic and transport properties of air downstream of the shock.

Method:

Linkage to this subprogram is achieved by a statement of the general form:

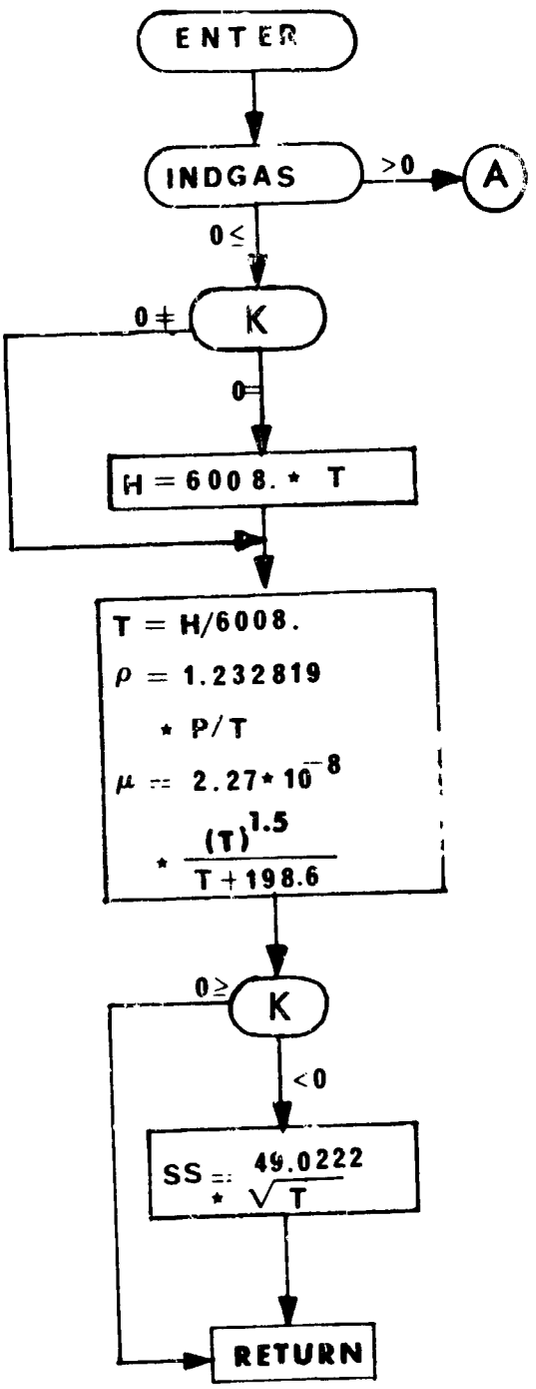
```
CALL CHEMP (P,H,T,RHO,AMU,SS,K)
```

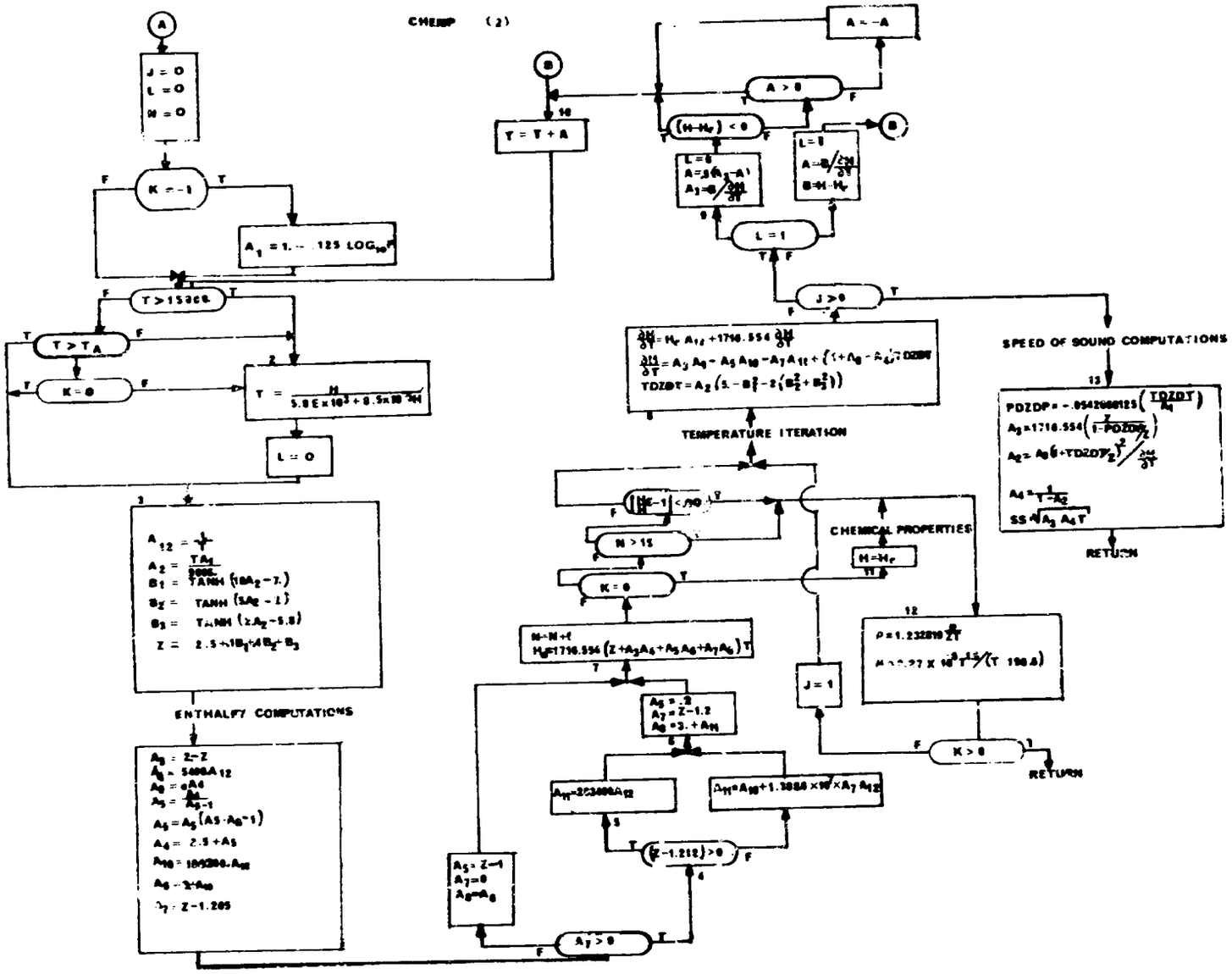
K = 0:	The pressure and temperature are assumed to be defined. The enthalpy, density and coefficient of viscosity are computed.
K = 1:	The pressure and enthalpy are assumed to be defined. The temperature, density and coefficient of viscosity are computed.
K = -1:	The same as the case where K = 1 except that the speed of sound is also computed.

Remarks:

INDGAS = 1:	Real gas effects considered.
INDGAS = -1, 0:	Perfect gas assumed.

CHEMP (1)


ENTER
INDGAS
>0
A
0≤
0≠
K
0=
H = 6008. * T
T = H/6008.
ρ = 1.232819
* P/T
μ = 2.27 * 10⁻⁸
* (T)^1.5 / T + 198.6
0≥
K
<0
SS = 49.0222 / √T
RETURN


CHEMP (2)
J = 0
L = 0
N = 0
K = -1
T > T_A
K = 0
L = 0
ENTHALPY COMPUTATIONS
T = T + A
A = -A
A > 0
L = 1
J > 0
TEMPERATURE ITERATION
SPEED OF SOUND COMPUTATIONS
RETURN
CHEMICAL PROPERTIES
N > 15
K = 0
J = 1
K > 0
RETURN
A_7 > 0

93. CONV - Non-Linear Equation Solver

Purpose

To find a zero of the equation $g(x) = 0$ where $g'(x)$ exists in the vicinity of the zero

Method

The method of false position is used with the Aitken δ^2 process to improve the rate of convergence.

Let $F(r,s) = \dfrac{rg(s) - sg(r)}{g(s) - g(r)}$ (False position)

$D(r,s,t) = t - \dfrac{(t - s)^2}{t-2s + r}$ (δ^2 - process)

Given an estimate x_1, of the zero, the following steps are executed:

1. $x_2 = (1.0001)x_1$ or $x_2 = x_1 + .0001$, whichever is larger
2. $x_3 = F(x_1, x_2)$
3. Let x_4 be the x_1 or x_2 which produces a g whose sign is opposite that of $g(x_3)$. If there is no such x_4, let x_4 be the x_1 or x_2 producing the smallest value of g. Let $x_5 = F(x_3, x_4)$
4. Let $z = x_1$ or x_2 which was not set equal to x_4. $x_6 = x_5 \dfrac{(x_5 - x_3)^2}{x_5 - 2x_3 + z} = D(z_1,x_3,x_5)$
5. Let $x_2 = x_6$ and x_1 be the z_1, x_3, or x_5 producing a g whose sign is opposite that of $g(x_6)$. If no such x exists, let $x_1 = x_5$. Return to Step 2.

This procedure is repeated until $\left|g(x)\right| < \epsilon$, ϵ a given tolerance.

Usage

Given an x_n and $g(x_n)$, the routine computes a new value x_{n+1}.

Entry CALL CONV (EPS, MAX, GXN, XN, IND, K)

An initial call is made with K = 0. This is done only once in the program, prior to the second entries first call, and initializes EPS, MAX, and indicators in the routine. EPS is the convergence tolerance, ϵ. MAX is the maximum number of interations allowed before an error return is made.

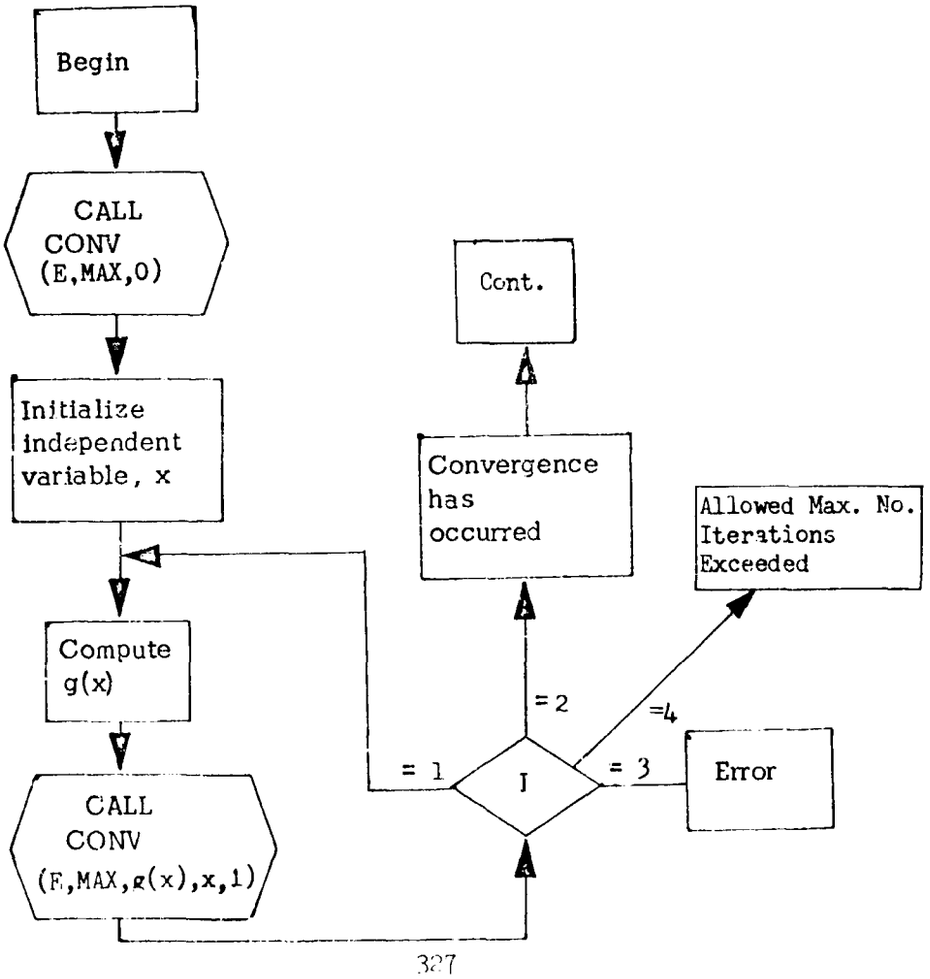
After every calculation of $g(x_n)$, a call is made with K = 1, and the current values of $g(x_n)$ and x_n. Return is made with a new estimate x_{n+1}, and a return indicator IND, where

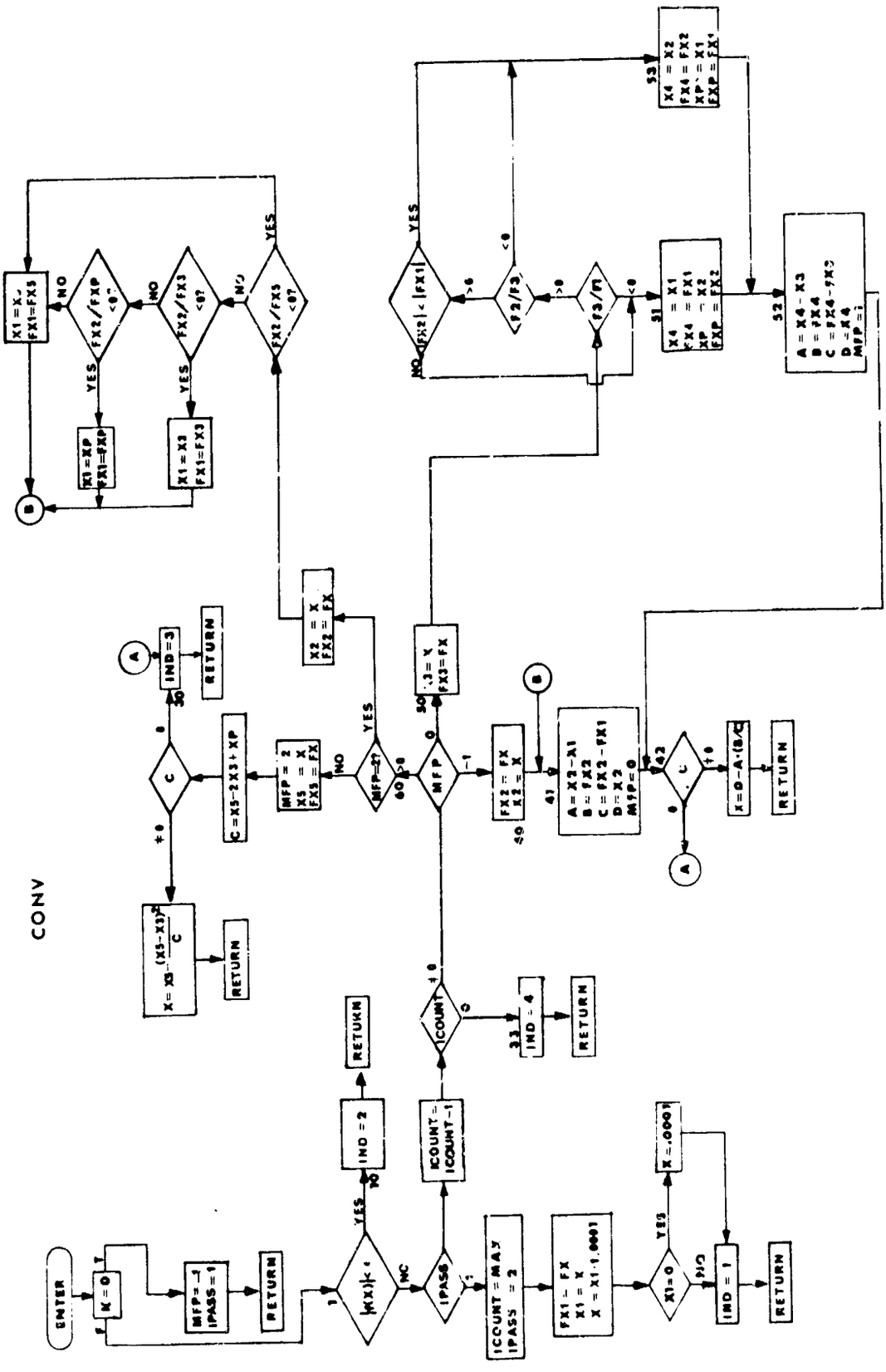
$$IND = \begin{cases} 1, & \text{Calculate } g(x) \text{ again} \\ 2, & \text{Successful convergence (i.e., } |g(x)| < \epsilon) \\ 3, & \text{Error return, division by zero was imminent, returns with old value of } x_n. \\ 4, & \text{Allowed maximum number of iterations exceeded.} \end{cases}$$

GXN = value of $g(x_n)$

XN = current estimate of x. This is the value x_n corresponding to $g(x_n)$ when the call is made, and is either a new estimate x_{n+1} on return from CONV with IND = 1, or is the old value x_n on return with IND = 2,3,4.

The user first gives the call statement with K = 0 for initialization. Then he establishes an initial guess, x_n, evaluates $g(x_n)$, and gives the second call, with K = 1. If convergence is not obtained, the routine returns with IND = 2 and a new x-guess in XN. The user must now re-evaluate GXN using the new x-guess. The process continues until IND = 2, 3, or 4. For IND = 4, no convergence in MAX tries. For IND = 3 a division by zero has occurred. For IND = 2, the correct x-value is in XN. The following flow diagram is intended to clarify the use of CONVRG.




CONV
ENTER
X = 0
MFP = -1
IPASS = 1
RETURN
|FX| < E
YES
NO
IND = 2
RETURN
ICOUNT = ICOUNT - 1
IPASS
ICOUNT = MAX
IPASS = 2
FX1 = FX
X1 = X
X = X1 + .0001
X1 = 0
X = .0001
IND = 1
RETURN
ICOUNT
IND = 4
RETURN
MFP
FX2 = FX
X2 = X
A = X2 - X1
B = FX2
C = FX2 - FX1
D = X2
MFP = 0
C
X = 0 - A/(2C)
RETURN
X = X3 - (X3 - X1)^2 / C
RETURN
C = X3 - 2X3 + XP
MFP = 2
X3 = X
FX3 = FX
MFP = 2?
X2 = X
FX2 = FX
X3 = X
FX3 = FX
IND = 3
RETURN
X1 = XP
FX1 = FXP
X1 = X3
FX1 = FX3
X1 = X
FX1 = FX3
FX2/FXP < 0 ?
FX2/FX3 < 0 ?
FX3/FX1 < 0 ?
|FX2| < |FX1|
F2/F3
F3/F1
X4 = X1
FX4 = FX1
XP = X2
FXP = FX2
X4 = X2
FX4 = FX2
XP = X1
FXP = FX1
A = X4 - X3
B = FX4
C = FX4 - FX3
D = X4
MFP = 1

99. TFFM and TFFM2 - Multiengine Thrust and Fuel Flow

Purpose:

The multiengine thrust and fuel flow program provides the means of introducing the data for the various engines. It corrects the thrust for atmospheric effects when appropriate and resolves the thrust vectors into their components.

Usage:

Linkage to TFFM is accomplished via the general statement:

CALL TFFM (IENTRY)

where IENTRY is a fixed point variable.

IENTRY = 1

This is the pre-data initialization. At this entry the subscripts for all tables are computed and the following data is initialized:

INDTFF = 0	= thrust option indicator
$N_i = 1.$	= throttle setting
$CNVM_i = 1.$	= mass rate units conversion factor
$CNVT_i = 1.$	= thrust units conversion factor
$INDENG_i = 0$	= engine option indicator
$\epsilon_{T1_i} = 1.$	= propellant mass flow rate factor
$\epsilon_{T2_i} = 1.$	= specific impulse factor
$\epsilon_{T3_i} = 1.$	= action time factor
$\epsilon_{T4_i} = 1.$	= propellant loading factor
$\epsilon_{T5_i} = 1.$	= perturbation factor
$\epsilon_{T6_i} = 1.$	= perturbation factor
$\phi_i = 0$	= nozzle rotation about x-axis
$\lambda_i = 0$	= nozzle swivel angle from x-axis
$P_{R_i} = 0$	= thrust reference pressure

where i as used above is the set of integers 1, 2, 3 corresponding to the three engines.

IENTRY = 2

This is the post-data initialization. At this entry no computations are performed if INDTFF = 0. If INDTFF $\neq$ 0 the number of engines is determined from the indicator INDENG. If no INDENG has been loaded, an error exit occurs.

IENTRY = 3

At this entry the various computations are performed. If INDTFF = 0, no computations are performed. When INDTFF is 10 or 11 and INDENG is defined, for a given engine, the following computations are performed for that engine:

INDTFF = 10 when INDENG = 11 gives the Multiengine, Noncontrolled, Perturbed Thrust Option. For this option the tabulated data functional relationships are:

$$T_R = f(\tau)$$

$$\dot{A}_p = f(\tau)$$

and the computations performed for the engine are as follows:

$\tau = \frac{\tau^s \epsilon_{T1}}{\epsilon_{T2} \epsilon_{T3}}$ = reference time as a function of stage time

$\dot{m}_{ENG} = \frac{T_R(\tau)}{G_{WM} I_{SP_{R_{ENG}}}} \cdot \frac{\epsilon_{T1} \epsilon_{T4}}{\epsilon_{T2}}$ = fuel flow rate + internal inert product flow rate

$T = \epsilon_{T1} \epsilon_{T4} T_R(\tau) - (P - P_R)A_e$ = total corrected thrust

$\dot{m}_t = -(\dot{m}_{ENG} + \dot{A}_p)$ = total mass flow rate

INDTFF = 10 when INDENG = 12 gives the Multiengine, Controlled, Perturbed Thrust Option. For this option the tabulated data functional relationships are:

$$\bar{\dot{m}}_f = f(N)$$

$$\dot{A}_p = f(\tau^s)$$

and the computations performed for the engine are as follows:

$\dot{m}_f = \bar{\dot{m}}_f \epsilon_{T5}$ = perturbed mass flow rate

$T' = G_{WM} I_{SP_{R_{ENG}}} \dot{m}_f$ = perturbed thrust

$T = \epsilon_{T1} \epsilon_{T4} T' - (P - P_R)A_e$ = total corrected thrust

$\dot{m}_t = -(\dot{m}_f + \dot{A}_p)$ = total mass flow rate

INDTFF = 10 when INDENG = 13 gives the Multiengine, Air Breather, Perturbed Thrust Option. For this option the tabulated data functional relationships are:

$$T_R = f(N,h, \alpha, M_N)$$

$$\dot{\bar{m}}_t = f(N,h, \alpha, M_N)$$

and the computations performed for the engine are as follows:

$\dot{m}'_t = \dot{\bar{m}}_t \epsilon_{T6}$ = perturbed mass flow rate

$T = T_R \epsilon_{T6}$ = perturbed thrust

$\dot{m}_t = -\dot{m}'_t$ = vehicle mass flow rate

INDTFF = 11 when INDENG = 11 gives the Multiengine, Noncontrolled, Perturbed Thrust Having Alternate Table Format. For this option the tabulated data functional relationships are:

$$\dot{\bar{m}}_f = f(\tau^3)$$

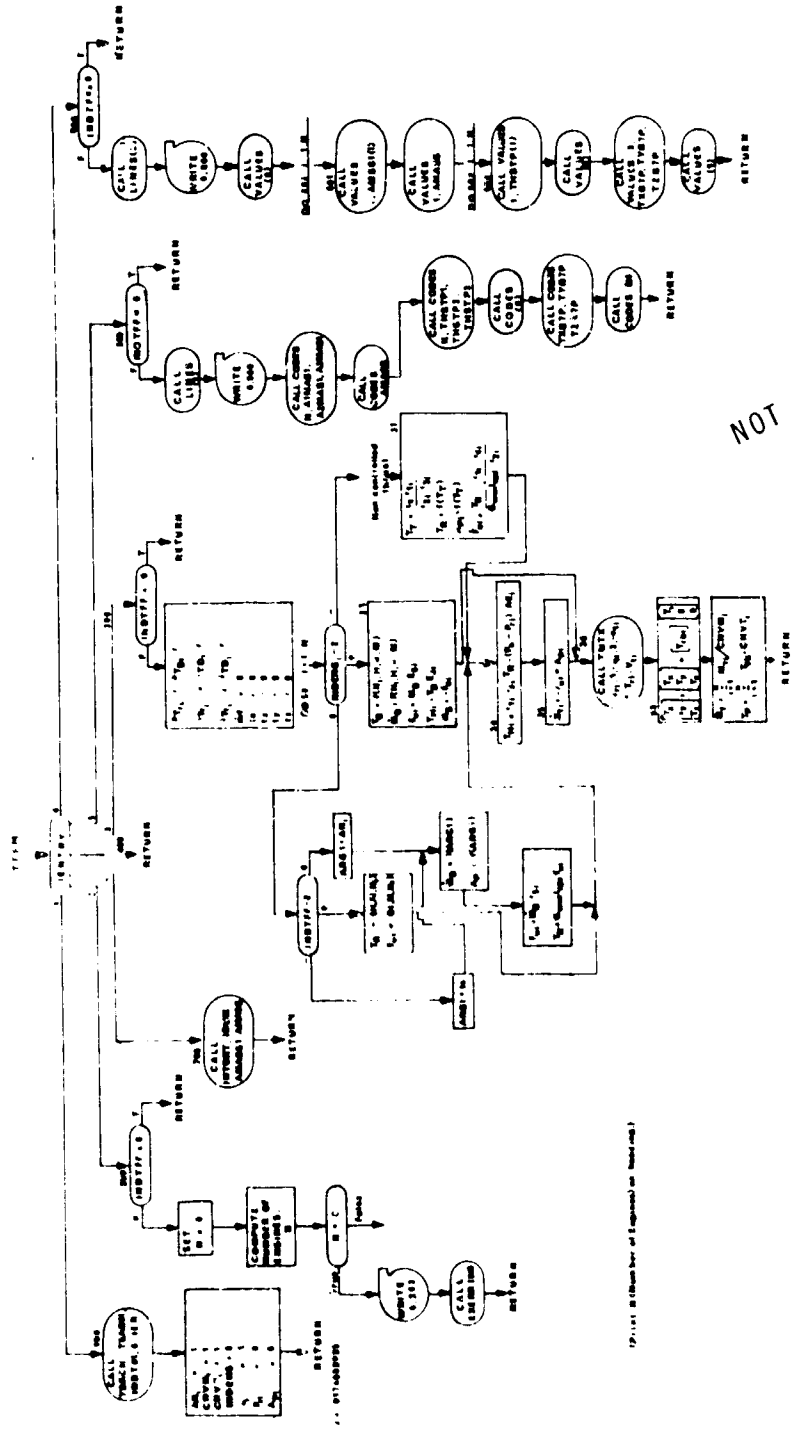
$$\dot{A}_p = f(\tau^3)$$

The computations are identical to the computations in the Multiengine, Controlled, Perturbed Thrust Option, i.e. INDTFF = 10 when INDENG = 12.

NOTES:

While it is possible to change from the single engine option (TFFS) in a given stage to the multiengine option (TFFM) in a later stage, the opposite is not possible.

A flow chart for TFFM is presented. TFFM2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.



NOT REPRODUCIBLE

100. ATMS59 and ATMS592 - 1959 Atmosphere Calculation Routine

Purpose:

To compute the atmosphere characteristics: density, speed of sound, pressure, temperature and kinematic viscosity. All are a function of altitude.

Method:

All atmosphere characteristics are computed using the 1959 ARDC model atmosphere. Values of the atmosphere are computed for positive altitudes. If this altitude is negative, the sea level values will be obtained.

Usage:

Linkage is affected by

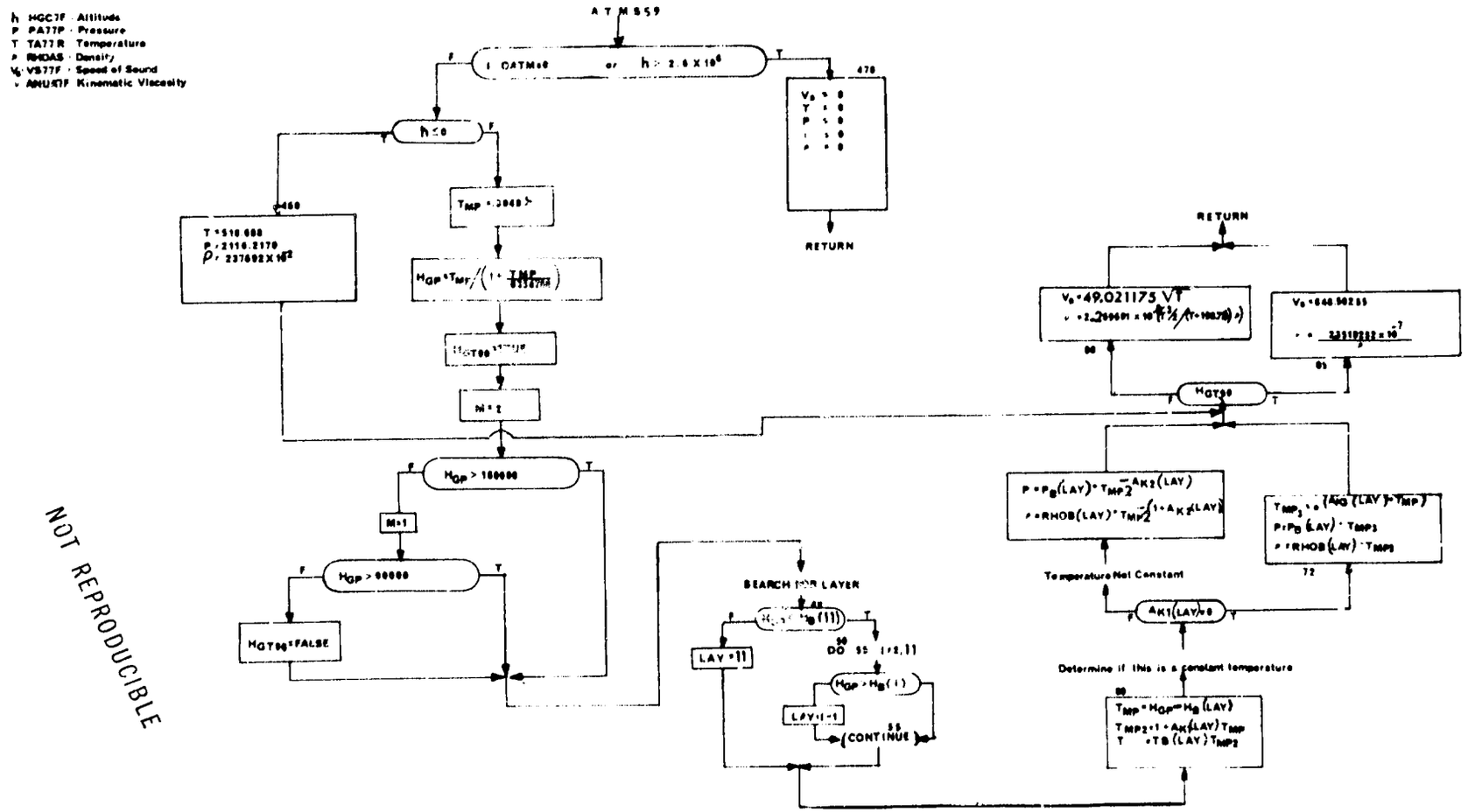
```
CALL ATMS59 (HGC7F)
```

where

HGC7F = altitude in feet.

Remarks:

ATMS59 and ATMS592 are identical except for the use of differing vehicle COMMON blocks.



NOT REPRODUCIBLE

101. ATMS62 and ATMS622 - 1962 Atmosphere Calculation Routine

Purpose:

To compute the atmosphere characteristics: density, speed of sound, pressure, temperature and kinematic viscosity. All are a function of altitude.

Method:

All atmosphere characteristics are computed using the 1962 ARDC model atmosphere. Values of the atmosphere are computed for positive altitudes. If this altitude is negative, the sea level values will be obtained.

Usage:

Linkage is affected by

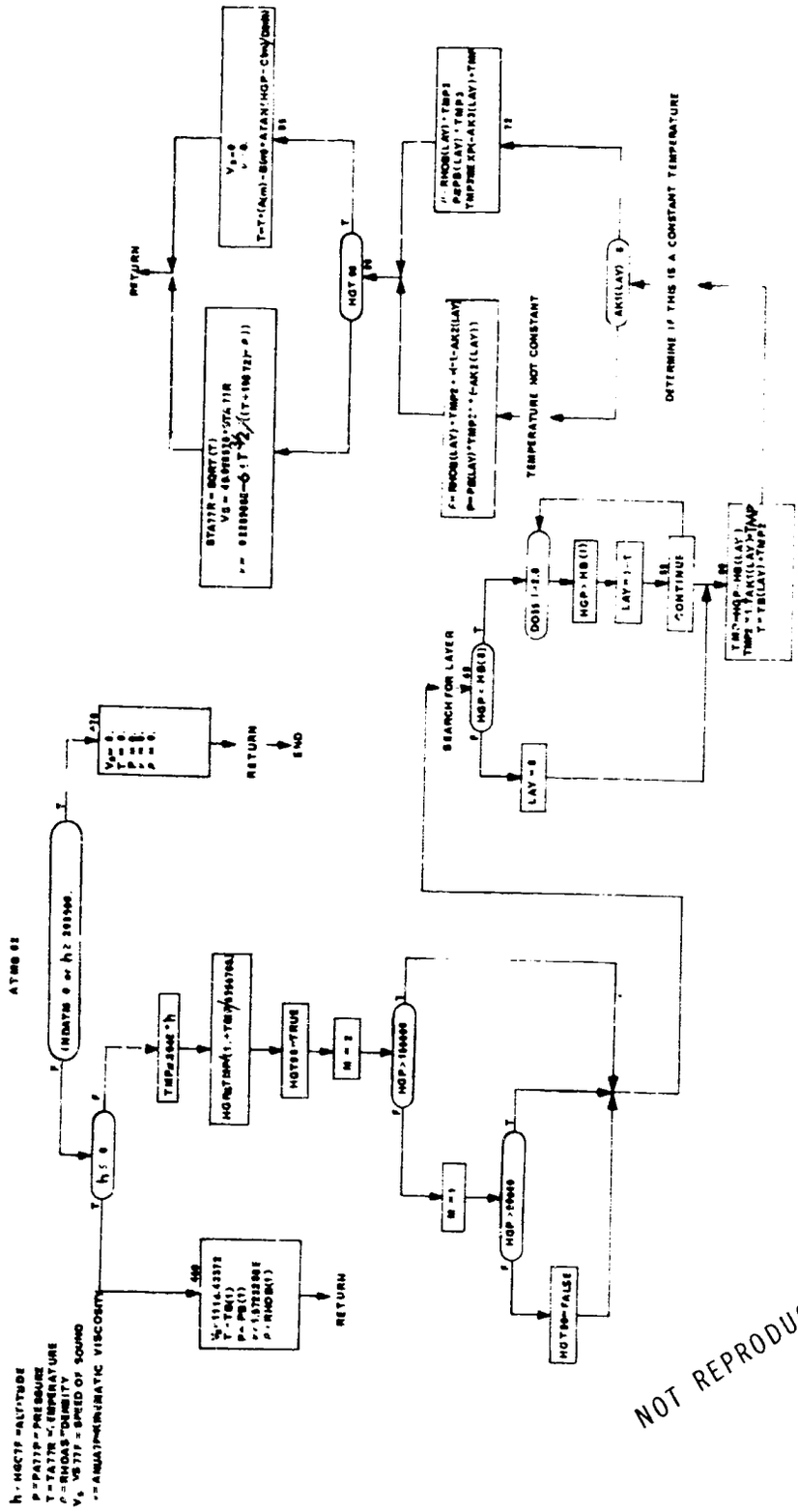
```
CALL ATMS62 (HGC7F)
```

where,

HGC7F = altitude in feet.

Remarks:

ATMS622 is identical to ATMS62 except for the use of vehicle 2 COMMON blocks.



NOT REPRODUCIBLE

102. PUT - Character Manipulation Routine

Purpose:

To replace the Ith character of a string of characters with the first character of another string of characters.

Usage:

Entry is made to this routine by the following statement:

CALL PUT(P, I, L)

where the Ith character of string P will be replaced by the first character of string L.

103. GET - Character Manipulation Routine

Purpose:

To replace the first character of a word with the Ith character of a string of characters.

Usage:

Entry is made to the routine by the following statement:

CALL GET(P, I, L)

where the first character of the word L will be replaced by the Ith character of string P. The remaining nine characters of word L are replaced by blanks.

104. OPTBA and OPTBA2 - Bank-Angle Iteration Routine

Purpose:

To define instantaneous bank-angle on the basis of local optimization criteria.

Method:

A local minimization criteria, (t), is created where

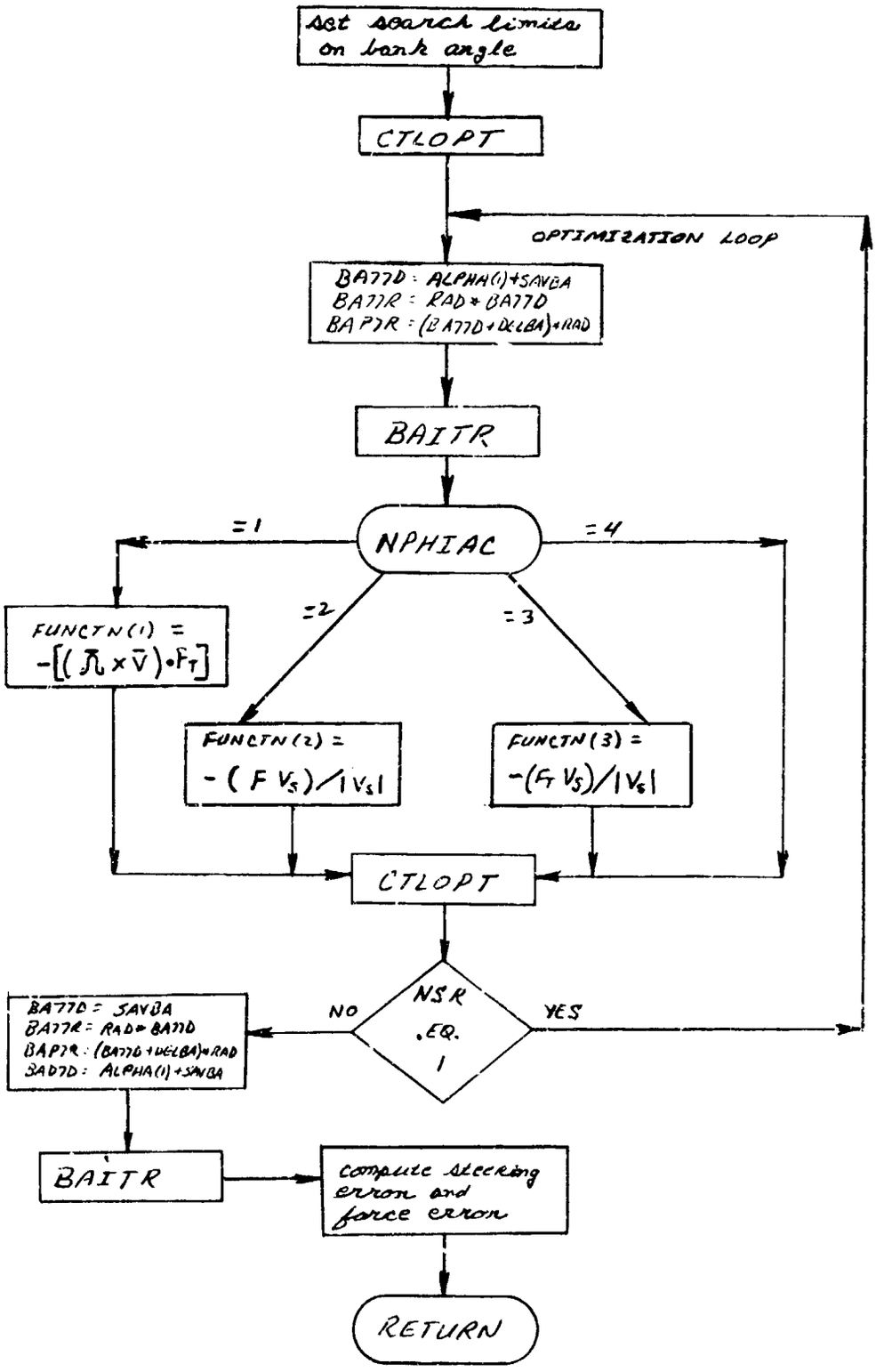
$$\phi(t) = \phi(B_A)$$

An inner loop parameter optimization procedure, CTLOPT, is used to define the bank-angle value which satisfies the resulting local (t = constant) optimization problem. The local minimization criteria is selected from one of several combat guidance laws. Final steering errors and force vector pointing errors are also computed.

Remarks:

A flow chart for OPTBA is presented. OPTBA2 is identical except for the use of vehicle 2 COMMON blocks and auxiliary subroutines.

OPTBA


Set search limits on bank angle
CTLOPT
OPTIMIZATION LOOP
BA77D = ALPHA(1)+SAVBA
BA77R = RAD * BA77D
BAPJR = (BA77D + DELBA) * RAD
BAITR
NPHIAC
=1
=2
=3
=4
FUNCTN(1) = -[(Ω x V̄)·F_T]
FUNCTN(2) = -(F V_S)/|V_S|
FUNCTN(3) = -(F_T V_S)/|V_S|
CTLOPT
NSR .EQ. 1
NO
YES
BA77D = SAVBA
BA77R = RAD * BA77D
BAPJR = (BA77D + DELBA) * RAD
SAD7D = ALPHA(1) + SAVBA
BAITR
compute steering error and force error
RETURN

105. IMAINOP - Internal Parameter Optimization Interface Routine

Purpose:

To act as an interface between the two vehicle trajectory equations and the parameter optimization routines.

Method:

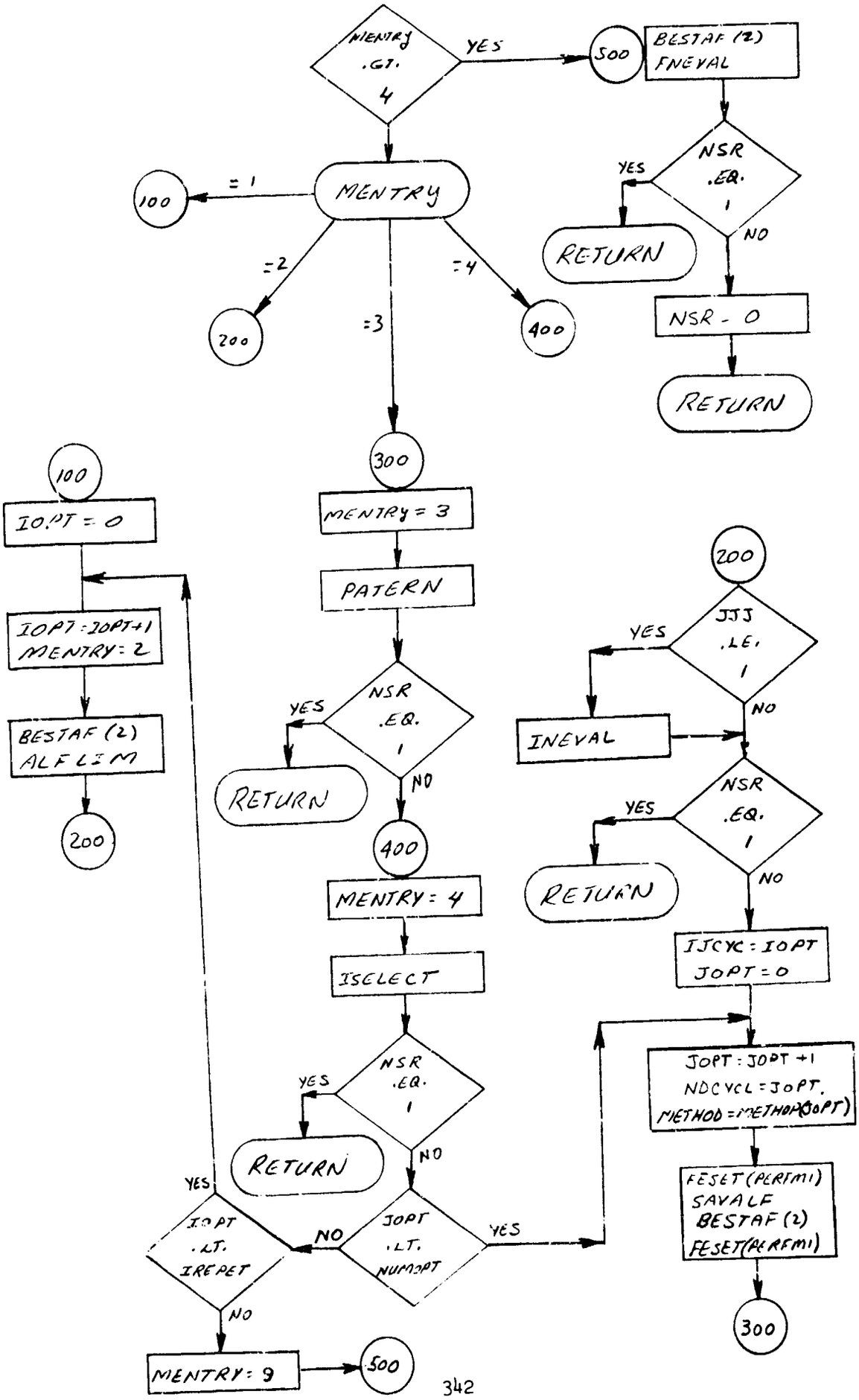
Core size requirements limit the internal optimization capability to

a. Sectioning search

b. Creeping search

c. Pattern search

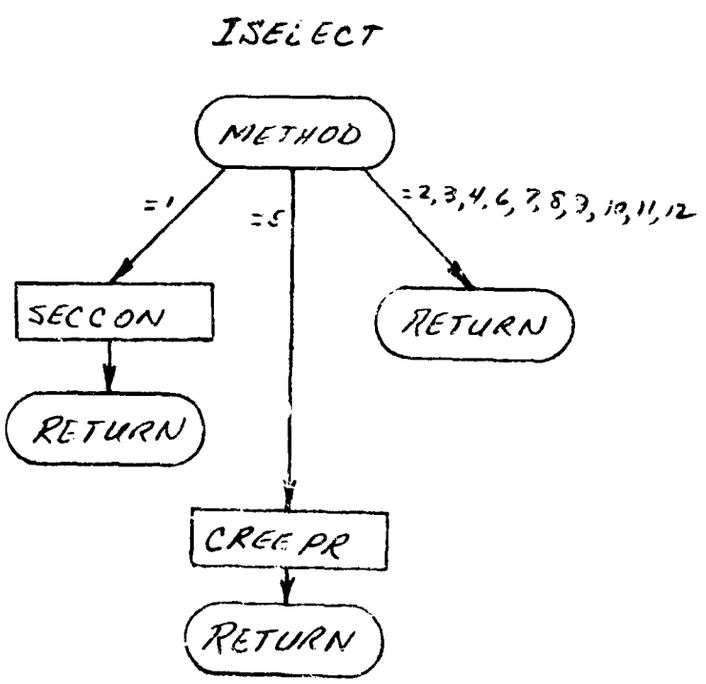
IMAINOP interfaces the trajectory equations to these three search options.


MENTRY .GT. 4
YES
500
BESTAF (2) FNEVAL
NSR .EQ. 1
YES
RETURN
NO
NSR = 0
RETURN
MENTRY
= 1
100
=2
200
=3
=4
400
300
MENTRY = 3
PATERN
NSR .EQ. 1
YES
RETURN
NO
400
MENTRY = 4
ISELECT
NSR .EQ. 1
YES
RETURN
NO
JOPT .LT. NUMOPT
YES
NO
IOPT .LT. IREPET
YES
NO
MENTRY = 9
500
100
IOPT = 0
IOPT = IOPT+1
MENTRY = 2
BESTAF (2) ALFLIM
200
200
JJJ .LE. 1
YES
NO
INEVAL
NSR .EQ. 1
YES
RETURN
NO
IJCYC = IOPT
JOPT = 0
JOPT = JOPT +1
NDCYCL = JOPT.
METHOD = METHOD(JOPT)
FESET (PERTMI)
SAVALF
BESTAF (2)
FESET (ALRFMI)
300

106. ISELECT - Interval Search Selection Routine

Purpose:

To select either sectioning or creeping search in the internal parameter optimization loop.



107. MULT33 - A Matrix Multiplication Routine

Purpose:

To post multiply a 3 x 3 matrix by a 3 x 3 matrix.

Method:

The result of A B = C is computed using single precision

Usage:

The matrix multiplication is obtained by the statement:

CALL MULT33 (A, B, C)

where

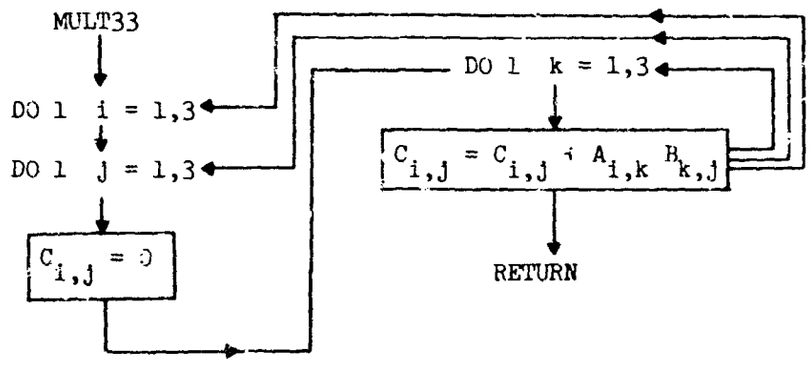
A = array name of the 3 x 3 matrix A

B = array name of the 3 x 3 matrix B

C = array name where results are to be stored as the 3 x 3 matrix

Remarks:

This routine calls no other routines.



SECTION VII
PROGRAM CTLS

Program CTLS controls the variational optimization option. The tasks performed by the CTLS program include the following:

1. Print the CTLS summary, which includes the values of the optimization functions. (E)

2. Evaluate the forward trajectory by considering the improvement or deterioration of the optimization functions. (E)

3. Decide whether to accept a forward trajectory as a valid step, or to reject it. (E)

4. To determine the α-perturbation to use on the next forward trajectory.

5. To put together the α-perturbation that is desired by using the information from REV on tape ILTAP, constructing the CTABLE and putting it out on tape IATAP.

6. To output the restart cards for trajectories which are accepted as valid steps. (E)

7. Housekeeping - e.g. updating the pass number and cycle number, PHBRST, PSBRST, etc.

The CTLS segment is entered either from EXE or REV. The "E" designates those tasks which pertain only to entry to CTLS from EXE.

CTLS proper allows the user to select which control system will be used in the optimization program.

Usage

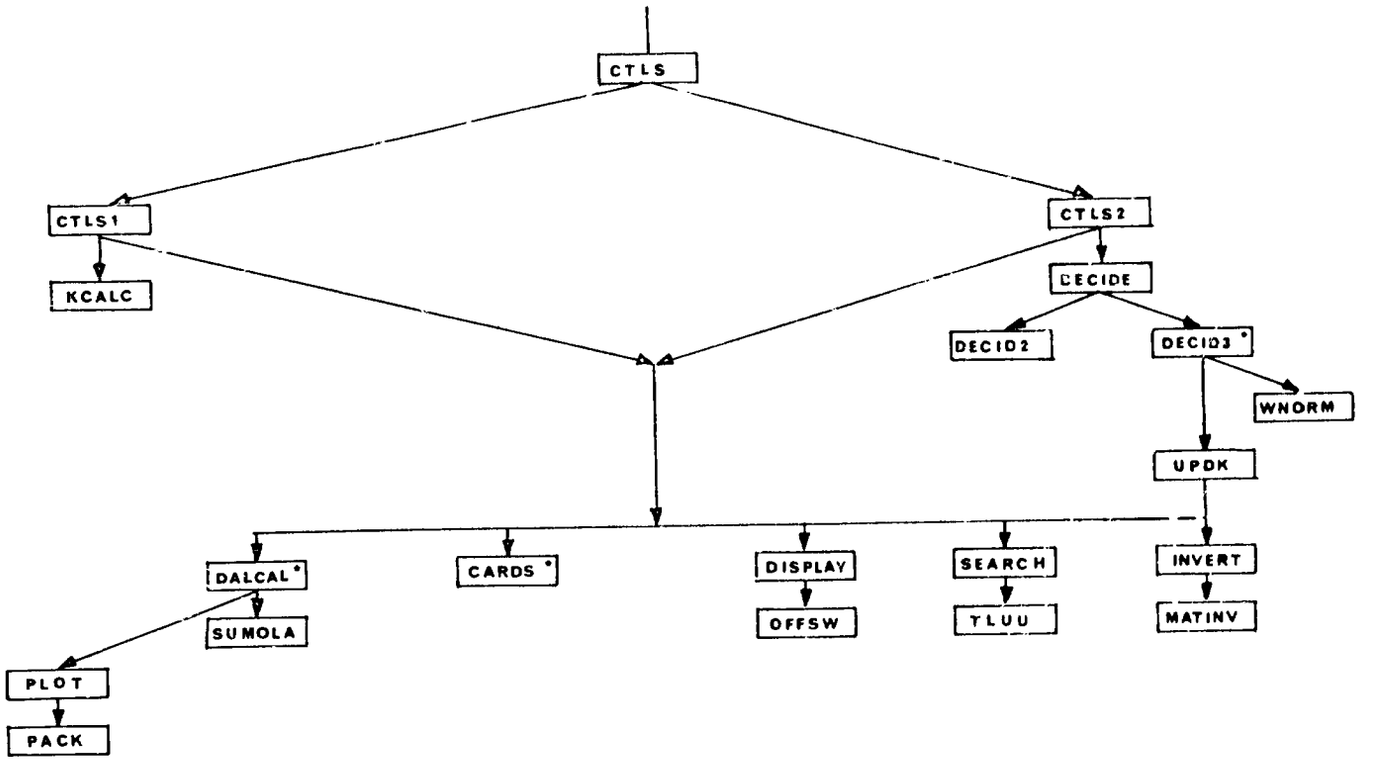
CALL CHAIN (CTLS)

Remarks

INDSEL is of the form ij. i determines which control system will be used with the optimization program. For i = 1, CTLS1 is used. For i = 3, CTLS2 is used, i also determines which decision subprogram will be called. Currently i = 1 and i = 3 are the only usable options. j determines which equations subprogram will be used.

CTLS calls CTLS1 and CTLS2.

Organizational Chart for CTLS



*DESIGNATES MULTIPLE ENTRY POINTS

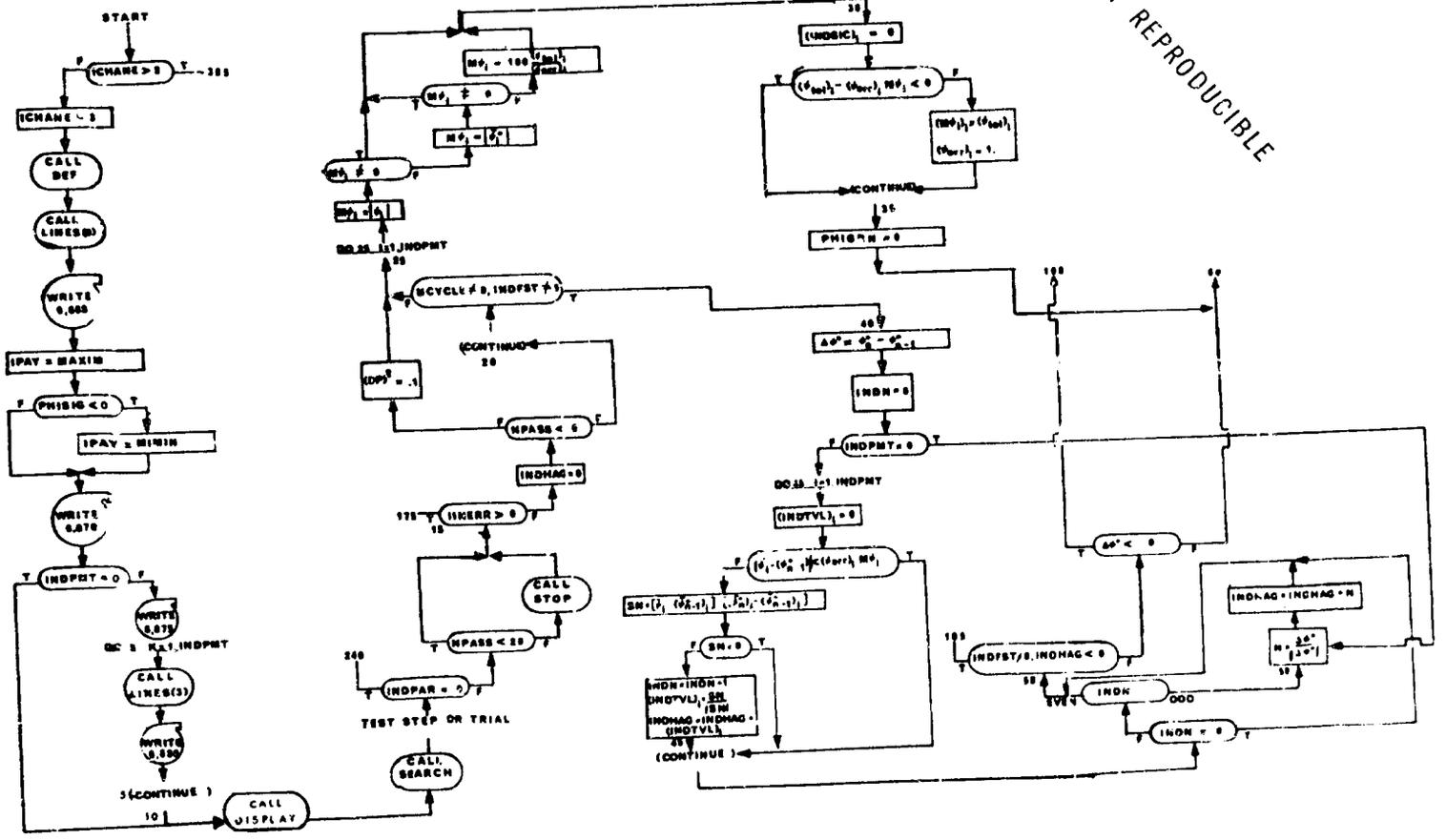
1. CTLS1 - Original Control System

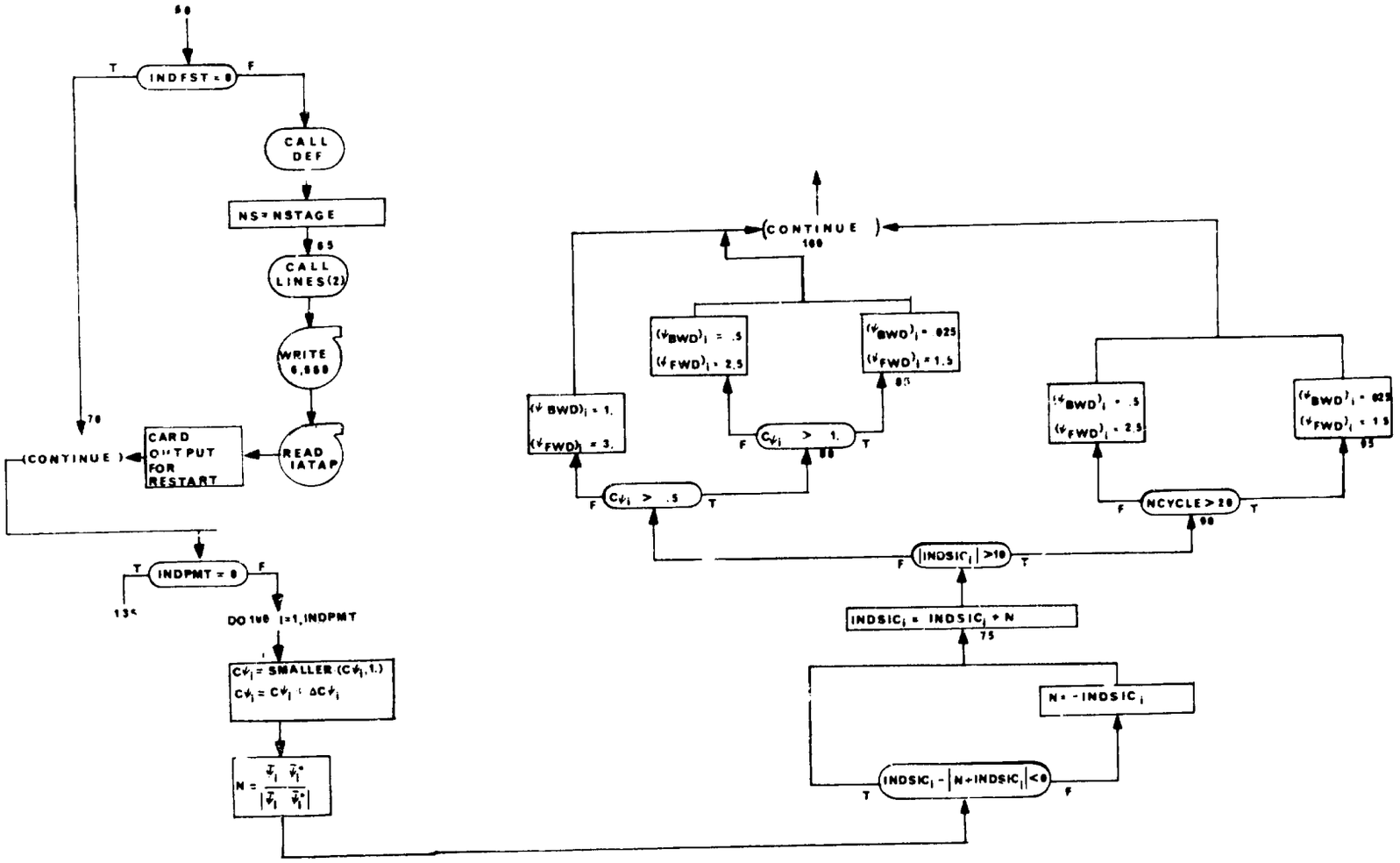
The control system is a predictor-corrector method of control over the convergence of the end constraints and payoff function. A linear prediction of the changes in end constraints and payoff function, due to a change in the control variable(s), is made and attempted. If the trial is near enough to the linear prediction, the control variable is changed to give the best change in end constraints and payoff function, and the final values of the functions (for the cycle) are computed. If the trial is not near enough, the change in the control variable(s) is reduced and another prediction and attempt are made. If the end points and payoff function are too near the linear prediction, the amount of change is increased. Under normal circumstances, the control will never take more than six passes but if for some reason the INDERR is set positive more passes can be taken. If six or more passes are made by the control system then INDSIC is set to 0 and MY_i and Y_{TOL_i} are reset. On return from the reverse integeration if $DP^2 - d\beta_i \, I_{YY}^{-1} \, d\beta_i$ is negative then $DP^2 = 1.1 \, DP^2_{n-1}$ and only the CY_i corresponding to the constraints which are diverging are decreased.

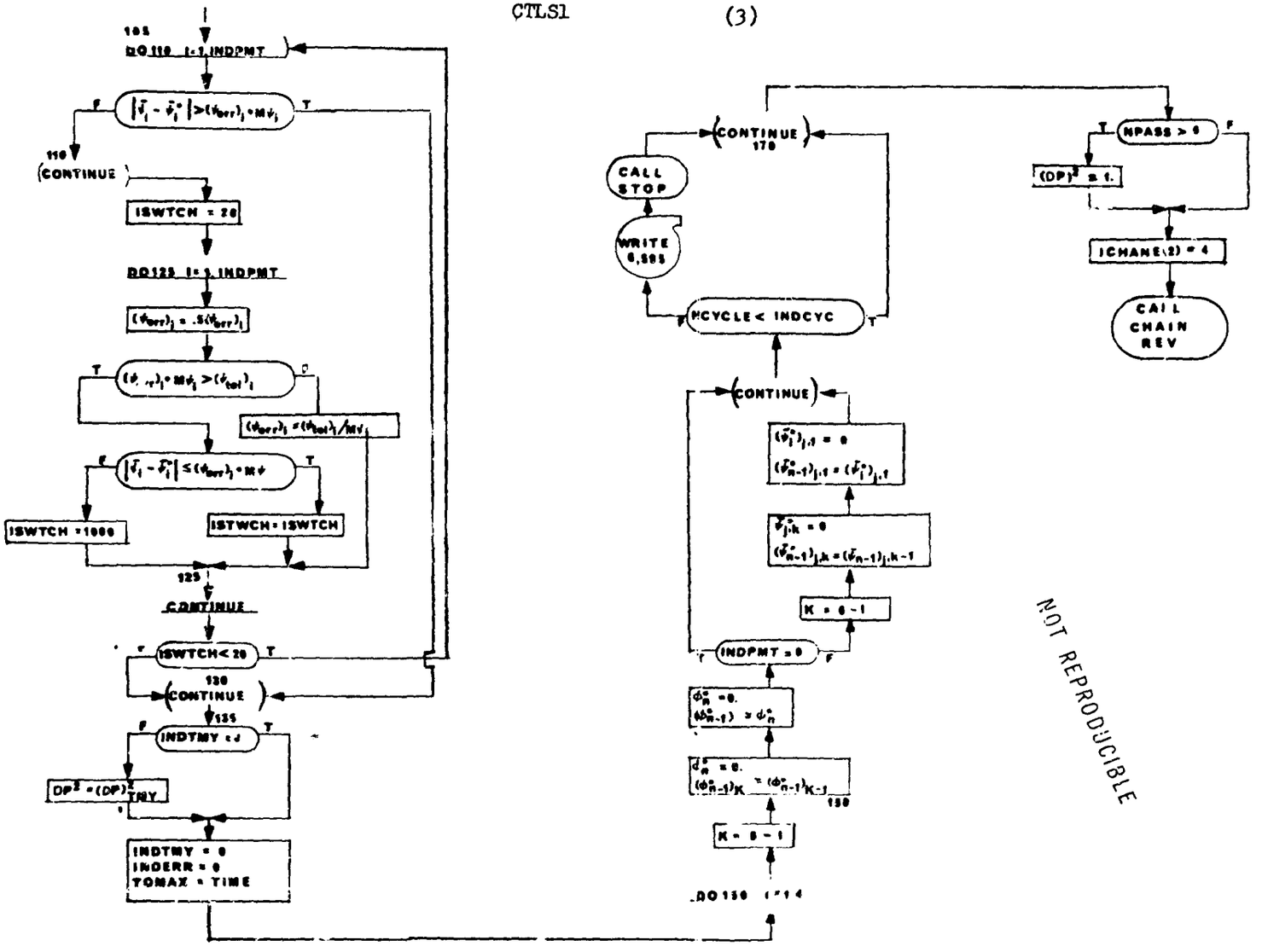
There are two entries and two exits in this program. One entry is from the program which computes the functions and matrices of partial derivatives (DIFEQ); this entry is to the portion of the control system which compares the linear predictions to the actual values of the end point and payoff function changes. An exit is made, after each cycle is completed, to a program which computes the [I] matrices and other values needed to compute the mode shape of the change in the control variable(s) (REV).

Another entry is made from that program (REV) to the portion which checks for the conditions needed to terminate and computes the control variable change for the first trial of a new cycle. The second exit is to the program which computes the functions and matrices of partial derivatives (DIFEQ).

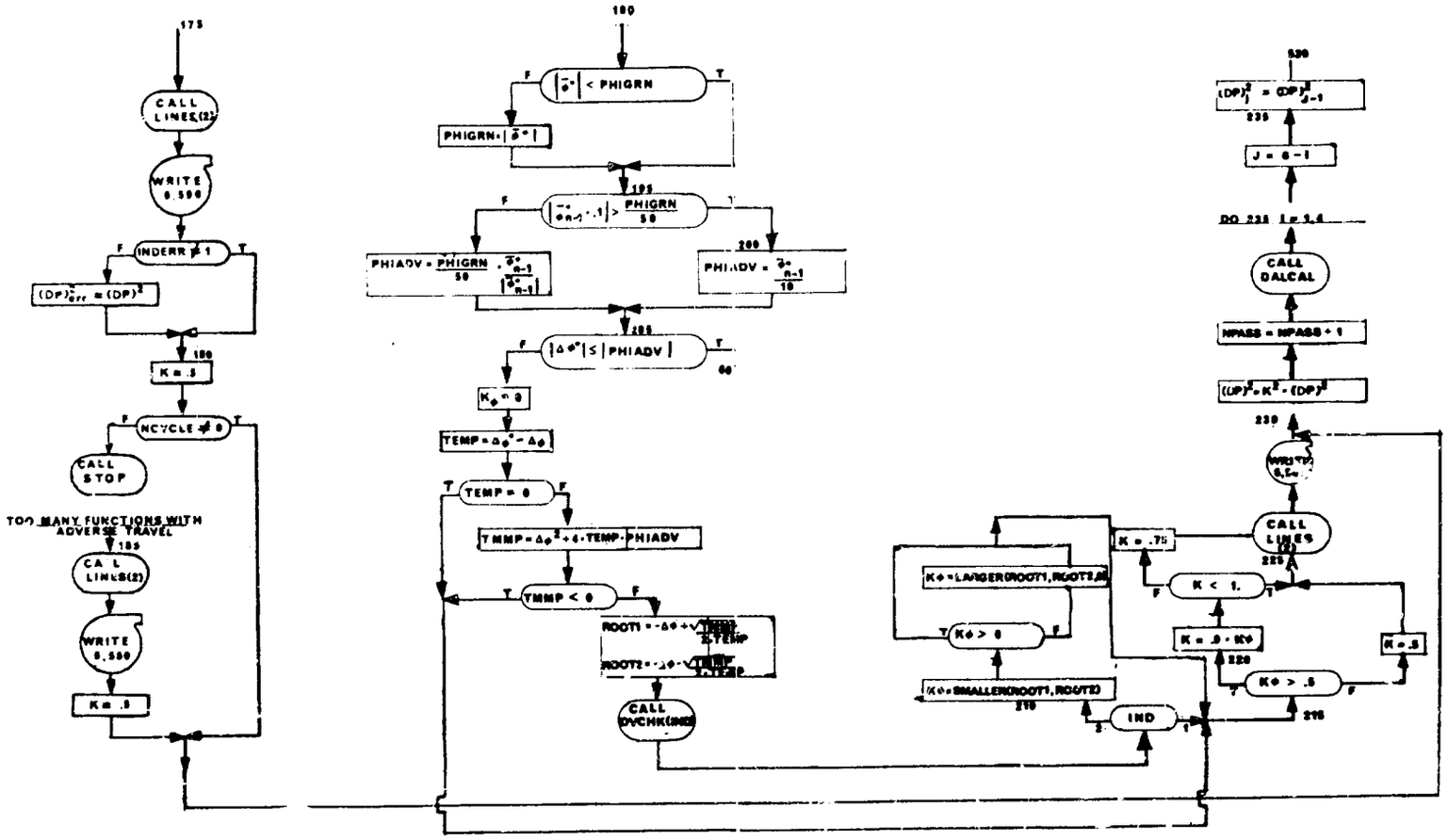
NOT REPRODUCIBLE



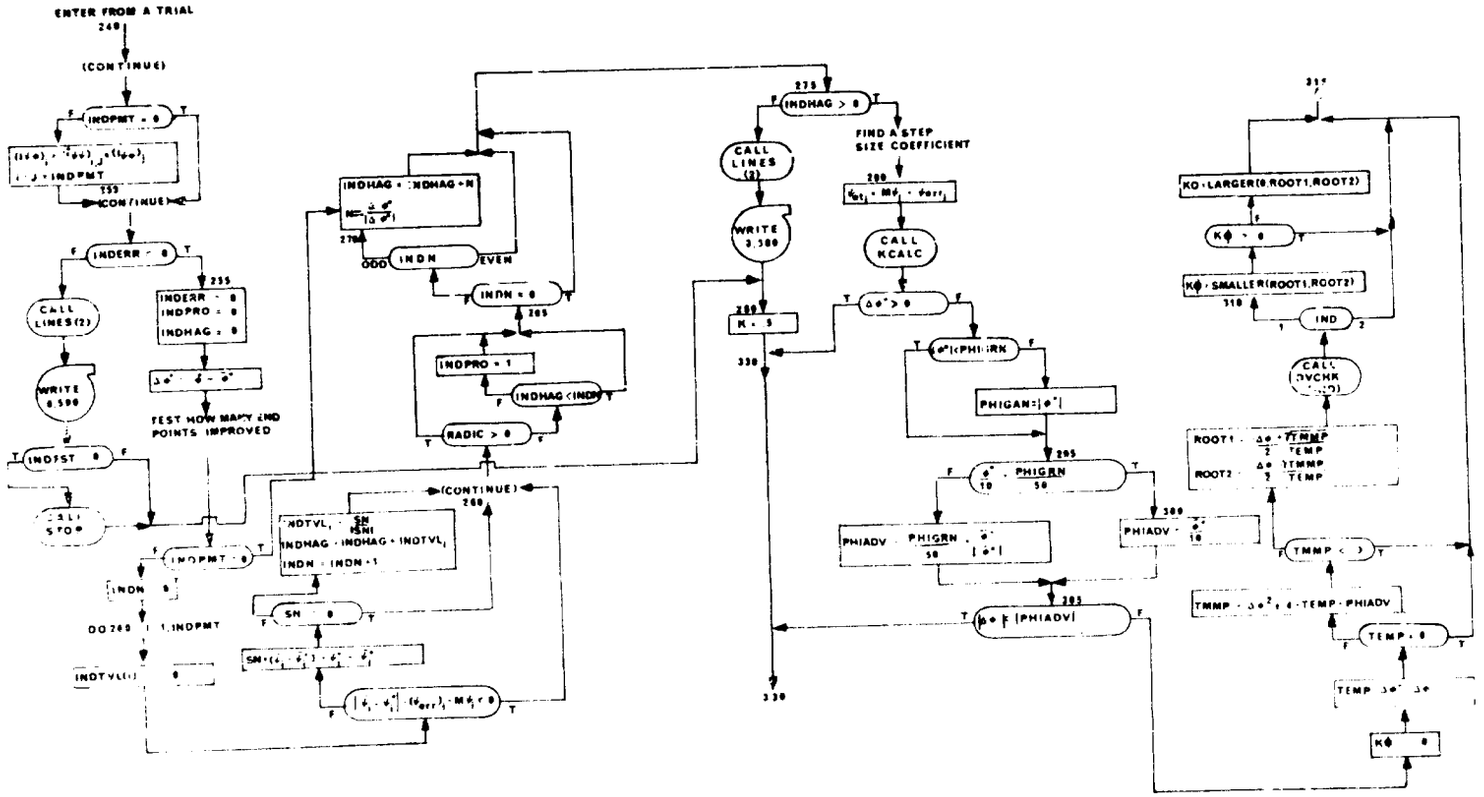

INDFST = 0
T
F
CALL DEF
NS = NSTAGE
65
CALL LINES(2)
WRITE 6,888
READ IATAP
70
CARD OUTPUT FOR RESTART
CONTINUE
INDPMT = 0
T
F
135
DO 150 I=1,INDPMT
CΨi = SMALLER (CΨi, 1.)
CΨi = CΨi + ΔCΨi
N = ΨiΨi* / |ΨiΨi*|
CONTINUE
100
(ΨBWD)i = .5
(ΨFWD)i = 2.5
(ΨBWD)i = .025
(ΨFWD)i = 1.5
85
(ΨBWD)i = 1.
(ΨFWD)i = 3.
CΨi > 1.
80
CΨi > .5
90
NCYCLE > 20
95
|INDSICi| > 10
INDSICi = INDSICi + N
75
N = -INDSICi
INDSICi - |N+INDSICi| < 0

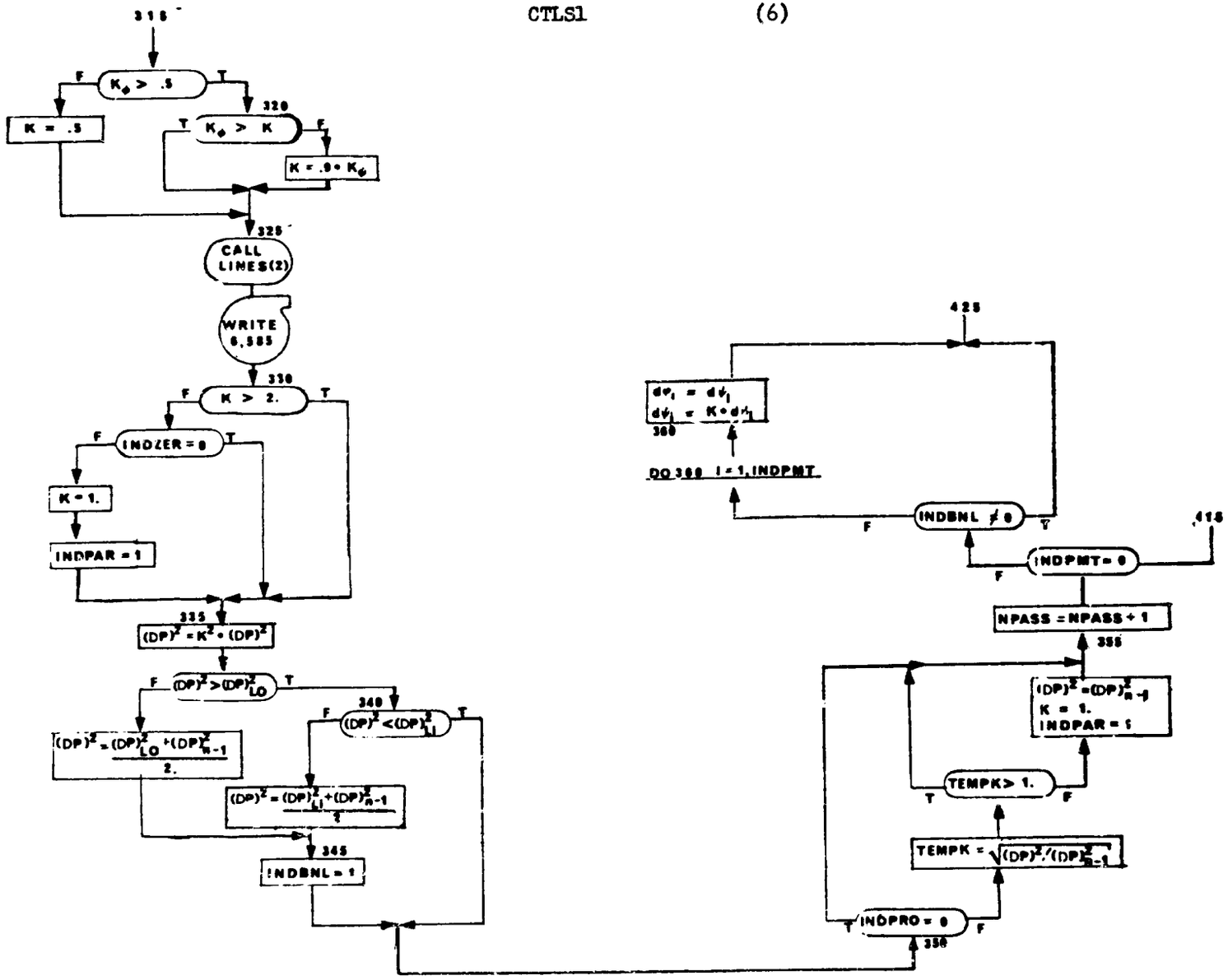


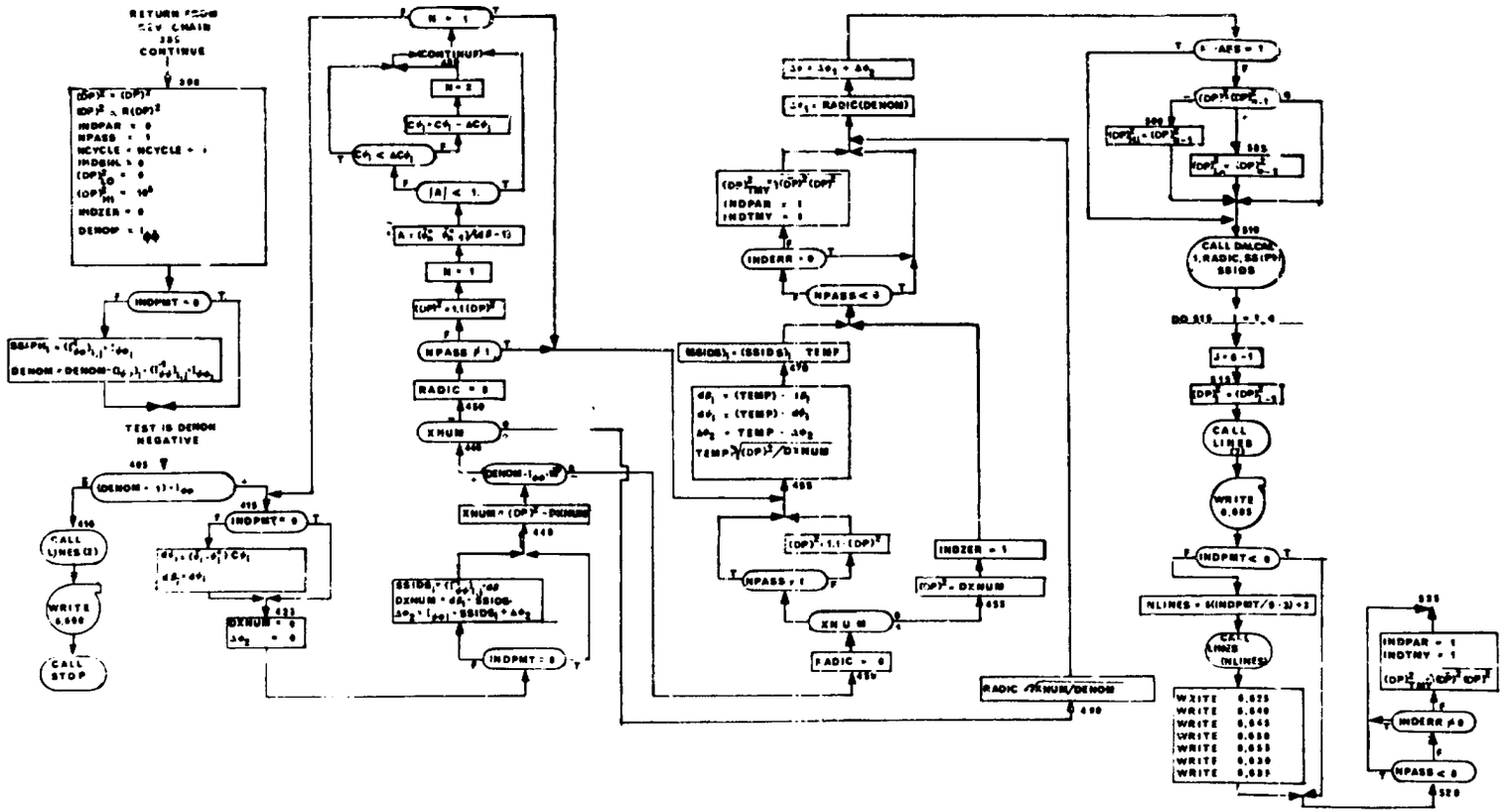
NOT REPRODUCIBLE

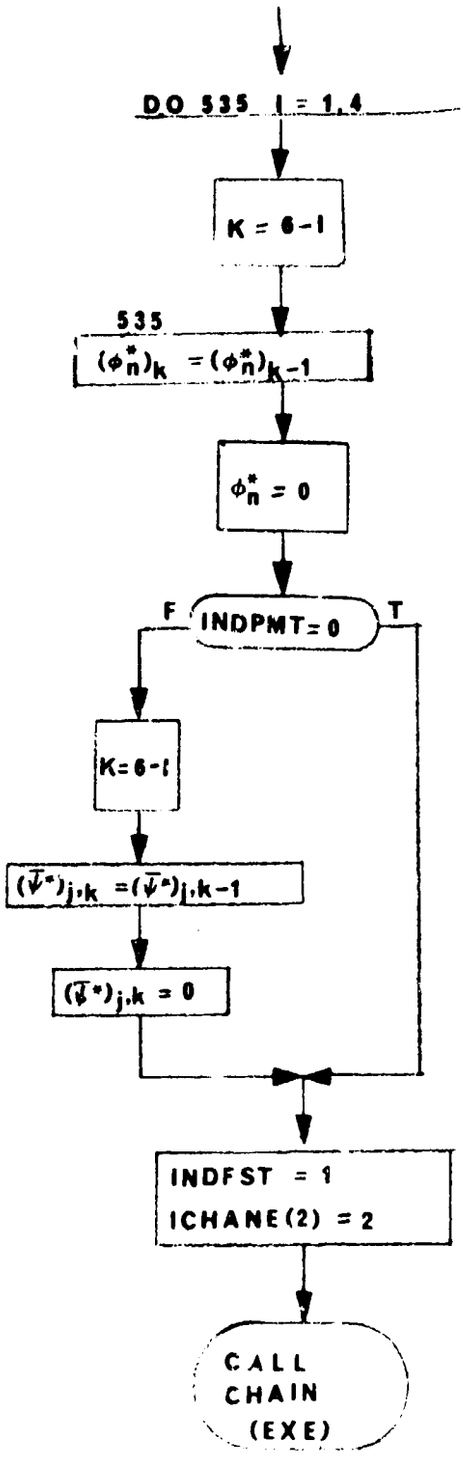

175
CALL LINES(2)
WRITE 6,500
INDERR ≠ 1
(DP)²err = (DP)²
180
K = .5
NCYCLE ≠ 0
CALL STOP
TOO MANY FUNCTIONS WITH ADVERSE TRAVEL
185
CA L LINES(2)
WRITE 6,500
K = .5
190
|δ°| < PHIGRN
PHIGRN = |δ°|
195
|δ°n-1 - 1| > PHIGRN / 50
PHIADV = PHIGRN / 50 · δ°n-1 / |δ°n-1|
200
PHIADV = δ°n-1 / 10
205
|Δφ°| ≤ |PHIADV|
60
Kφ = 0
TEMP = Δφ° - Δφ
TEMP = 0
TMMP = Δφ² + 4·TEMP·PHIADV
TMMP < 0
ROOT1 = -Δφ + √TMMP / 2.TEMP
ROOT2 = -Δφ - √TMMP / 2.TEMP
CALL DYCHK(IND)
Kφ = LARGER(ROOT1, ROOT2)
Kφ > 0
Kφ = SMALLER(ROOT1, ROOT2)
210
IND
K = .75
CALL LINES (2)
225
K < 1.
K = .9 · Kφ
220
Kφ > .5
K = .5
215
230
(DP)²J = (DP)²J-1
235
J = 6 - 1
DO 233 I = 5,6
CALL DALCAL
NPASS = NPASS + 1
(DP)² = K² · (DP)²

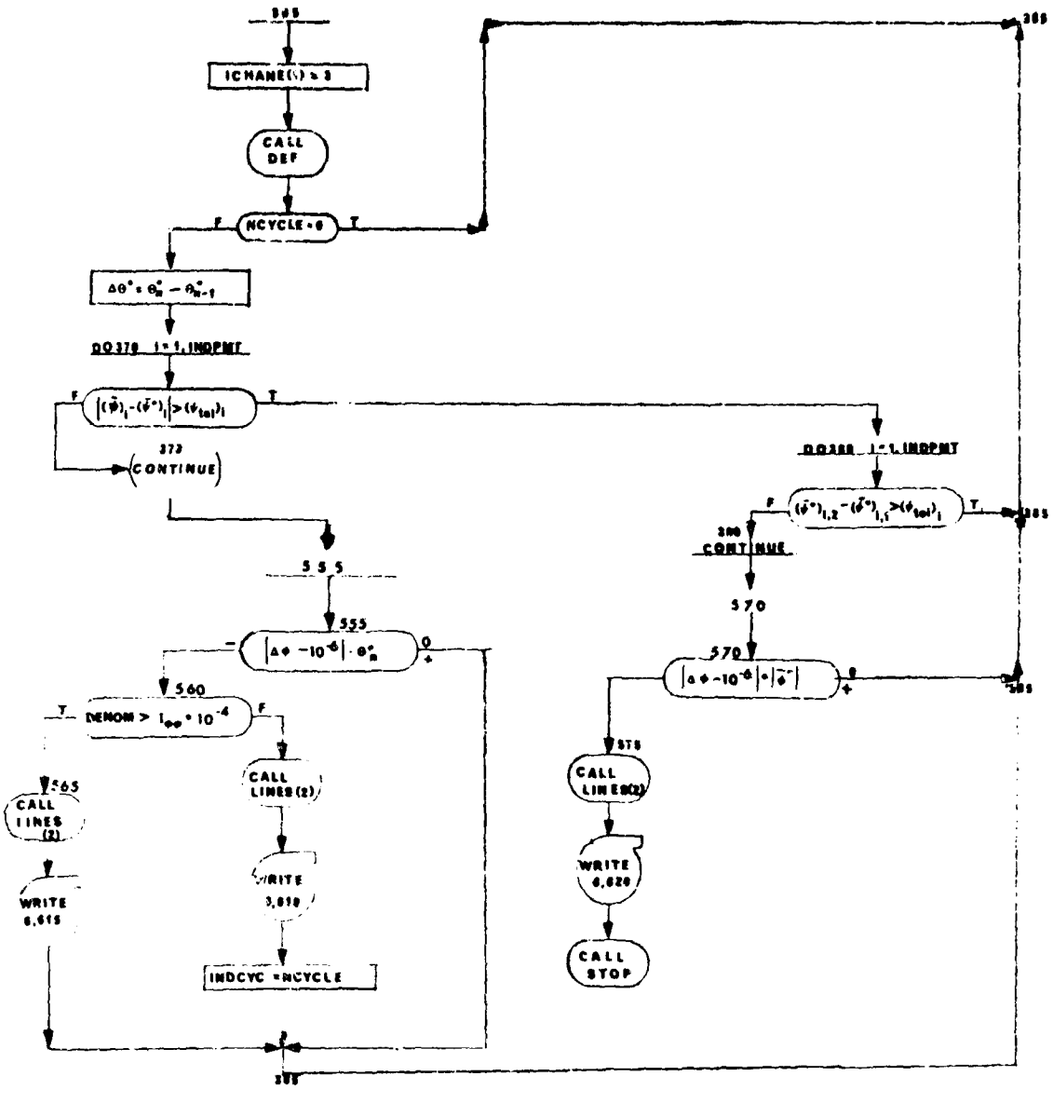
CTLS1 (5)


ENTER FROM A TRIAL
(CONTINUE)
INDPMT = 0
CALL LINES(2)
WRITE 0,590
INDERR = 0
INDPRO = 0
INDHAG = 0
TEST HOW MANY END POINTS IMPROVED
INDFST = 0
CALL STOP
INDPMT = 0
INDN = 0
DO 260 I = 1, INDPMT
INDTVL(I) = 0
INDHAG = INDHAG + N
INDN
ODD
EVEN
INDN = 0
INDPRO = 1
INDHAG < INDN
RADIC > 0
(CONTINUE)
INDTVL
INDHAG = INDHAG + INDTVL
INDN = INDN + 1
SN = 0
INDHAG > 0
CALL LINES (2)
WRITE 3,380
K = .5
FIND A STEP SIZE COEFFICIENT
CALL KCALC
PHIGAN = |φ*|
PHIGRN
PHIADV
|PHIADV|
KO = LARGER(0,ROOT1,ROOT2)
KΦ = 0
KΦ = SMALLER(ROOT1,ROOT2)
IND
CALL DVCHK
ROOT1
ROOT2
TMMP
TEMP = 0
TEMP = Δφ² + 4 · TEMP · PHIADV
KΦ = 0


315
$K_\phi > .5$
F
T
K = .5
320
$K_\phi > K$
T
F
$K = .9 \cdot K_\phi$
325
CALL LINES(2)
WRITE 6,585
330
K > 2.
F
T
INDZER = 0
F
T
K = 1.
INDPAR = 1
335
$(DP)^2 = K^2 \cdot (DP)^2$
$(DP)^2 > (DP)^2_{LO}$
F
T
340
$(DP)^2 < (DP)^2_{LI}$
F
T
$(DP)^2 = \frac{(DP)^2_{LO} + (DP)^2_{n-1}}{2.}$
$(DP)^2 = \frac{(DP)^2_{LI} + (DP)^2_{n-1}}{2}$
345
INDBNL = 1
425
$d\psi_i = d\psi_i$
$d\psi_i = K \cdot d\psi_i$
360
DO 360 I = 1, INDPMT
INDBNL ≠ 0
F
T
415
INDPMT = 0
F
NPASS = NPASS + 1
355
$(DP)^2 = (DP)^2_{n-1}$
K = 1.
INDPAR = 1
TEMPK > 1.
T
F
$TEMPK = \sqrt{(DP)^2 / (DP)^2_{n-1}}$
INDPRO = 0
T
F
350




DO 535 I = 1,4
K = 6-I
535
$(\phi_n^*)_k = (\phi_n^*)_{k-1}$
$\phi_n^* = 0$
F
INDPMT=0
T
K=6-I
$(\bar{\psi}^*)_{j,k} = (\bar{\psi}^*)_{j,k-1}$
$(\bar{\psi}^*)_{j,k} = 0$
INDFST = 1
ICHANE(2) = 2
CALL
CHAIN
(EXE)


CALL DEF
NCYCLE = 0
DO 370 I = 1, INDPMT
CONTINUE
555
555
560
DENOM > $I_{\phi\phi}$ * 10^{-4}
CALL LINES (2)
WRITE 6,615
WRITE 3,610
INDCYC = NCYCLE
CONTINUE
570
570
CALL LINES(2)
WRITE 6,620
CALL STOP

2. CTLS2 - Arbitrary Control System

Purpose

To serve as an "arbitrary" control system; namely, to perform the routine chores that must be done in the CTLS segment, but to defer the important decisions to subprogram DECIDE.

Method

The flow chart provides the best description of those tasks performed in CTLS2. CTLS2 may be entered at two different parts of the program: when segment CTLS is brought in after segment EXE or when segment CTLS is brought in after segment REV. At every point where a critical decision must be made, a call is made to DECIDE with a particular entry point.

There are four primary decisions that must be made by DECIDE:

a. The $\delta\alpha$ mode shape to use.

b. Whether to accept a pass as a valid step.

c. If the decision concerning (b) is negative, the step size to use for the next pass.

d. Whether to compute partials on the next pass.

Entry Point for DECIDE	Situation
1	20 passes have been made; decide whether to terminate.
2	Initialize parameters on the nominal trajectory.
3	A pass for which partials were computed has missed cutoff.
4	A pass for which partials were computed must be accepted or rejected.
5	To output additional restart cards.
6	A pass for which partials were not computed has missed cutoff.
7	A pass for which partials were not computed must be evaluated.
8	Initialize parameters for each cycle (after REV); set $d\beta$.
9	The mode shape must be approved (if not, determine a new $d\beta$).
10	After a pass, to set the number of literal constraints, INDCON; this will be the same as INDPMT if it is not set by DECIDE.
11	To print any information after the perturbation has been made (i.e. after DALCAL).
12	DENOM $= I_{\varphi\varphi} - I_{\Psi\varphi} I_{\Psi\Psi}^{-1} I_{\Psi\phi} < 0$ (This situation is theoretically impossible.)
13	To set INDCON (see 10) after the reverse integration. Also, to invert $I_{\Psi\Psi}$.

CTLS2 takes care of the following important chores itself:

1. Updating indicators, e.g. INDFST, NCYCLE, NPASS, etc.

2. Outputting the CTLS Summary print.

3. Outputting restart cards for CTABLE and DELP2.

4. Updating PSSTAR, PHSTAR, PSBRST, and PHBRST tables. These tables contain the values printed under "CTLS Summary".

5. Calling the next segment (either GRAPH or REV).

6. Computing DENOM = $I_{\varphi\varphi} - I_{\psi\varphi}\, I_{\psi\psi}^{-1}\, I_{\psi\varphi}$ and SSIPH = $I_{\psi\varphi}\, I_{\psi\psi}$ and SSIDS = $d\beta\, I_{\psi\psi}^{-1}$

7. Recalculating parameters (e.g. $d\beta$) to be compatible with the change DP^2 determined by the value of COEFK which is set in DECIDE.

8. Computing RADIC = $\sqrt{\dfrac{DP^2 - d\beta\, I_{\psi\psi}^{-1}\, d\beta}{I_{\varphi\varphi} - I_{\psi\varphi}\, I_{\psi\psi}^{-1}\, I_{\psi\varphi}}}$

9. Calling DALCAL to either compute a new mode shape for $\delta\alpha$ or to change the amplitude of the existing mode shape.

10. Terminating the case when it is impossible to continue.

11. Making sure that $DP^2 - d\beta\, I_{\psi\psi}^{-1} d\beta$ is not negative. DECIDE will be called repetitively with entry point 9 until this condition is satisfied.

Communication between CTLS2 and DECIDE is accomplished through the calling sequence, primarily. There are three indicators used here which need clarification:

INDAMP: Decide sets INDAMP $\neq 0$ if the desire is to change the amplitude of the current perturbation (by multiplying the perturbation by COEFK). INDAMP is set to 0 by DECIDE if a new mode shape is to be attempted.

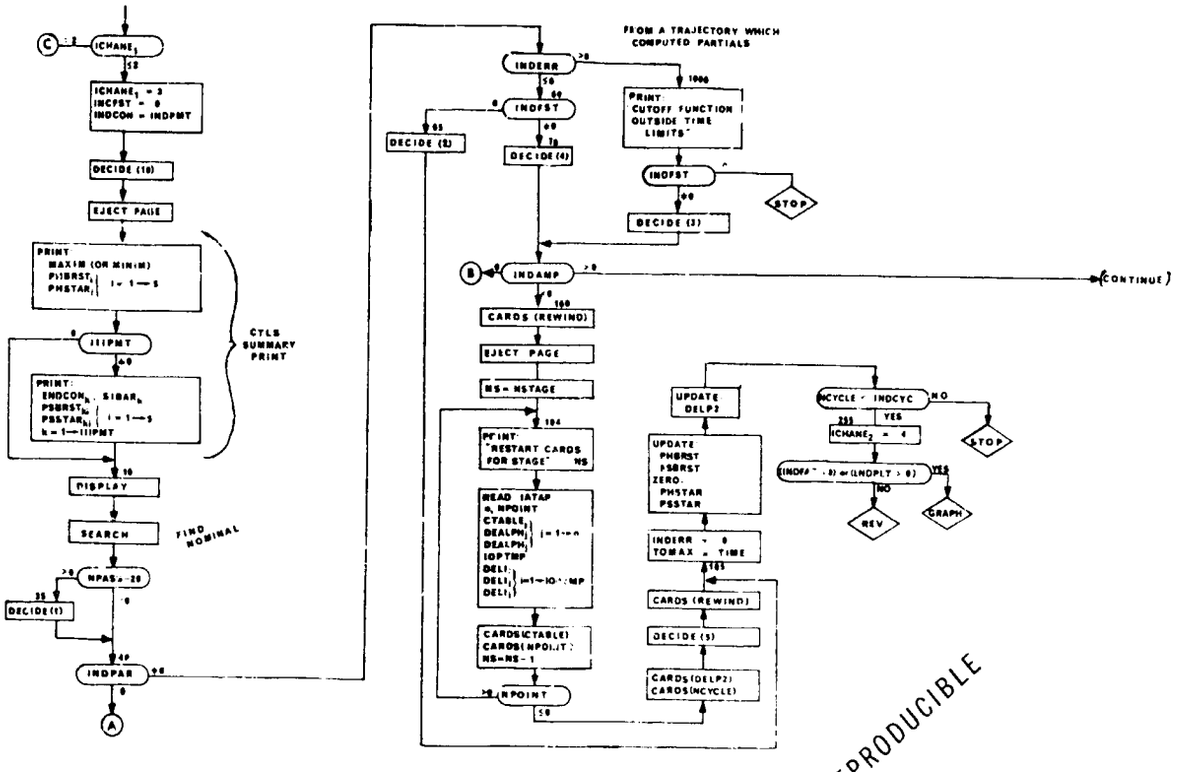
ITBAL: ITBAL is updated by CTLS2; it is the number of calls made to DECIDE (9) in order to attempt to find a feasible mode shape.

INDBAL: INDBAL is set $\neq 0$ by DECIDE(9) to inform CTLS2 that it is satisfied with the mode shape.

Remarks

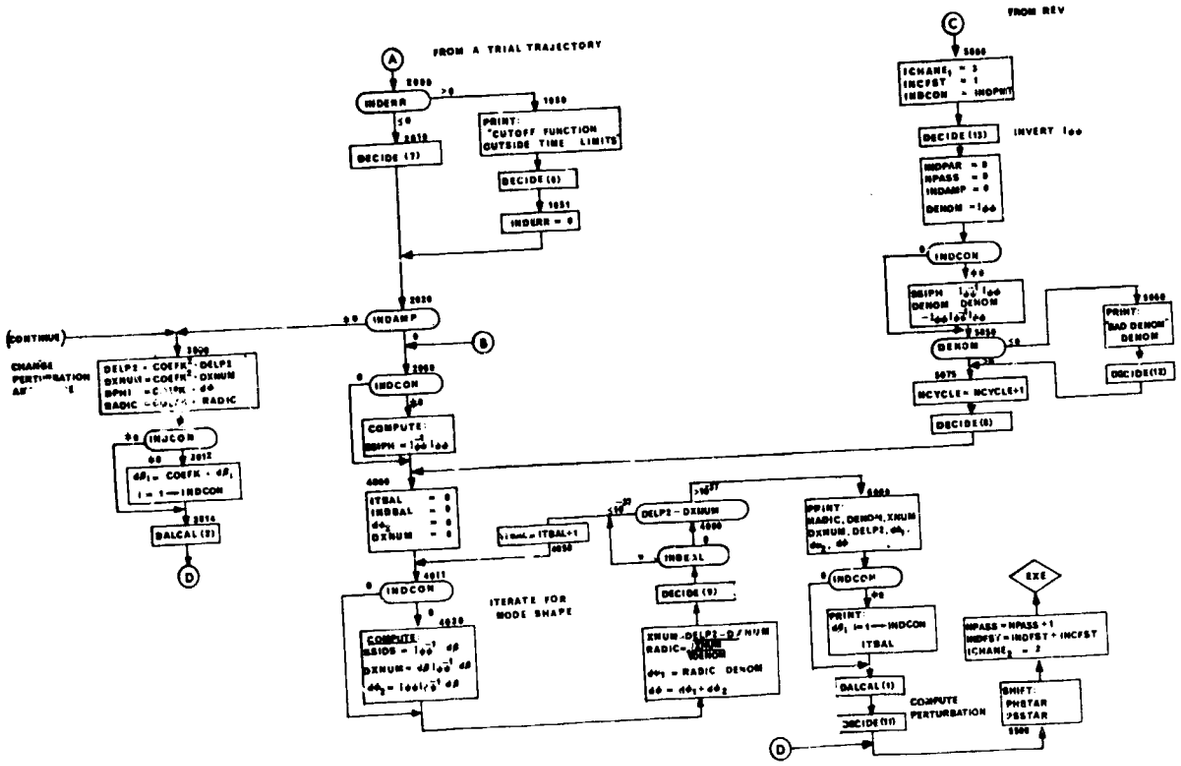
Changes in the step size made by DECIDE, must be done by setting COEFK; S2 will set DELP2 = COEFK^2* DELP2.

CTLS2 (1)



NOT REPRODUCIBLE

CTLS2 (2.


FROM A TRIAL TRAJECTORY
FROM REV
INDERR
PRINT: "CUTOFF FUNCTION OUTSIDE TIME LIMITS"
DECIDE (7)
DECIDE (8)
INDERR = 0
INDAMP
INDCON
COMPUTE:
ITBAL = 0
INBBAL = 0
DXNUM = 0
DELP2 - DXNUM
INDEXL
DECIDE (9)
ITERATE FOR MODE SHAPE
CONTINUE
CHANGE PERTURBATION
INJCON
DALCAL (3)
ICHANE = 3
INCFST = 1
INDCON = INDPRT
DECIDE (13)
INVERT
INDPAR = 0
NPASS = 0
INDAMP = 0
DENOM = 1
DENOM
PRINT: BAD DENOM DENOM
DECIDE (12)
NCYCLE = NCYCLE+1
DECIDE (8)
PRINT: INDCON ITBAL
DALCAL (3)
DECIDE (11)
COMPUTE PERTURBATION
EXE
NPASS = NPASS + 1
INDFST = INDFST + INCFST
ICHANE = 2
SHIFT: PHETAR PSSTAR

3. DISPLAY - Console Display Routine

Purpose:

To output data to the display console on the 10-inch display tube.

Method:

By setting sense switch 6 a peripheral processor is selected and the CTLS summary is displayed on a 10-inch display tube

Usage:

Entry is made to the routine by the following statement
CALL DISPLAY (IPAY,IIIPMT)

where

IPAY is payoff function
IIIPMT is the number of constraints.

4. SEARCH - Search Routine

Purpose:

Compute a nominal starting trajectory for the optimization program.

Method:

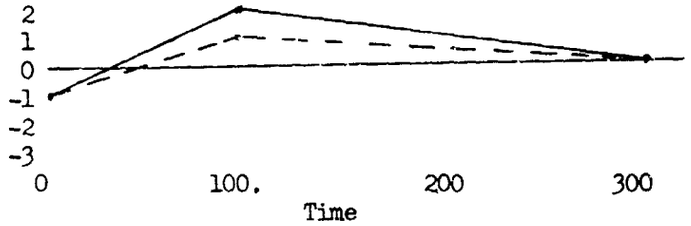
The search routine allows the user to vary a control parameter to satisfy any single end constraint in the hope this is a better nominal trajectory. Any parameter used in the initial condition optimization or any half point in the CTABLE may be used by the search routine as an adjusting variable. Any trajectory the search routine finds may or may not be a good trajectory for the optimization program. In fact, sometimes what appears to be a good starting trajectory by inspection turns out not to be a good nominal trajectory for the optimization program. Therefore, if one fails to get started with the first nominal, a second attempt should be made with another nominal.

Assume the same test case as before but now set a half point in the CTABLE as the adjusting parameter.

Example:

NPOINT	3
CTABLE	0, 100., 300., -1., 2., 0

If the above table is assumed to be α then it has the shape as follows:



LIMDER	BCD	1QEXN
DESIRE	BCD	1HGC7F
CONVAR		2.
SECLIM		1.
KPOINT		5
LIMIT		2000.
QEXN		270000.

A half point alpha at time 100. seconds is the control variable. LIMDER, DESIRE, LIMIT, and QEXN are unchanged from the previous example. CONVAR is set equal to 2. SECLIM is set to 1. and on the second try the α history would appear as the dotted line. ADJUST is not needed when working with the CTABLE. KPOINT is set equal to 5 which is the 5th location in the CTABLE.

Assume that conditions exist such that one would like to have a trajectory that would terminate at 270,000 feet altitude $\pm$ 2000 feet tolerance by adjusting the initial γ

Entry is made to the search routine with one data card.

INDMOD	1

Other data is expected if the above data card is included.

LIMDER	BCD	1QEXN
DESIRE	BCD	1HGC7F
ADJUST	BCD	1GAM7D
CONVAR		0
SECLIM		2.
KPOINT		0
LIMIT		2000.
QEXN		270000.
GAM7D		0

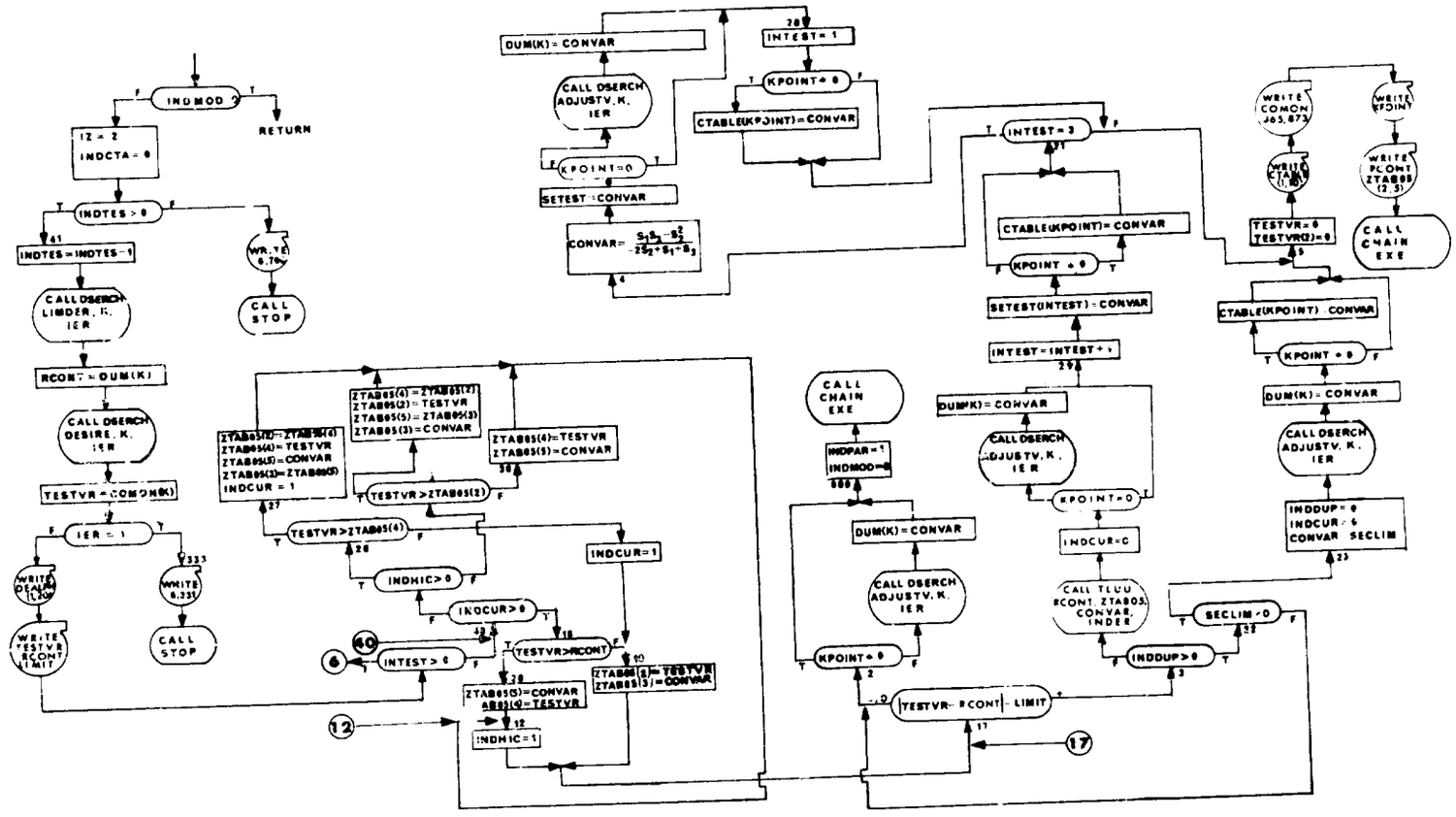
LIMDER contains the BCD word of some unused cell in core. QEXN contains the required end value of altitude. DESIRE contains the BCD word of the end constraint. ADJUST contains the BCD word of the control parameter being varied. CONVAR should have the same initial value as the control variable. SECLIM is the value of the control variable on the second try. KPOINT is the location of the half point in CTABLE to be varied. In this case γ is the control variable and KPOINT must be set to zero. LIMIT is the tolerance on altitude. A + or - is assumed on the tolerance.

Any location in CTABLE may be varied the same by changing the necessary data.

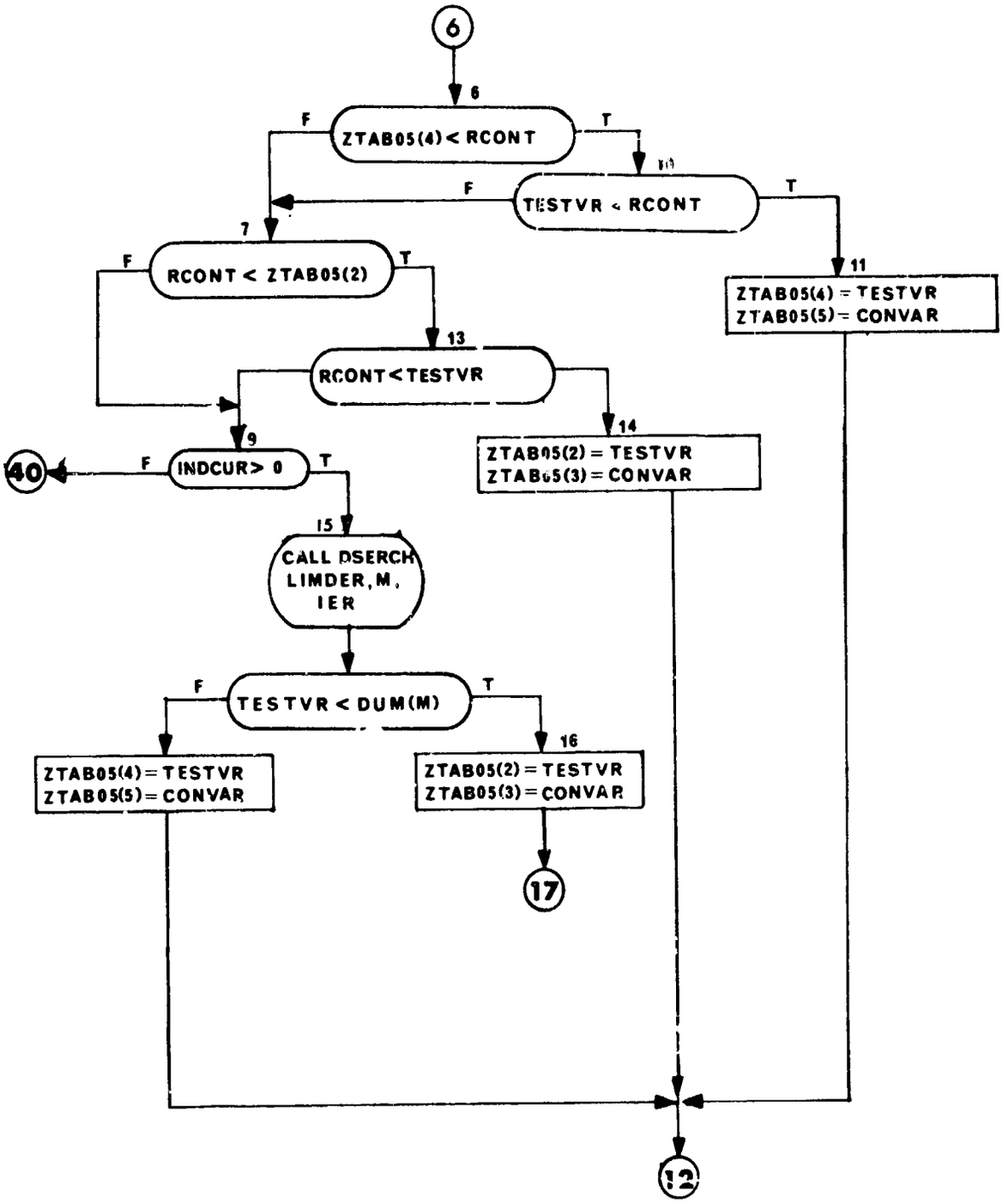
Remarks:

All data for the search routine should be in DATA1.

The CTABLE must be put in DATA1 as a function of trajectory time (INDCTA = 0).


INDMOD ?
RETURN
IZ = 2
INDCTA = 0
INDTES > 0
INDTES = INDTES - 1
CALL DSERCH LIMDER, K, IER
RCONT = DUM(K)
CALL DSERCH DESIRE, K, IER
TESTVR = COMCH(K)
IER = 1
WRITE DEALER (1,20)
WRITE TESTVR RCONT LIMIT
CALL STOP
WRITE (6,333)
CALL STOP
DUM(K) = CONVAR
CALL DSERCH ADJUSTV, K, IER
KPOINT = 0
SETEST = CONVAR
CONVAR = ...
INTEST = 1
KPOINT = 0
CTABLE(KPOINT) = CONVAR
INTEST = 3
CTABLE(KPOINT) = CONVAR
KPOINT = 0
SETEST(INTEST) = CONVAR
INTEST = INTEST + 1
DUM(K) = CONVAR
CALL DSERCH ADJUSTV, K, IER
KPOINT = 0
INDCUR = 0
CALL TLUU RCONT, ZTAB05, CONVAR, INDER
INDDUP > 0
SECLIM = 0
WRITE COMON (65,67)
WRITE CTABLE (1,10)
WRITE KPOINT
WRITE PCONT ZTAB05 (2,5)
CALL CHAIN EXE
TESTVR = 0
TESTVR(2) = 0
CTABLE(KPOINT) = CONVAR
KPOINT = 0
DUM(K) = CONVAR
CALL DSERCH ADJUSTV, K, IER
INDDUP = 0
INDCUR = 6
CONVAR SECLIM
CALL CHAIN EXE
INDPAR = 1
INDMOD = 0
DUM(K) = CONVAR
CALL DSERCH ADJUSTV, K, IER
KPOINT = 0
TESTVR - RCONT - LIMIT
ZTAB05(4) = ZTAB05(2)
ZTAB05(2) = TESTVR
ZTAB05(5) = ZTAB05(3)
ZTAB05(3) = CONVAR
ZTAB05(4) = TESTVR
ZTAB05(5) = CONVAR
INDCUR = 1
TESTVR > ZTAB05(2)
TESTVR > ZTAB05(4)
INDHIC > 0
INDCUR = 1
INDCUR > 0
INTEST > 0
TESTVR > RCONT
ZTAB05(5) = CONVAR
ZTAB05(4) = TESTVR
INDHIC = 1
ZTAB05(4) = TESTVR
ZTAB05(3) = CONVAR

SEARCH (2)


6
6
F
ZTAB05(4) < RCONT
T
10
F
TESTVR < RCONT
T
7
F
RCONT < ZTAB05(2)
T
11
ZTAB05(4) = TESTVR
ZTAB05(5) = CONVAR
13
RCONT < TESTVR
9
14
40
F
INDCUR > 0
T
ZTAB05(2) = TESTVR
ZTAB05(3) = CONVAR
15
CALL DSERCH
LIMDER, M,
IER
F
TESTVR < DUM(M)
T
16
ZTAB05(4) = TESTVR
ZTAB05(5) = CONVAR
ZTAB05(2) = TESTVR
ZTAB05(3) = CONVAR
17
12

5. CARDS - Restart Cards Routine

Purpose:

To output information which may be used to restart the program.

Usage:

CALL CARDS (SYM,VAR,N,ITAPE)

SYM = symbol to be output or signal to rewind card tape.

VAR = location of values to be output.

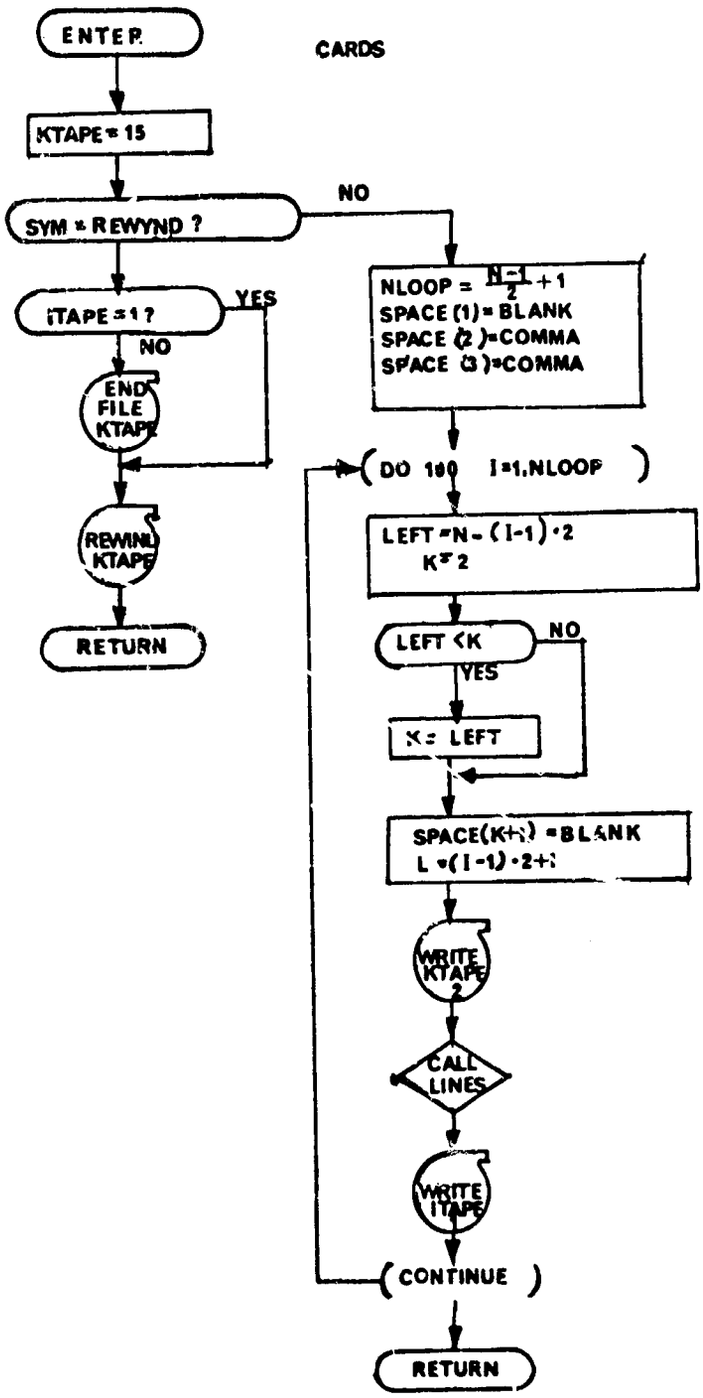
N = number of values to be output.

ITAPE = tape on which printed output occurs or signal to place end of file on card tape.

Restart cards for each cycle are output on the output tape ITAPE. Restart cards for the last completed cycle are output on the card tape KTAPE. ITAPE=6 and KTAPE=15. All values are output in OCTAL.

Remarks:

CTLS1 and CTLS2 call CARDS.
CARDS calls LINES.


ENTER
CARDS
KTAPE=15
SYM=REWYND ?
NO
ITAPE=1 ?
YES
NO
END FILE KTAPE
REWIND KTAPE
RETURN
NLOOP = (N-1)/2 + 1
SPACE(1)=BLANK
SPACE(2)=COMMA
SPACE(3)=COMMA
DO 100 I=1,NLOOP
LEFT=N-(I-1)*2
K=2
LEFT<K
NO
YES
K=LEFT
SPACE(K+1)=BLANK
L=(I-1)*2+1
WRITE KTAPE 2
CALL LINES
WRITE ITAPE
CONTINUE
RETURN

6. DALCAL - Delta Alpha Calculation

Purpose:

This subprogram of CTLS computes $\delta\sigma$ and δx_o by combining the various linearly independent solutions to the adjoint equations computed in REV. It is necessary that this task be deferred until this time because the coefficients in the linear combination for computing $\delta\sigma$ and δx_o are determined by the control system and hence are not available during REV. Also, depending on the control system being used, it may be necessary to change the coefficients from one pass to the next. The values of the coefficients ($I_{\Psi\Psi}^{-1} d\beta$ = SSIDS) determine the mode shape of the perturbation. This subprogram not only computes $\delta\sigma$ (DEALPH) but it updates (CTABLE) with this perturbation. Perturbations, δx_o (DELI), to initial conditions, x_o, (VALINS), are computed and the initial conditions are updated here in a manner analagous to the $\delta\sigma$ computations.

Method:

There are two entry points to this subprogram. The first entry point is for computing the perturbation for the first pass of a cycle or for any pass where the mode shape is being changed. The second entry point is for changing the amplitude of the previous perturbation.

This subprogram will terminate the case, if, for any given stage, the history exceeds the limits of the dimension of CTABLE.

At entry point 1, $\delta\alpha$ and δx_o are computed according to:

$$\delta\sigma = \left[W^{-1}G'\lambda_{\Phi\Omega} - W^{-1}G'\lambda'_{\Psi\Omega} I_{\Psi\Psi}^{-1} I_{\Psi\Phi}\right] \sqrt{\frac{DP^2 - d\beta I_{\Psi\Psi}^{-1} d\beta}{I_{\Phi\Phi} - I_{\Psi\Phi} I_{\Psi\Psi}^{-1} I_{\Psi\Phi}}} + W^{-1}G'\lambda'_{\Psi\Omega} I_{\Psi\Psi}^{-1} d\beta$$

$$\delta x_o = \left[U^{-1}R'\lambda_{\Phi\Omega} - U^{-1}R'\lambda'_{\Psi\Omega} I_{\Psi\Psi}^{-1} I_{\Psi\Phi}\right] \sqrt{\frac{DP^2 - d\beta I_{\Psi\Psi}^{-1} d\beta}{I_{\Phi\Phi} - I_{\Psi\Phi} I_{\Psi\Psi}^{-1} I_{\Psi\Phi}}} + U^{-1}R'\lambda'_{\Psi\Omega} I_{\Psi\Psi}^{-1} d\beta$$

In the above expressions, the radical (RADIC) is available since it has been computed in CTLS. Also $I_{\Psi\Psi}^{-1} d\beta$ = SSIDS, and $I_{\Psi\Psi}^{-1} I_{\Psi\Phi}$ = SSIPH have been computed in CTLS.

$\delta\sigma$ calculations will be made for each major stage. δx_o calculations are made only for the first stage.

The outline of the subprogram follows:

Entry Point 1

(1) Get $\lambda_{\Phi\Omega}GW^{-1}$ and $\lambda_{\Psi\Omega}GW^{-1}$, α, and W^{-1} for the ith major stage from tape ILTAP (last stage comes first).

(2) Compute $\delta\alpha$ from $\lambda_{\Phi\Omega}GW^{-1}$, $\lambda_{\Psi\Omega}GW^{-1}$, RADIC, SSIDS, and SSIPH.

(3) Compute the new α: $\alpha = \alpha + \delta\alpha$.

(4) Plot α, $\delta\alpha$ and W^{-1} if requested.

(5) If more stages go to (10); otherwise go to (6).

(6) Get $\lambda_{\Phi\Omega}RU^{-1}$, $\lambda_{\Psi\Omega}RU^{-1}$, and x_o from tape ILTAP.

(7) Compute δx_o from $\lambda_{\Phi\Omega}RU^{-1}$, $\lambda_{\Psi\Omega}RU^{-1}$, RADIC, SSIDS, and SSIPH.

(8) Compute the new x_o: $x_o = x_o + \delta x_o$.

(9) Print the x_o, δx_o, U^{-1} if requested.

(10) Put α, W^{-1}, $\delta\alpha$, x_o, U^{-1}, δx_o on IATAP (all in one record).

(11) If there are more major stages to go (1); otherwise return.

Entry Point 2

(1) Get α, W^{-1}, $\delta\alpha$, x_o, U^{-1}, δx_o from tape IATAP.

(2) Recover to the last good α: $\alpha = \alpha - \delta\alpha$

(3) Modify $\delta\alpha$: $\delta\alpha = k \cdot \delta\alpha$

(4) Compute the new α: $\alpha = \alpha + \delta\alpha$

(5) If more major stages, go to (9); otherwise go to (6).

(6) Recover to the last good x_o: $x_o = x_o - \delta x_o$

(7) Modify δx_o: $\delta x_o = k \cdot \delta x_o$

(8) Compute the new x_o: $x_o = x_o + \delta x_o$

(9) Put α, W^{-1}, $\delta\alpha$, x_o, U^{-1}, δx_o on tape ILTAP (all in one record).

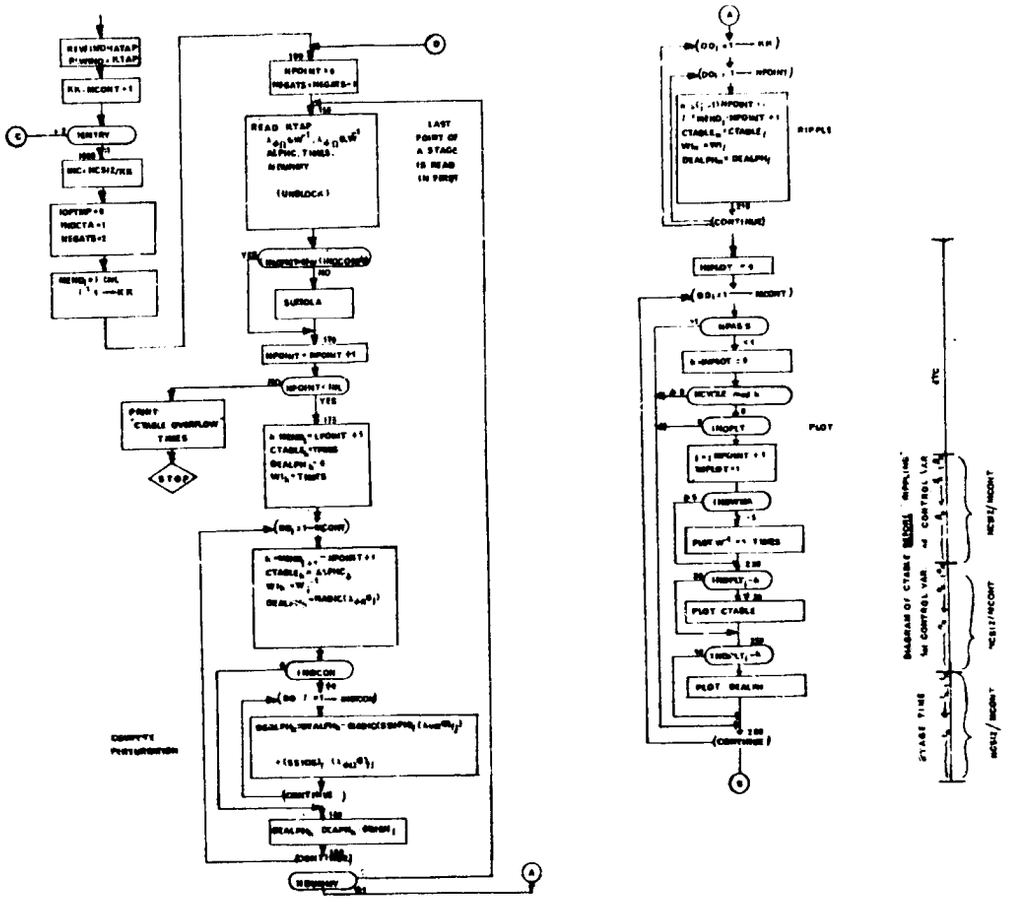
(10) If more stages go to (1); otherwise go to (11).

(11) Switch IATAP and ILTAP and return.

Once entry point 2 has been called, entry point 1 may not be called again until the next cycle because the information it needs on ILTAP is gone.

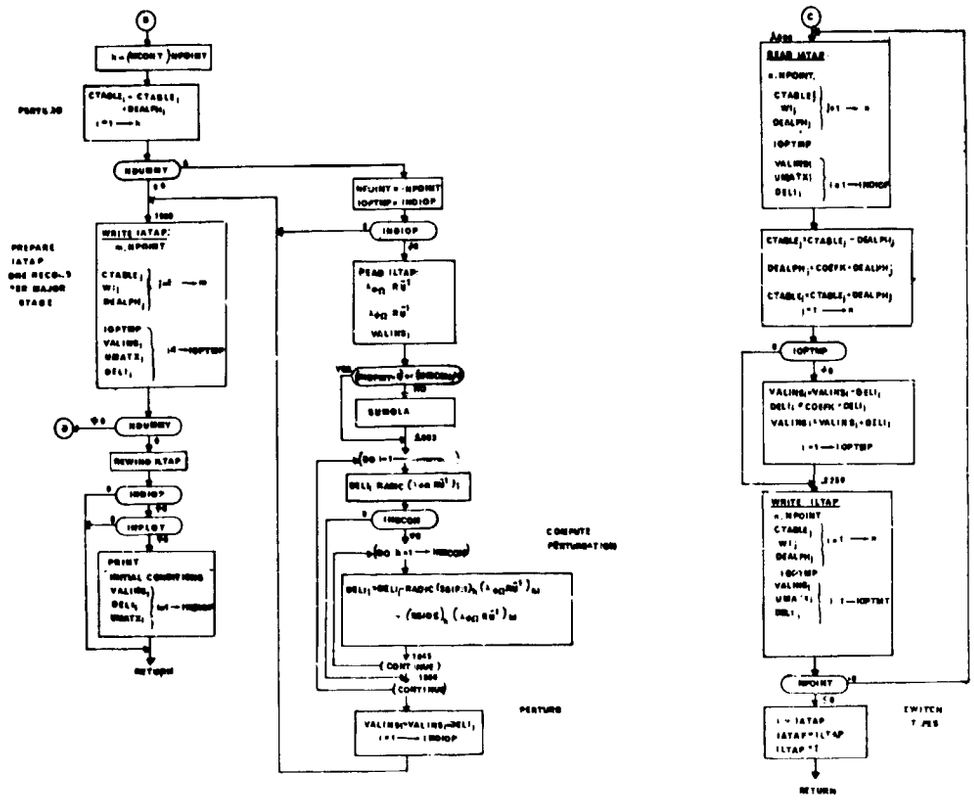
The information from REV on ILTAP is blocked. Reading this tape (entry point 1) is accomplished by calls to the ad-hoc unblocking routine UNBLOCK.

DALCAL (1)



NOT REPRODUCIBLE

DALCAL (2)



7. KCALC - Step Size Logic Routine for Control System 1

Purpose

This subroutine determines the "best" step size coefficient based on the given data, and is intended to be a part of the over-all logic necessary for step size evaluation and control.

Usage

CALL KCALC (ARG1, ARG2,...,ARG14, ARG15)

where

ARG1 = Φ^*	ARG8 = $\bar{\Psi}_{NL}(1)$
ARG2 = $\bar{\Phi}^*$	ARG9 = $\Psi_{ER}(1)$
ARG3 = $\Delta\Phi$	ARG10 = $\Psi_{BWD}(1)$
ARG4 = $\bar{\Phi}_{NL}$	ARG11 = $\Psi_{FWD}(1)$
ARG5 = $\bar{\Psi}(1)$	ARG12 = $C_{\Psi}(1)$
ARG6 = $\Psi^*(1)$	ARG13 = $\Delta\Psi(1)$
ARG7 = $\bar{\Psi}^*(1)$	ARG14 = RADIC
	ARG15 = Variable name where K is to be stored.

This routine is designed for use with the "CTLS1" control system. An initial set of step size coefficients (K_{Φ}^*, $K_{\Psi 2}^*$, ..., K_{Ψ}^*) is generated from the given data. This initial set of coefficients is used to control the calculation of a "new" set of step size coefficients. (K'_1, K'_2, ..., K'_n).

The "best" value (K) is then calculated from the new set of coefficients and an exit is made from the subroutine.

The initial set of step size coefficients is generated in the following manner:

GIVEN Φ^*, $\bar{\Phi}^*$, $\Delta\Phi$, $\bar{\Phi}_{NL}$, Ψ_i^*, $\bar{\Psi}_i^*$, $\bar{\Psi}_{NL_i}$, $\Delta\Psi_i$

THEN $\Delta\Phi^* = (\Phi^* - \bar{\Phi}^*)$; $\Delta\Psi_i^* = (\Psi_i^* - \bar{\Psi}_i^*)$;

$$\Phi_{NL} = \frac{(\Delta\Phi^* - \Delta\Phi)}{\Delta\Phi}; \quad \Psi_{NL_i} = \frac{(\Delta\Psi_i^* - \Delta\Psi_i)}{\Delta\Psi_i}$$

$$K_{\Phi}^* = \frac{\bar{\Phi}_{NL}}{\Phi_{NL}} \qquad K_{\Psi i}^* = \frac{\bar{\Psi}_{NL_i}}{\bar{\Psi}_{NL_i}},$$

where i = 1, 2, ..., n.

The values (K_{Φ}^*, $K_{\Psi 1}^*$, $K_{\Psi 2}^*$, ..., $K_{\Psi p}^*$) are re-arranged into descending order (K_1, K_2, ..., K_{n+1}). The value K_1 must be such that $.5 \leq K_1$.

If this condition is not met, then the "best" value (K) is defined to be .5 and an exit is made from the subroutine.

If this condition is met, then a logic scheme is applied to calculate a new set of coefficients.

The new set of step size coefficients (K'_1, K'_2, ..., K'_n) is calculated as follows:

a) Tolerance check (only for Ψ_i constraints)

GIVEN $\Psi_i = \Psi_i^*$;

THEN $\Psi_i' = \left| \Psi_i - \Psi_i^* \right|$,

where i = 1, 2, ..., n.

GIVEN Ψ_i', Ψ_{ER_i};

THEN if $\left| \Psi_i' \right| \leq \Psi_{ER_i}$, Test K_1;

if $K_1 = K_{\Psi i}^*$, then proceed to next i; (*), otherwise perform (b), (c), and (d); where i = 1, 2, ..., n

b) End point movement (only for Ψ_i constraints)

GIVEN $\bar{\Psi}_i$, $\bar{\Psi}_i^*$, $\Delta\Psi_i^*$, Ψ_{FWD_i}, Ψ_{BWD_i}, C_{Ψ_i};

THEN $\Delta = (\bar{\Psi}_i - \bar{\Psi}_i^*)\, \Delta\Psi_i^*$;

if $\Delta \leq 0$,

$$\Psi_{MOT_i} = \frac{\Delta\Psi_i \left| (\bar{\Psi}_i - \bar{\Psi}_i^*)\, \Psi_{BWD_i},\ C_{\Psi_i} \right|_i}{\Delta\Psi_i^*}$$

otherwise,

$$\Psi_{MOT_i} = \frac{\Delta\Psi_i \left| (\bar{\Psi}_i - \bar{\Psi}_i)\, \Psi_{FWD_i}\ C_{\Psi_i} \right|_i}{\left| \Delta\Psi_i^* \right|}$$

where i = 1, 2, ..., n.

c) Calc roots K_i'', K_i''' (only for Ψ_i constraints)

GIVEN $\Delta\Psi_i^*$, $\Delta\Psi_i$, Ψ_{MOT_i};

THEN $(K_i'', K_i''') = \dfrac{-B \pm \sqrt{B^2 - 4AC}}{2A}$ (**)

where $A = (\Delta\Psi_i^* - \Delta\Psi_i)$,

$B = \Delta\Psi_i$,

$C = -\Psi_{MOT}$, and $i = 1, 2, \ldots, n$.

d) Test K_i'', K_i''' (only for Ψ_i constraints)

For all i, (i = 1, 2, ..., n), the value of K_i' must be chosen such that K_i' = least positive value and such that $K_i' \leq K_{i-1}'$, where $K_o' = K_1$. If $(K_i'', K_i''') < 0$, then no new K_i value is computed.

The "best" value (K) then is defined as $.5 \leq K_n' \leq 2.0$; and an exit is made from the subroutine.

(*)
It should be noted that recycling occurs if the Ψ_i constraint error is within the prescribed tolerance (Ψ_{ER_i}) and the controlling step size coefficient (K_1) is $K_{\Psi i}^*$. At the end of (d) the controlling step size coefficient is changed to ($K_2, K_3, \ldots, K_{n+1}$) and the cycle restarted at (a). If recycling is necessary the present "best" value ($K_n' = K_\ell$) is saved and the recalculated "best" value ($K_{n+1} = K_{\ell+1}$) must be such that $K_{\ell+1} \geq K_\ell$

where $\ell = 1, 2, \ldots, n$.

If all constraint errors are within the prescribed tolerances, then K_Φ^* is chosen to be the "best" value (K) and an exit is made from the subroutine.

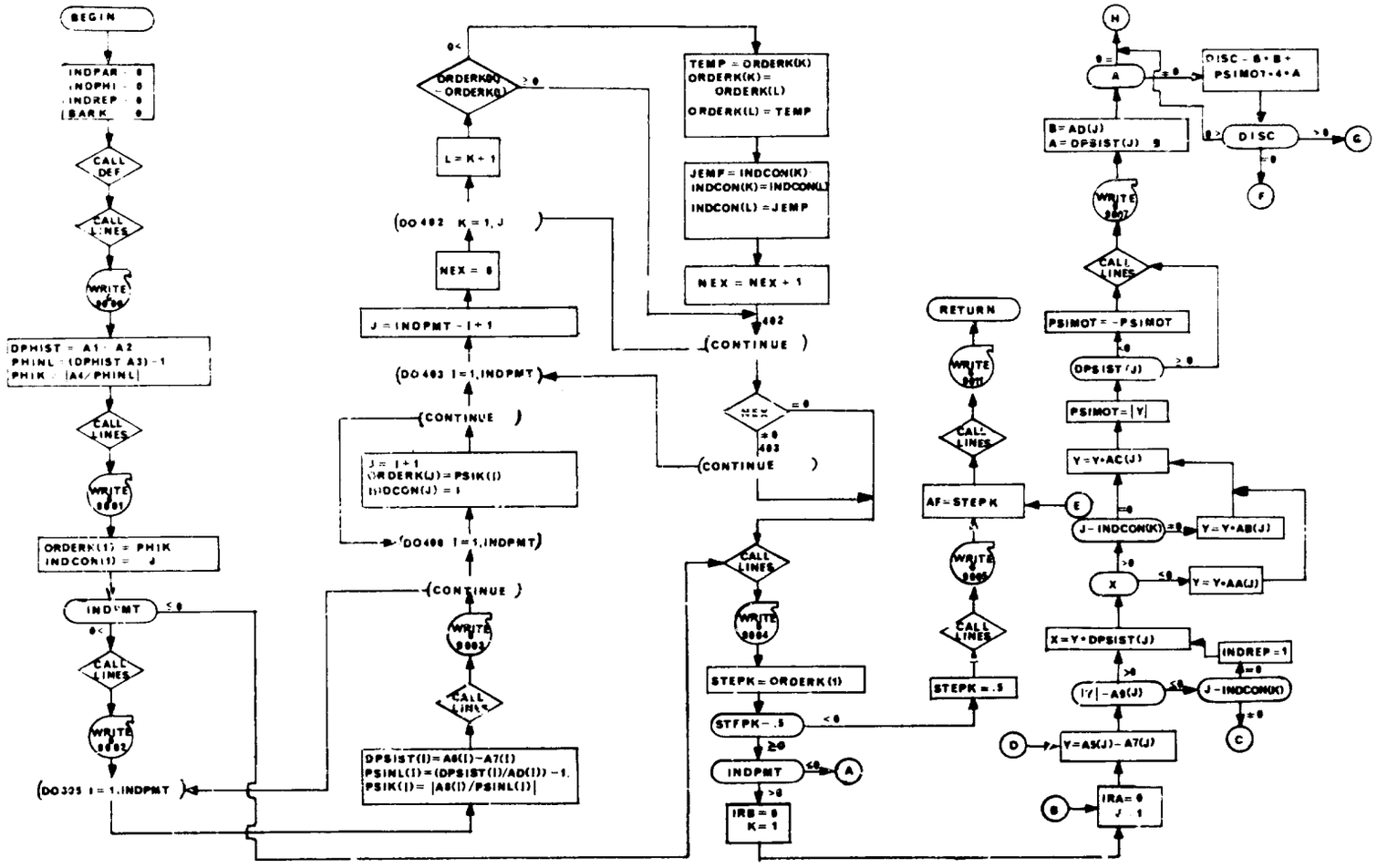
(**)
If $B^2 - 4AC < 0$, then no K_i'' and K_i''' is computed.

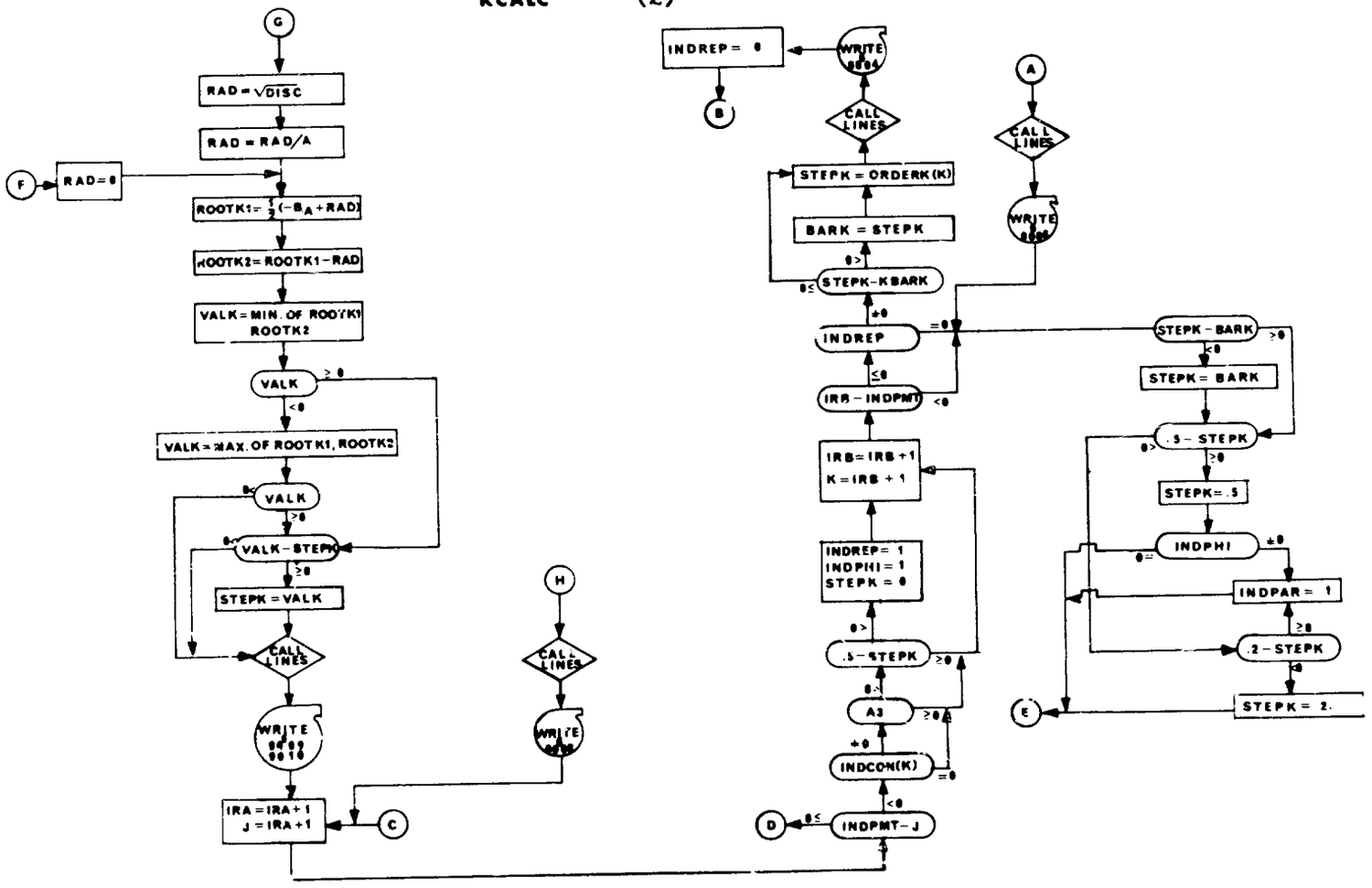
Remarks

KCALC is called by CTLS1.

KCALC calls DEF and LINES.

KCALC (1)


BEGIN
INDPAR = 0
INDPHI = 0
INDREP = 0
BARK = 0
CALL DEF
CALL LINES
WRITE 9000
DPHIST = A1 - A2
PHINL = (DPHIST A3) - 1
PHIK = |A4/PHINL|
CALL LINES
WRITE 9001
ORDERK(1) = PHIK
INDCON(1) = J
INDPMT
CALL LINES
WRITE 9002
(DO 305 I = 1,INDPMT)
DPSIST(I)=A6(I)-A7(I)
PSINL(I)=(DPSIST(I)/AD(I))-1.
PSIK(I)= |A8(I)/PSINL(I)|
CALL LINES
WRITE 9003
(CONTINUE)
(DO 400 I = 1,INDPMT)
J = I + 1
ORDERK(J) = PSIK(I)
INDCON(J) = I
(CONTINUE)
(DO 403 I = 1,INDPMT)
J = INDPMT - I + 1
NEX = 0
(DO 402 K = 1, J)
L = K + 1
ORDERK(K) - ORDERK(L)
TEMP = ORDERK(K)
ORDERK(K) = ORDERK(L)
ORDERK(L) = TEMP
JEMP = INDCON(K)
INDCON(K) = INDCON(L)
INDCON(L) = JEMP
NEX = NEX + 1
402
(CONTINUE)
NEX
403
(CONTINUE)
CALL LINES
WRITE 9004
STEPK = ORDERK(1)
STFPK - .5
INDPMT
IRB = 0
K = 1
STEPK = .5
CALL LINES
WRITE 9005
AF = STEPK
CALL LINES
WRITE 9011
RETURN
IRA = 0
J = 1
Y = A5(J) - A7(J)
|Y| - A9(J)
J - INDCON(K)
INDREP = 1
X = Y * DPSIST(J)
X
Y = Y * AA(J)
J - INDCON(K)
Y = Y * AB(J)
Y = Y * AC(J)
PSIMOT = |Y|
DPSIST(J)
PSIMOT = -PSIMOT
CALL LINES
WRITE 9007
B = AD(J)
A = DPSIST(J) - B
A
DISC - B * B + PSIMOT * 4 * A
DISC
H
G
F
E
D
C
B


G
RAD = √DISC
RAD = RAD/A
F
RAD = 0
ROOTK1 = ½(-B_A + RAD)
ROOTK2 = ROOTK1 - RAD
VALK = MIN. OF ROOTK1 ROOTK2
VALK
≥ 0
< 0
VALK = MAX. OF ROOTK1, ROOTK2
VALK
> 0
VALK - STEPK
≥ 0
STEPK = VALK
CALL LINES
WRITE
H
CALL LINES
WRITE
IRA = IRA + 1
J = IRA + 1
C
INDREP = 0
B
WRITE
CALL LINES
STEPK = ORDERK(K)
BARK = STEPK
> 0
STEPK - KBARK
≤ 0
+0
INDREP
= 0
≤ 0
IRB - INDPMT
< 0
IRB = IRB + 1
K = IRB + 1
INDREP = 1
INDPHI = 1
STEPK = 0
> 0
.5 - STEPK
≥ 0
> 0
A3
≥ 0
+0
INDCON(K)
= 0
< 0
D
≤ 0
INDPMT - J
A
CALL LINES
WRITE
STEPK - BARK
≥ 0
< 0
STEPK = BARK
> 0
.5 - STEPK
≥ 0
STEPK = .5
INDPHI
± 0
= 0
INDPAR = 1
≥ 0
.2 - STEPK
< 0
STEPK = 2.
E

8. INVERT - $I_{\psi\psi}$ Inversion

Purpose:

To invert the $I_{\psi\psi}$ matrix.

Usage:

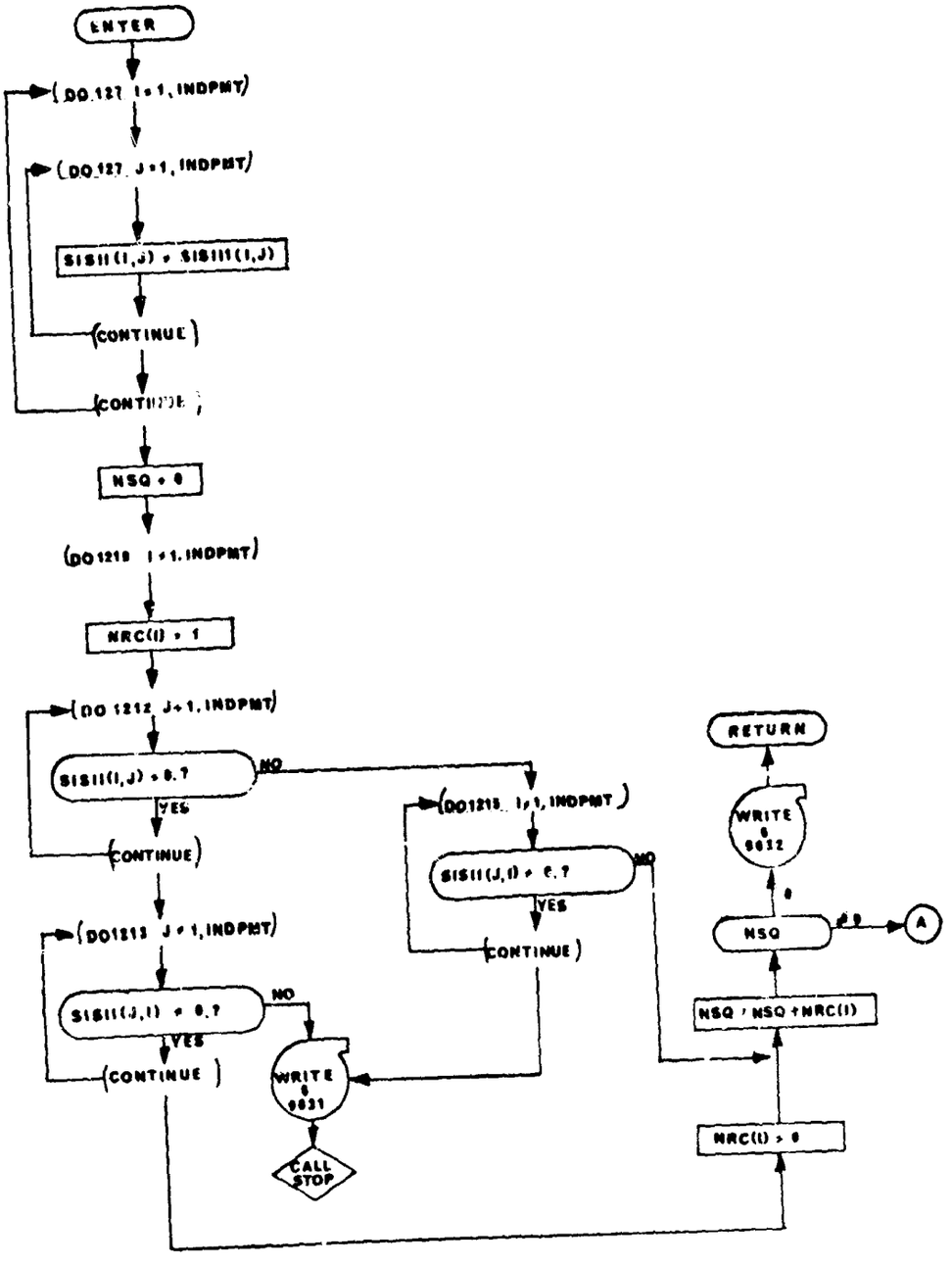
CALL INVERT

From theoretical considerations, $I_{\psi\psi}$ should be symmetric. If a column (row) of $I_{\psi\psi}$ is zero and the corresponding row (column) is non-zero, the program is terminated. If a column and corresponding row are both zero, they are eliminated from the matrix. The resulting matrix is inverted and any zero rows or columns are inserted into the inverse. $I_{\psi\psi}$ is post-multiplied by $I^{-1}_{\psi\psi}$ as a check.

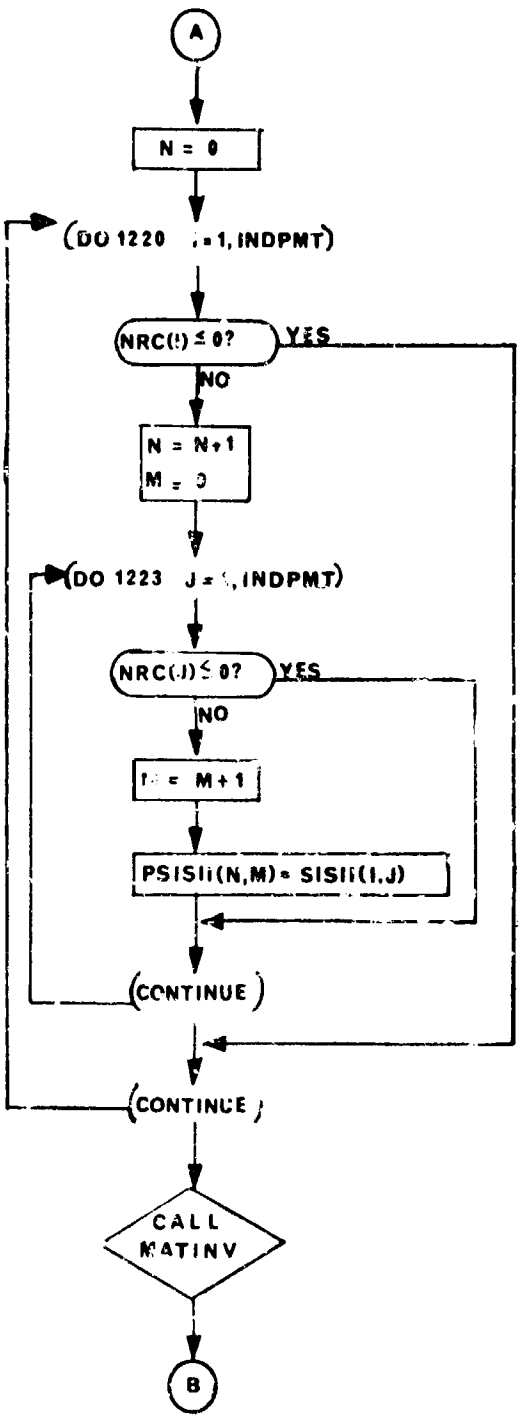
Remarks:

CTLS1 and UPDK call INVERT.
INVERT calls MATINV and STOP.

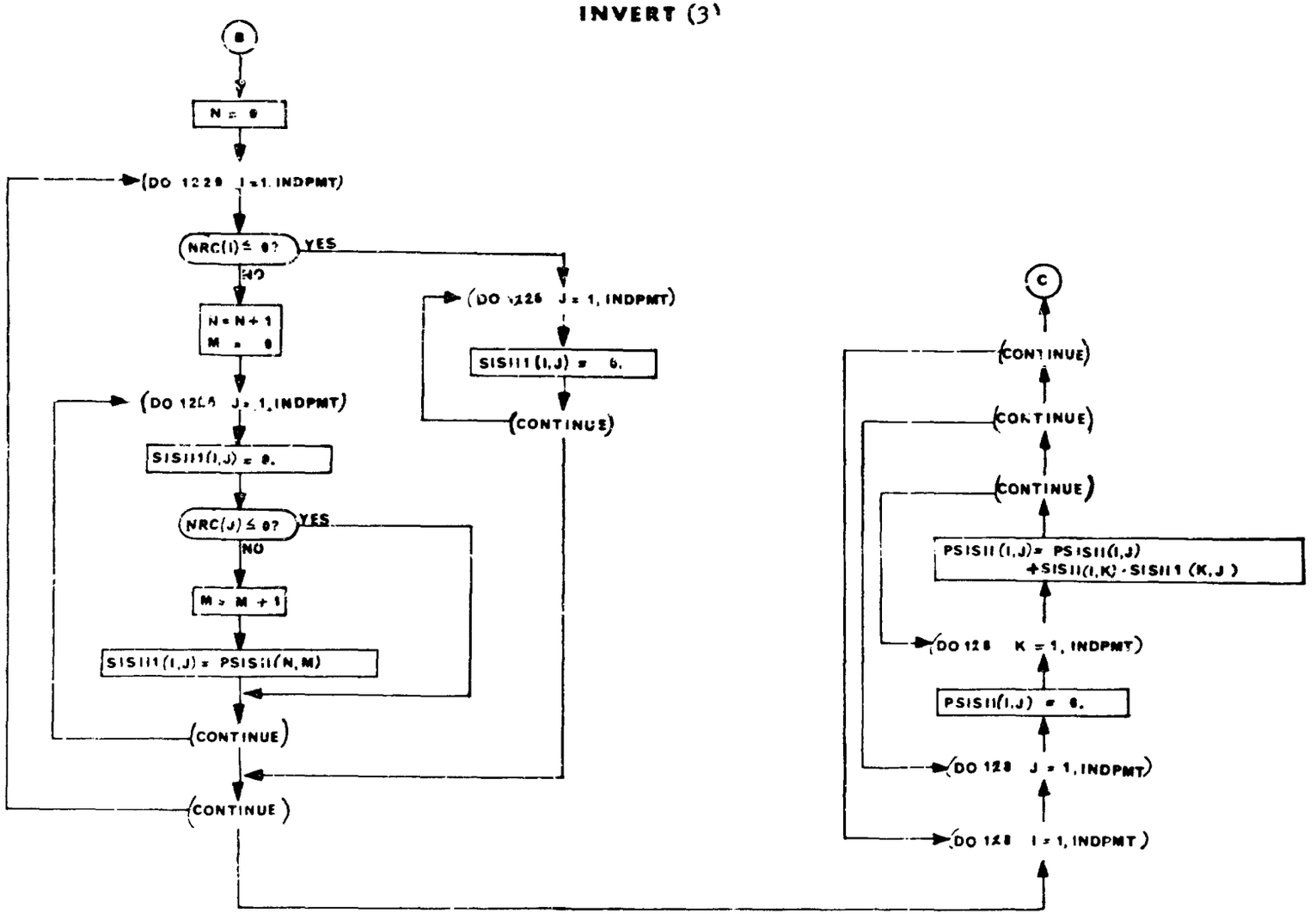
INVERT (1)


ENTER
(DO 127 I = 1, INDPMT)
(DO 127 J = 1, INDPMT)
SISII(I,J) = SISIII(I,J)
(CONTINUE)
(CONTINUE)
NSQ = 0
(DO 1218 I = 1, INDPMT)
NRC(I) = 1
(DO 1212 J = 1, INDPMT)
SISII(I,J) = 0.?
NO
YES
(CONTINUE)
(DO 1213 J = 1, INDPMT)
SISII(J,I) = 0.?
NO
YES
(CONTINUE)
WRITE 6 9031
CALL STOP
(DO 1215 I = 1, INDPMT)
SISII(J,I) = 0.?
NO
YES
(CONTINUE)
NSQ = NSQ + NRC(I)
NRC(I) = 0
NSQ
≠ 0
A
0
WRITE 6 9032
RETURN

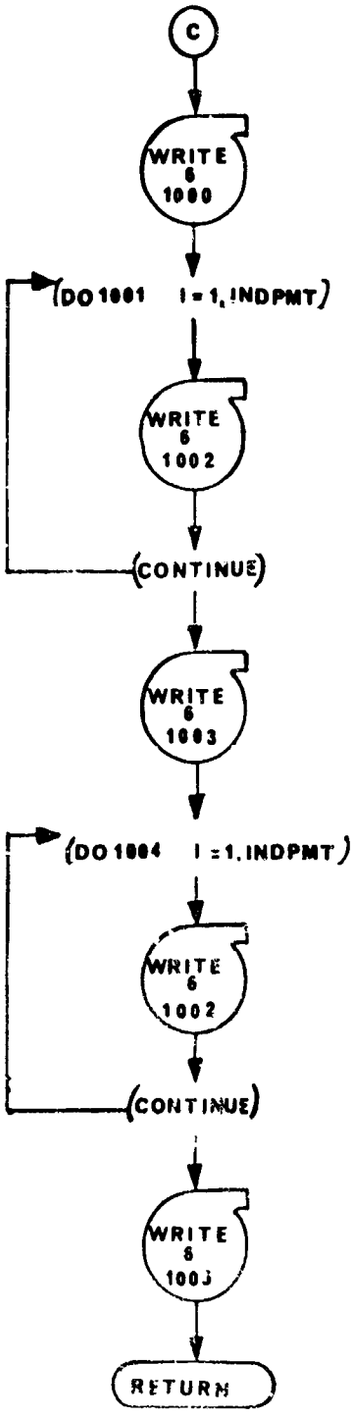
INVERT (2)


A
N = 0
(DO 1220 I=1, INDPMT)
NRC(I) ≤ 0?
YES
NO
N = N+1
M = 0
(DO 1223 J = 1, INDPMT)
NRC(J) ≤ 0?
YES
NO
M = M+1
PSISII(N,M) = SISII(I,J)
(CONTINUE)
(CONTINUE)
CALL MATINV
B

INVERT (3)


B
N = 0
(DO 1229 I=1, INDPMT)
NRC(I) ≤ 0?
YES
NO
N = N + 1
M = 0
(DO 1228 J = 1, INDPMT)
SISIII(I,J) = 0.
(CONTINUE)
(DO 1225 J = 1, INDPMT)
SISIII(I,J) = 0.
NRC(J) ≤ 0?
YES
NO
M = M + 1
SISIII(I,J) = PSISII(N,M)
(CONTINUE)
(CONTINUE)
C
(CONTINUE)
(CONTINUE)
(CONTINUE)
PSISII(I,J) = PSISII(I,J)
+SISII(I,K)·SISIII(K,J)
(DO 128 K = 1, INDPMT)
PSISII(I,J) = 0.
(DO 128 J = 1, INDPMT)
(DO 128 I = 1, INDPMT)

INVERT (4)


C
WRITE 6 1000
DO 1001 I = 1, INDPMT
WRITE 6 1002
CONTINUE
WRITE 6 1003
DO 1004 I = 1, INDPMT
WRITE 6 1002
CONTINUE
WRITE 6 1003
RETURN

9. DECIDE - Driver for Decision Routines

Purpose

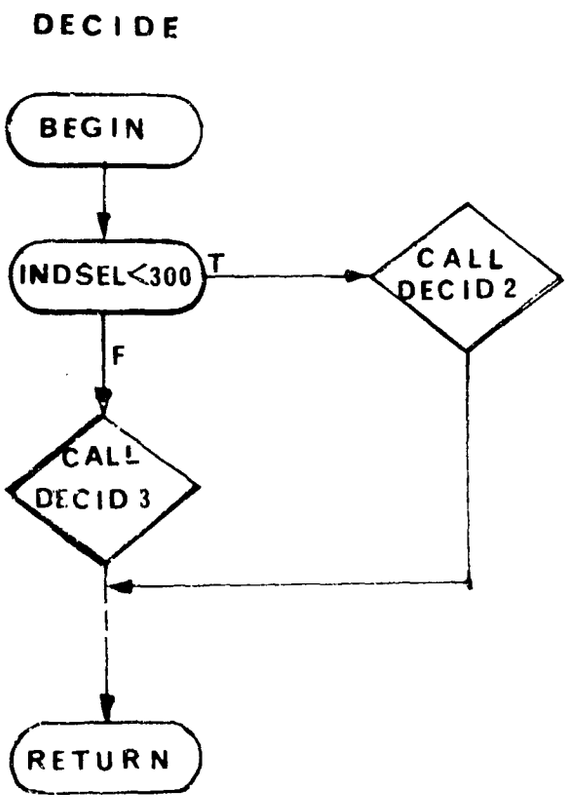
To allow the user to select the decision subprogram which will be used with CTLS2 in the optimization program.

Usage

CALL DECIDE (IENTRY)

IENTRY = indicator which determines what decision must be made by the decision subprogram.

INDSEL is of the form i0j. The decision subprogram is determined by i. If i = 3, DECID3 is used. If i = 2, DECID2 is called. DECID2 is currently a dummy routine. INDSEL is nominally 101. j determines which set of differential equations will be used.



10. OFFSW - Display Drop Routine

Purpose:

To drop the on-line display from the console.

Usage:

Type "DROP" at console.

Remark:

This is a special-purpose ASCENT routine. No flow chart is included.

11. TLUU - Two Dimensional Table Look-Up Routine

Purpose

Given an argument X, to compute Y = f(X) from a table of X and Y values by linear interpolation.

Method

Same as for TLU.

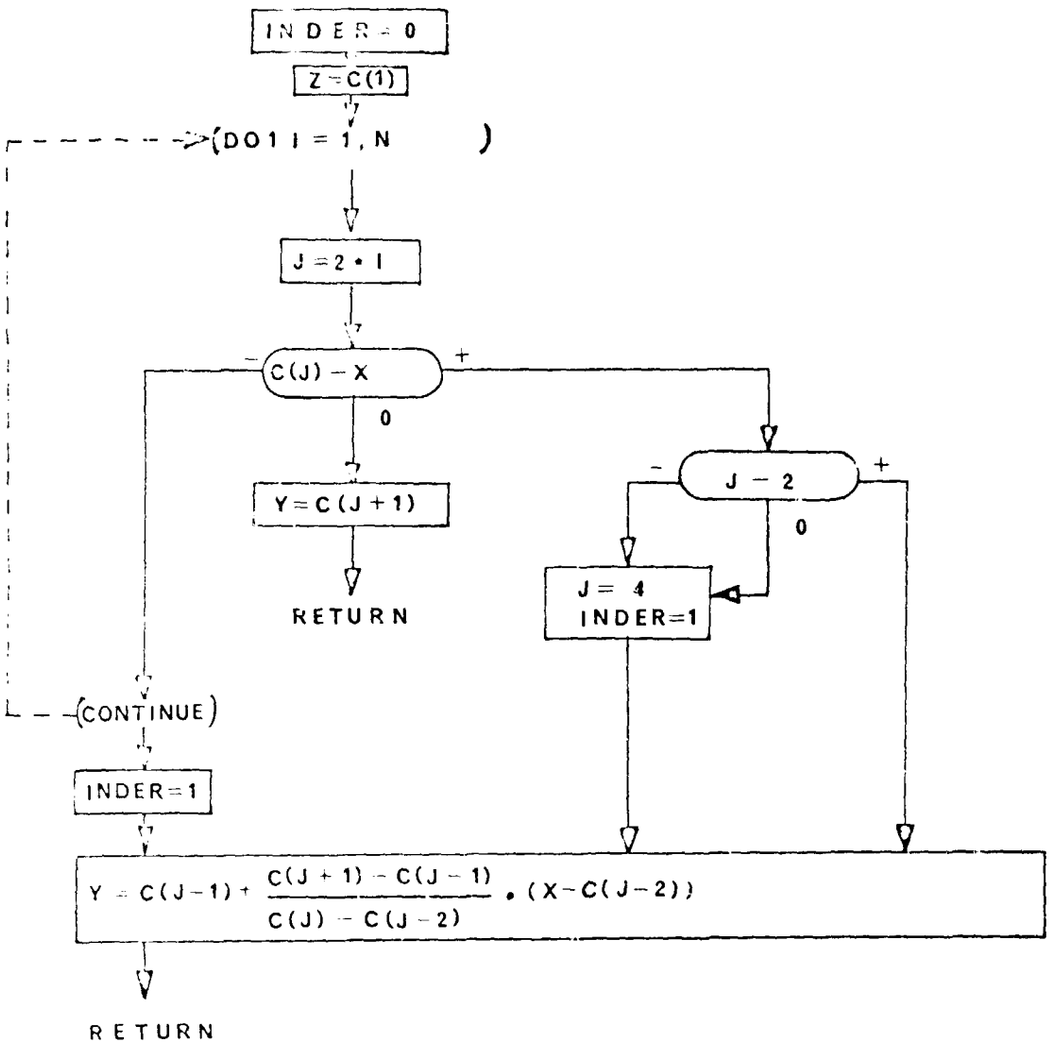
Usage

Same as for TLU.

Remarks

This subroutine is the same as TLU except the table is not given as a common subscript.

TLUU


INDER = 0
Z = C(1)
DO 1 I = 1, N
J = 2 • I
C(J) − X
−
+
0
Y = C(J + 1)
RETURN
J − 2
−
+
0
J = 4
INDER=1
CONTINUE
INDER=1
Y = C(J−1) + C(J + 1) − C(J − 1) / C(J) − C(J − 2) • (X − C(J − 2))
RETURN

12. UNBLOCK - Unblocking Routine for λ's

Purpose:

To read and decode the blocked tape (ILTAP) that has been prepared in chain REV.

Method:

The values for the Δ change in the mode shape of α's have been stored in a large array to decrease the number of times an I/O device is selected.

Usage:

Entry is made to the routine by the statement:

```
CALL UNBLOCK (PHOLGW, TIMES, NDUMMY, INDPMT, MCONT, PSOLGW,
ALPHC, W2, ILTAP)
```

where:

PHOLGW = Mode shape change values

TIMES = Time from beginning stage

NDUMMY = Control word

MCOUNT = Number of control variables

INDPMT = Number of end and point constraints

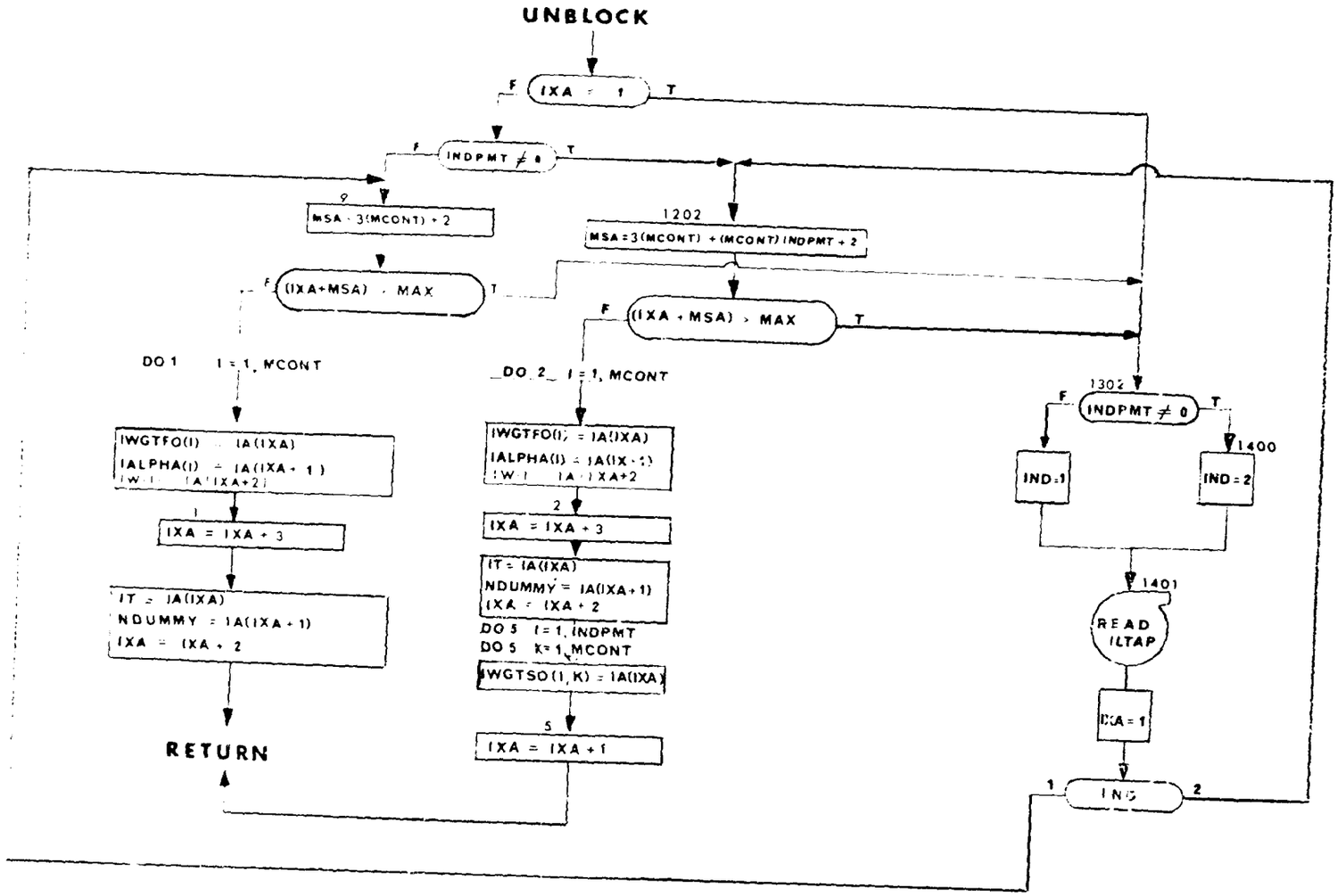
PSOLGW = Mode shape change values

ALPHC = α's from previous valid step.

W2 = Weighting Matrix elements

ILTAP = I/O device

No initialization is necessary outside this routine and no output occurs from this routine. Only the normal I/O FORTRAN routines are called.


UNBLOCK
IXA = 1
F
T
INDPMT ≠ 0
9
MSA = 3(MCONT) + 2
1202
MSA = 3(MCONT) + (MCONT)(INDPMT) + 2
(IXA+MSA) > MAX
(IXA + MSA) > MAX
DO 1 I = 1, MCONT
DO 2 I = 1, MCONT
IWGTFO(I) = IA(IXA)
IALPHA(I) = IA(IXA + 1)
IW = IA(IXA+2)
1
IXA = IXA + 3
IT = IA(IXA)
NDUMMY = IA(IXA + 1)
IXA = IXA + 2
RETURN
IWGTFO(I) = IA(IXA)
IALPHA(I) = IA(IX + 1)
IW = IA(IXA+2)
2
IXA = IXA + 3
IT = IA(IXA)
NDUMMY = IA(IXA+1)
IXA = IXA + 2
DO 5 I = 1, INDPMT
DO 5 K = 1, MCONT
IWGTSO(I, K) = IA(IXA)
5
IXA = IXA + 1
1302
INDPMT ≠ 0
1400
IND = 1
IND = 2
1401
READ
ILTAP
IXA = 1
1
ING
2

13. SUMOLA - Linear Combination Routine for λ's

Purpose:

To compute linear combinations of sensitivities.

Usage:

CALL SUMOLA (PHISEN, PSISEN, C, NPSI, NVAR)

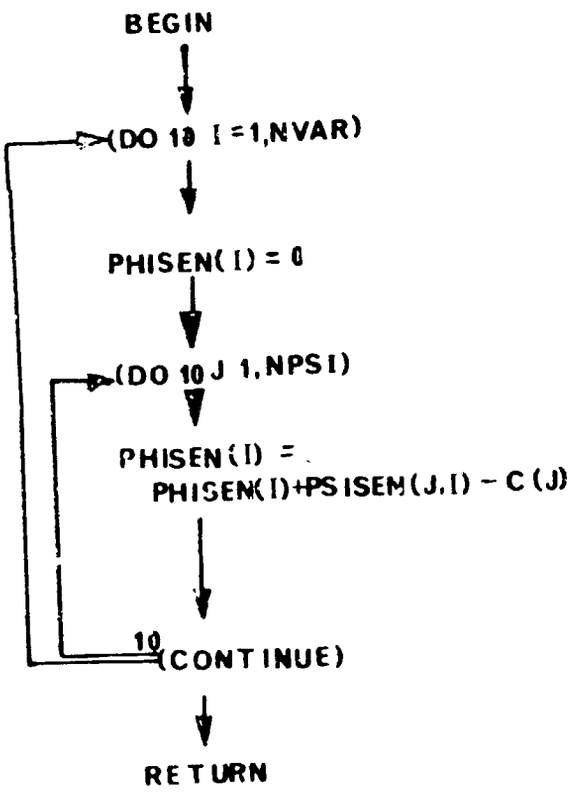
PHISEN	=	the linear combination to be computed.
PSISEN	=	the component sensitivities which are to be combined.
C	=	weights to be applied to component sensitivities.
NPSI	=	number of components
NVAR	=	length of each component array.

$$PHISEN_i = \sum_{j=1}^{NPSI} PSISEN_{j,i} \cdot C_j, \quad i = 1, NVAR$$

Remarks:

This routine is used with the "CTLS2" control system.

SUMOLA


BEGIN
(DO 10 I =1,NVAR)
PHISEN(I) = 0
(DO 10 J 1,NPSI)
PHISEN(I) =
PHISEN(I)+PSISEM(J,I) - C(J)
10 (CONTINUE)
RETURN

14. PLOT - Point Plot Subroutine

A point plot subroutine written by Mary Huff of McDonnell Automation Center in December 1965.

Purpose:

To plot a graph of one or more curves from given sets of rectangular coordinates using a minimum of storage.

Method:

To avoid overlapping characters, the printer limits the characters which can be printed on the plot to 109 per line and 51 per column. To plot a curve, it is therefore necessary for the routine to divide each page of the plot into 109 intervals in the X direction and 51 intervals in the Y direction. The width of the X intervals, called ΔX, is then $\frac{PXMAX - PXMIN}{108}$ where PXMAX and PXMIN are the maximum and minimum X values which can be plotted on that page. If a search of the X coordinates of the curves to be plotted then shows an X value falling within the range of the n-th interval, that is $PXMIN - \frac{\Delta X}{2} + (n-1)\Delta X \leq X < PXMIN + \frac{\Delta X}{2} + (n-1)\Delta X$, then a point will be plotted in the column representing the n-th interval. The routine allows for a multipage option in which the Y axis remains constant but the X axis varies from XMIN to XMAX over a range of several pages. The number of pages will equal $\frac{XRANGE}{PRANGE}$ where XRANGE is the range of X values over all the curves or XMAX - XMIN and PRANGE is the range of X values which could be plotted on a single page or $\Delta X \cdot 108$. The maximum X value on one page will become the minimum X value on the succeeding page until XMAX is reached. ΔX, which is needed to determine PXMAX for each page, must be determined before the plot is attempted. It may be supplied to the routine by the user or left to the routine to calculate before plotting the curves. If ΔX is left to the subroutine to calculate, a search of the curves to be plotted will find ΔX to be the minimum distance between any two points on any curve in the X direction. If the multipage option was specified, then ΔX is accepted as calculated by the routine or as specified by the user, and the number of pages will be $\frac{XRANGE}{PRANGE}$. If the single page option was specified, and $\Delta X = \frac{XMAX-XMIN}{108}$, then ΔX is accepted as specified or calculated, otherwise ΔX is set to $\frac{XMAX-XMIN}{108}$ to allow the full range of X values to fit on exactly a single page. In order to plot a graph of less than one page, the multipage option must be specified, even if ΔX is specified.

The log of y is plotted if the semilog option is used, and the log of X and Y if the log-log option is used. If the value for which the log is to be plotted is negative, the log of the positive value will be plotted, but an N will replace the specified character for that point on the curve. The

log of zero will not be attempted, but the zero will be plotted with the character N.

Any number of curves may be plotted on the same graph. The X coordinates of each curve must be stored as one column of the two dimensional X array and Y coordinates of the curve as the corresponding column of the Y array. The points need not be ordered, but the X and Y coordinates of a point must be at corresponding positions in their respective columns. The curves may be distinguished from each other by designating different plotting symbols. Any Hollerith Character may be used; however, the subroutine uses a plus sign for the border, a minus and an I for axes, and an N for attempting the log of a negative or zero number. If a point is common to more than one curve, the symbol of the last curve plotted will be shown.

The subroutine will eject a page and print the title specified by the user at the top of each page of the plot. The border option will outline each page of the graph with + signs. The axis option will print the X and Y axes (plot the curves Y = 0 and X = 0). The X scale is printed at 2" intervals and the Y scale at 1" intervals.

Usage:

The routine is entered by the FORTRAN statement
CALL PLOT(X,Y,M,A,IC,B,MPLL,XDELT,TITLE,IDM)

where

X is the name of a 2 dimensional array containing the X coordinates of all the curves to be plotted. The X coordinates of the n-th curve, for example, are stored from X(1,n) to X(m,n) where m is the number of points in the curve.

Y has the same dimensions as X and contains the Y coordinates of the curves.

M and A are the names of one dimensional arrays set up by the user. M(n) is the number of points in the n-th curve, whose first point is at X (1, n) and Y(1,n). A(N) is the character---left-adjusted in a 4 byte word---to be used in plotting the n-th curve.

IC is the number of curves to be plotted and must be less than or equal to the second dimension specified for the X and Y arrays in their DIMENSION statement.

B=1 no border, no axis
2 border, no axis
3 no border, X axis only
4 border, X axis only
5 no border, Y axis only
6 border, Y axis only
7 no border, both axes
8 border, both axes

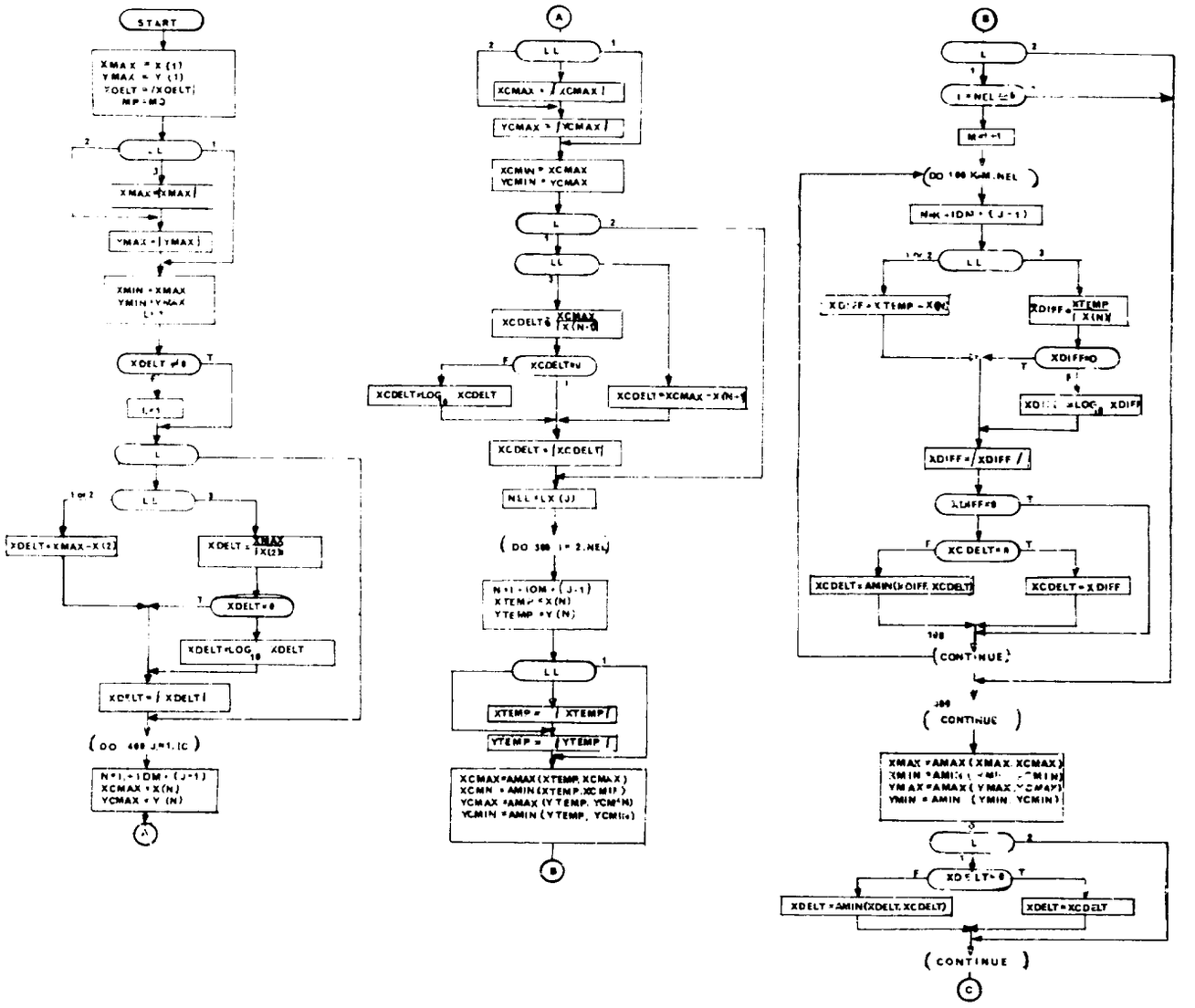
MP = = 0 single page plot desired
1 multiple page or fraction of page plot desired

LL = 1 plot given points
2 semilog (plot logs of Y coordinates)
3 log-log (plot logs of X and Y coordinates)

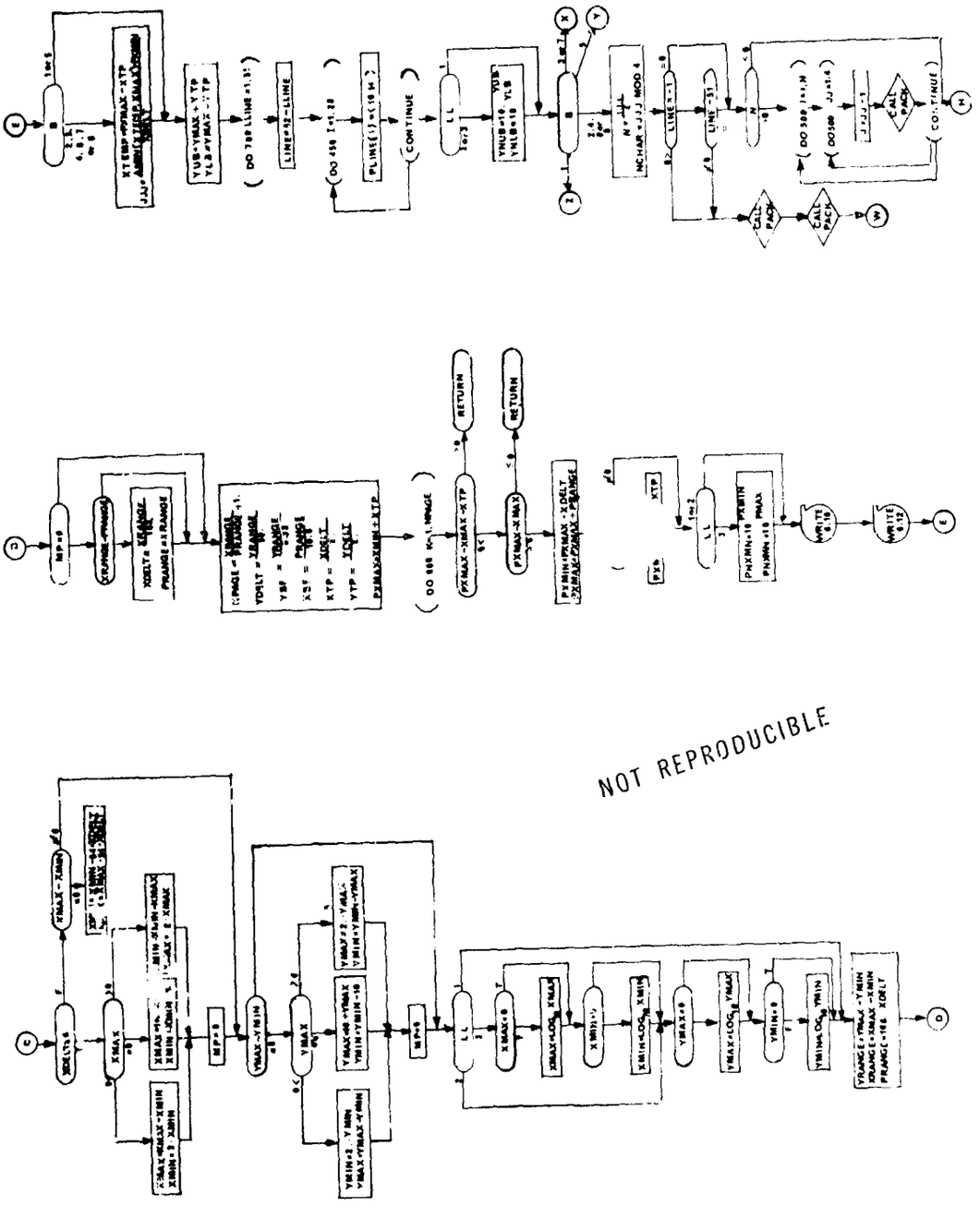
XDELT = 0 indicates ΔX is to be calculated
Otherwise specify ΔX in floating point

TITLE is the name of the array in which the title to head each page is stored. Only one line may be printed. The array must be at least 30 four byte words in length. Any words in excess of 30 will not be printed.

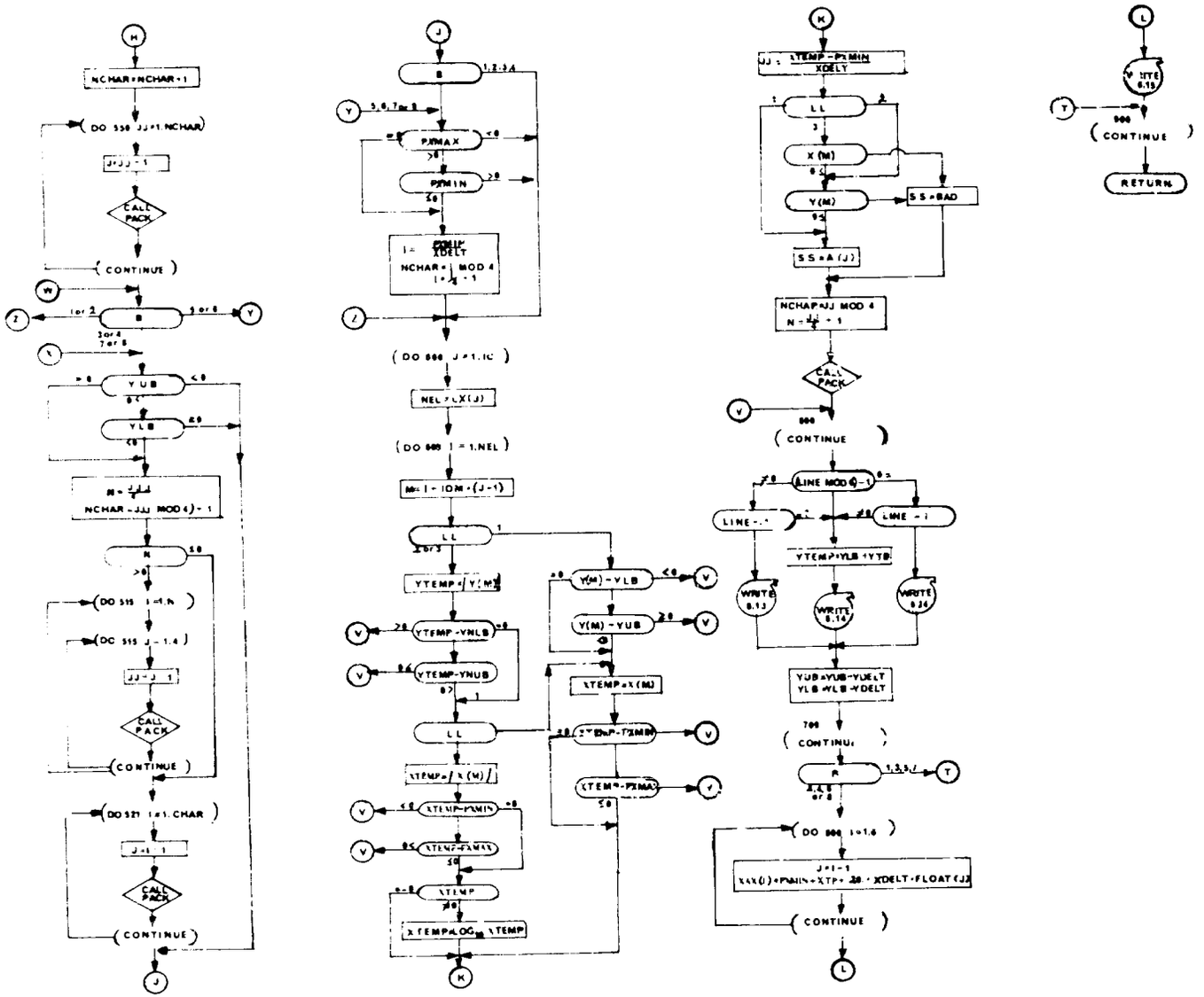
IDM is the first dimension specified for the X and Y arrays in their DIMENSION statement. It must be at least equal to the number of points in the curve which has the greatest number of points of the curves to be plotted.


START
XMAX = X(1)
YMAX = Y(1)
XDELT = |XDELT|
NP = NO
XMAX = |XMAX|
YMAX = |YMAX|
XMIN = XMAX
YMIN = YMAX
XDELT ≠ 0
XDELT = XMAX - X(2)
XDELT = XMAX / X(2)
XDELT = 0
XDELT = LOG10 XDELT
XDELT = |XDELT|
DO 400 J=1,IC
N = I + IOM * (J-1)
XCMAX = X(N)
YCMAX = Y(N)
A
XCMAX = |XCMAX|
YCMAX = |YCMAX|
XCMIN = XCMAX
YCMIN = YCMAX
XDELT = XCMAX / X(N-1)
XDELT = XCMAX - X(N-1)
XDELT = LOG10 XDELT
XDELT = |XDELT|
NLL = LX(J)
DO 300 I = 2,NLL
N = I + IOM * (J-1)
XTEMP = X(N)
YTEMP = Y(N)
XTEMP = |XTEMP|
YTEMP = |YTEMP|
XCMAX = AMAX (XTEMP, XCMAX)
XCMIN = AMIN (XTEMP, XCMIN)
YCMAX = AMAX (YTEMP, YCMAX)
YCMIN = AMIN (YTEMP, YCMIN)
B
I = NEL
DO 100 K=M, NEL
XDIFF = XTEMP - X(N)
XDIFF = XTEMP / X(N)
XDIFF = 0
XDIFF = LOG10 XDIFF
XDIFF = |XDIFF|
XDELT = 0
XDELT = AMIN (XDIFF, XDELT)
XDELT = XDIFF
100
CONTINUE
300
CONTINUE
XMAX = AMAX (XMAX, XCMAX)
XMIN = AMIN (XMIN, XCMIN)
YMAX = AMAX (YMAX, YCMAX)
YMIN = AMIN (YMIN, YCMIN)
XDELT = AMIN (XDELT, XDELT)
XDELT = XDELT
CONTINUE
C

PLOT (2)



NOT REPRODUCIBLE


NCHAR=NCHAR+1
DO 550 JJ=1,NCHAR
JJ=JJ-1
CALL PACK
CONTINUE
YUB
YLB
DO 515 I=1,N
DO 515 J=1,4
JJ=J-1
CALL PACK
CONTINUE
DO 521 I=1,CHAR
J=I-1
CALL PACK
CONTINUE
DO 888 J=1,IC
NEL=LX(J)
DO 888 I=1,NEL
LL
YTEMP=Y(M)
YTEMP-YNLB
YTEMP-YNUB
XTEMP=X(M)
XTEMP-PXMIN
XTEMP-PXMAX
XTEMP
XTEMP=LOG10 XTEMP
PXMAX
PXMIN
Y(M)-YLB
Y(M)-YUB
XTEMP-PXMIN
XTEMP-PXMAX
JJ=XTEMP-PXMIN/XDELY
X(M)
Y(M)
SS=BAD
SS=A(J)
CALL PACK
CONTINUE
LINE MOD 6-1
LINE=1
LINE=LINE+1
YTEMP=YLB+YTB
WRITE 8,13
WRITE 8,14
WRITE 8,24
YUB=YUB-YUELT
YLB=YLB-YDELT
CONTINUE
DO 800 I=1,6
XXX(J)=PXMIN+XTP+.28*XDELT*FLOAT(J)
CONTINUE
WRITE 8,15
CONTINUE
RETURN

15. MATINV - Matrix Inversion Routine

Purpose:

FORTRAN subroutine solves the matrix equation AX = B, where A is a square coefficient matrix and B is a matrix of constant vectors. A^{-1} is also obtained; indeed, inversion may be the sole aim in a particular usage. Finally, the determinant of A is available.

Method:

Jordan's method is used to reduce a matrix A to the identity matrix I through a succession of elementary transformations: $\ell_n \ell_{n-1} \ldots \ell_c$, A = I. If these transformations are simultaneously applied to I and to a matrix B of constant vectors, the result is A^{-1} and X where AX = B.

Usage:

Entrance is made via the FORTRAN statement in the calling program:

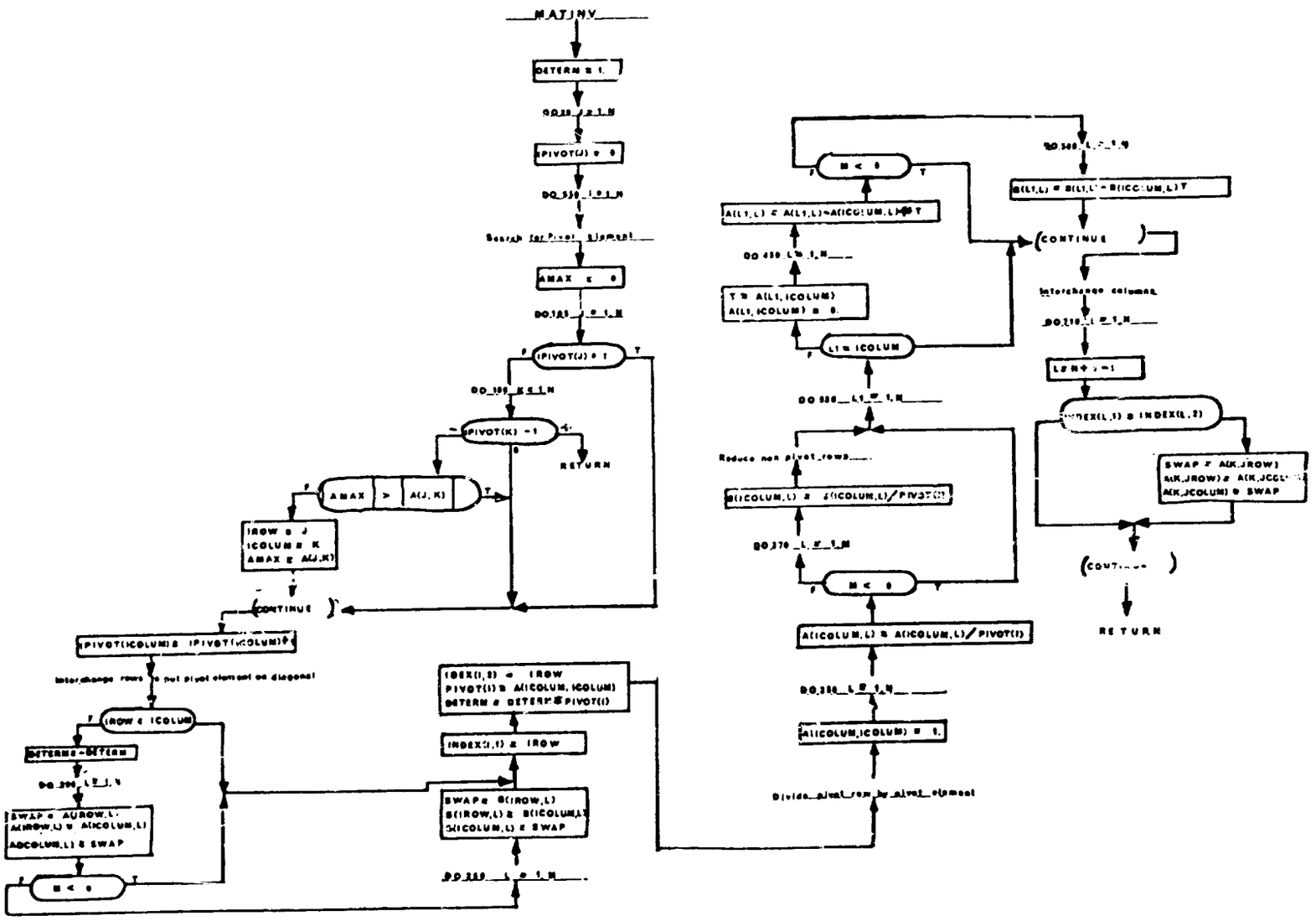
CALL MATINV (A, N, B, M, DETERM)

where
1) N is the order of A; N ≥ 1.
2) M is the number of column vectors in B.
3) DETERM is the location in which the determinant is to be placed.

Suitable variable names may replace the dummy variables listed above.

At the return to the calling program, A^{-1} is stored at A and X at B.

M = 0 or negative signals that the routine is to be used solely for inversion; note, however, that in the CALL statement an entry corresponding to B must still be present.


MATINV
DETERM = 1.
DO 20 J = 1,N
IPIVOT(J) = 0
DO 550 I = 1,N
Search for Pivot element
AMAX = 0
DO 105 J = 1,N
IPIVOT(J) = 1
DO 100 K = 1,N
IPIVOT(K) = 1
RETURN
AMAX > A(J,K)
IROW = J
ICOLUM = K
AMAX = A(J,K)
CONTINUE
IPIVOT(ICOLUM) = IPIVOT(ICOLUM)+1
Interchange rows to put pivot element on diagonal
IROW = ICOLUM
DETERM = -DETERM
DO 200 L = 1,N
SWAP = A(IROW,L)
A(IROW,L) = A(ICOLUM,L)
A(ICOLUM,L) = SWAP
M < 0
INDEX(I,1) = IROW
PIVOT(I) = A(ICOLUM,ICOLUM)
DETERM = DETERM*PIVOT(I)
INDEX(I,2) = ICOLUM
SWAP = B(IROW,L)
B(IROW,L) = B(ICOLUM,L)
B(ICOLUM,L) = SWAP
DO 350 L = 1,N
A(ICOLUM,ICOLUM) = 1.
Divide pivot row by pivot element
A(ICOLUM,L) = A(ICOLUM,L)/PIVOT(I)
DO 550 L1 = 1,N
L1 = ICOLUM
T = A(L1,ICOLUM)
A(L1,ICOLUM) = 0.
DO 450 L = 1,N
A(L1,L) = A(L1,L)-A(ICOLUM,L)*T
Reduce non-pivot rows
B(ICOLUM,L) = B(ICOLUM,L)/PIVOT(I)
DO 470 L = 1,N
M < 0
DO 300 L = 1,N
B(L1,L) = B(L1,L)-B(ICOLUM,L)*T
CONTINUE
Interchange columns
DO 710 I = 1,N
L = N+1-I
INDEX(L,1) = INDEX(L,2)
SWAP = A(K,JROW)
A(K,JROW) = A(K,JCOLUM)
A(K,JCOLUM) = SWAP
CONTINUE
RETURN

16. DECID3 - Step Size Routine for Control Systems

Purpose

To determine when to accept a pass as a valid step, wher to compute partials, the step size to use on the next trial and to invert the I_{yy} matrix and make other decisions to determine the $\delta\alpha$ mode shape.

Usage

CALL DECID3 (IENTRY, D2,D3,D4,D5,D6,D7,D8,D9,DPHI,DB,DK,DE,DF,DG, INDAMP,INDBAL,ITBAL)

CTLS2, DECID3, UPDK, and SUMOLA form a control system referred to in the formulation as "CTLS2". To select this control system input INDSEL of 103 (301).

Tasks are performed at each entry point as follows:

IENTRY = 1 is called when 20 or more passes have occurred on a cycle. The program is terminated.

IENTRY = 2 is called for nominal initialization.

IENTRY = 3 is called when a trial missed cutoff and computed partials. The step size is noted as a bad step. Partials are turned off and the next trial step size is set to one-half the current step size.

IENTRY = 4 is called after a successful trial is made with partials on. A parabolic prediction is performed. If a gain was made on the trial and the predicted percentage gain for another trial is less than 25% for another trial, the step is accepted. If a gain was made but the predicted percentage gain for another trial is 25% the step is not accepted immediately. Tests to limit the number of passes are applied if the step is not immediately accepted or no gain was made and trial may be accepted if a gain was made or a new step size may be determined for another trial with partials taken.

IENTRY = 5 is called for output of restart cards. None are output.

IENTRY = 6 is called when a trial missed cutoff and did not compute partials. The same action as IENTRY = 3 is taken.

IENTRY = 7 is called after a successful trial is made with partials off. A parabolic prediction is performed. If a gain was made on the trial and the predicted percentage gain for another trial is less than 5% partials will be taken the next trial. Tests to limit the number of passes are applied. These tests may turn on partials if a good step is available but no gain was made in the previous trial. A new step size for another trial is determined.

IENTRY = 8 is called for initialization each cycle.

IENTRY = 9 is called to make decisions for the $\delta\alpha$ mode shape. If step size is too small the program is terminated.

IENTRY = 10 is called to set INDCON for CTLS2 after a trial. INDCON is set to zero.

IENTRY = 11 is called after DALCAL is computed following the reverse integration. The only action is a RETURN.

IENTRY = 12 is called when DENOM is not positive. The program is terminated.

IENTRY = 13 is called to set INDCON, invert the $I_{\gamma\gamma}$ matrix, and update parameters for $\delta\alpha$ calculation after the reverse integration. INDCON is set to zero and UPDK is called to calculate parameters for $\delta\alpha$ calculation and to invert the $I_{\gamma\gamma}$ matrix.

After each trial trajectory, a point is generated on the graph of change in payoff ($\Delta\varphi$) vs. step size, unless cutoff was missed. A point with positive $\Delta\varphi$ is called a good point and the step size is called a good step. A point with negative $\Delta\varphi$ is called a bad point and the step size is called a bad step. The step size on a trial trajectory which misses cutoff is also called a bad step.

From the theoretical considerations the graph of $\Delta\varphi$ vs. step size must rise to a peak from the origin and fall off. Our objective is to arrive at a good point with step size as close to this peak as possible. We attempt to do this by a parabolic search procedure.

After each trial trajectory which has missed cutoff, partials are turned off and another trial will be made with step size 1/2 the last trial step.

After each trial trajectory which has not missed cutoff, a parabolic fit is made to the last trial point and any previous good points. If no previous good points are available, the fit is made to the origin, the last trial point, and the slope at the origin. If only one previous good point is available the fit is made to the origin, the previous good point, and the last trial point. If two previous good points are available, one lying on either side of the last trial point, the fit is made to those three points. If two such good points are not available, but two good points have been found which lie on one side of the last trial point, the fit is made to those three points.

After the fit has been made, a preliminary step size to use for another trial pass is determined. If the parabola is not concave downward the preliminary step will be the smaller of 5 times the last trial step and 9/10 the smallest previous bad step. If the parabola is concave downward, the step size is set to the larger of 1/10 the last trial step and the smallest of 5 times the last trial step size coordinate of the maximum point on the parabola, and 9/10 the smallest previous bad step.

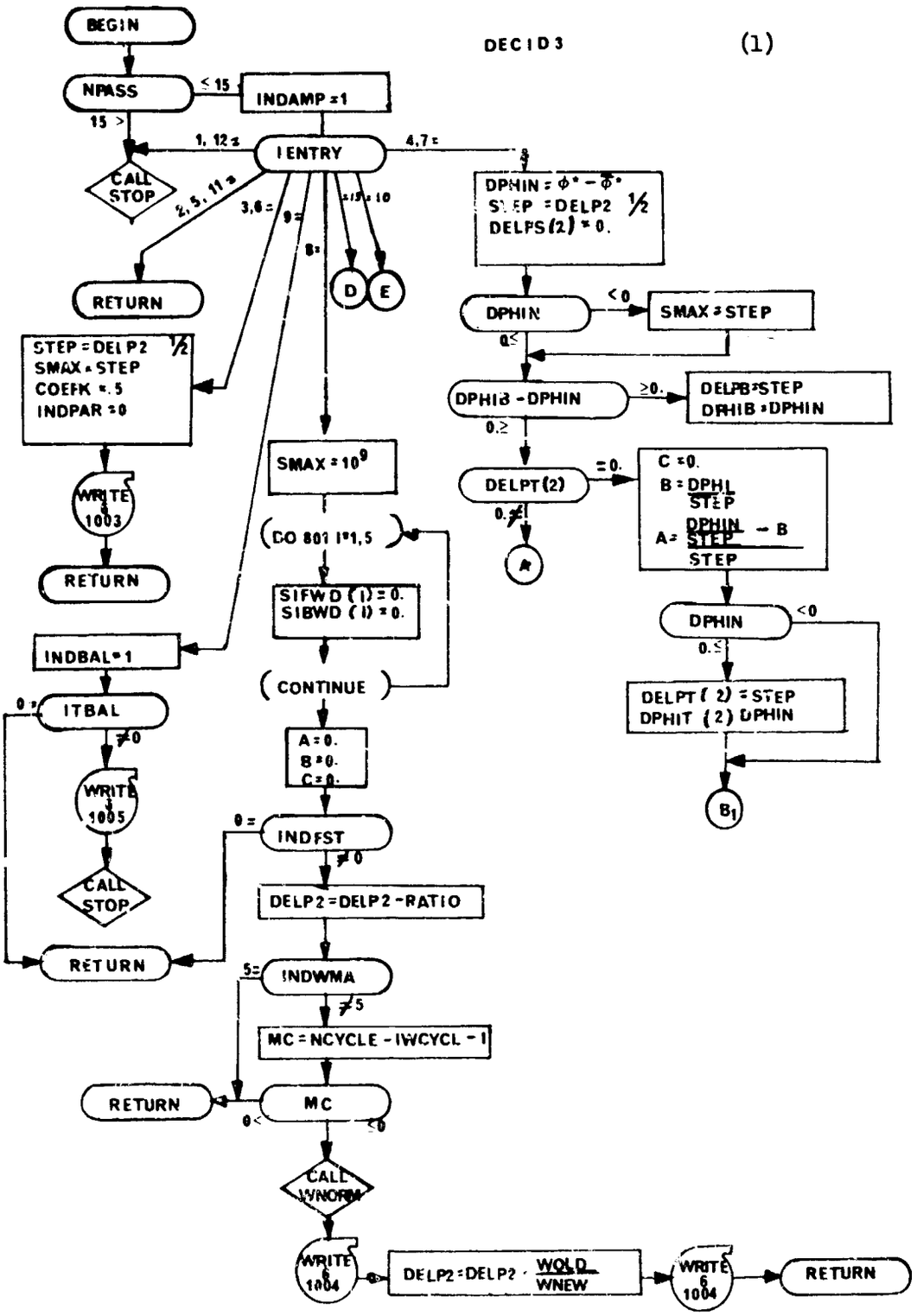
If less than 10 passes have been made and the last trial step had positive $\Delta\varphi$, the parabolic fit and the preliminary step size are used to find the predicted change in payoff from the parabola. If partials were taken and the predicted gain is less than 25% the last trial is accepted as a valid step. If partials were not taken and the predicted gain is less than 5% partials are turned on and another trial trajectory is made using the preliminary step size.

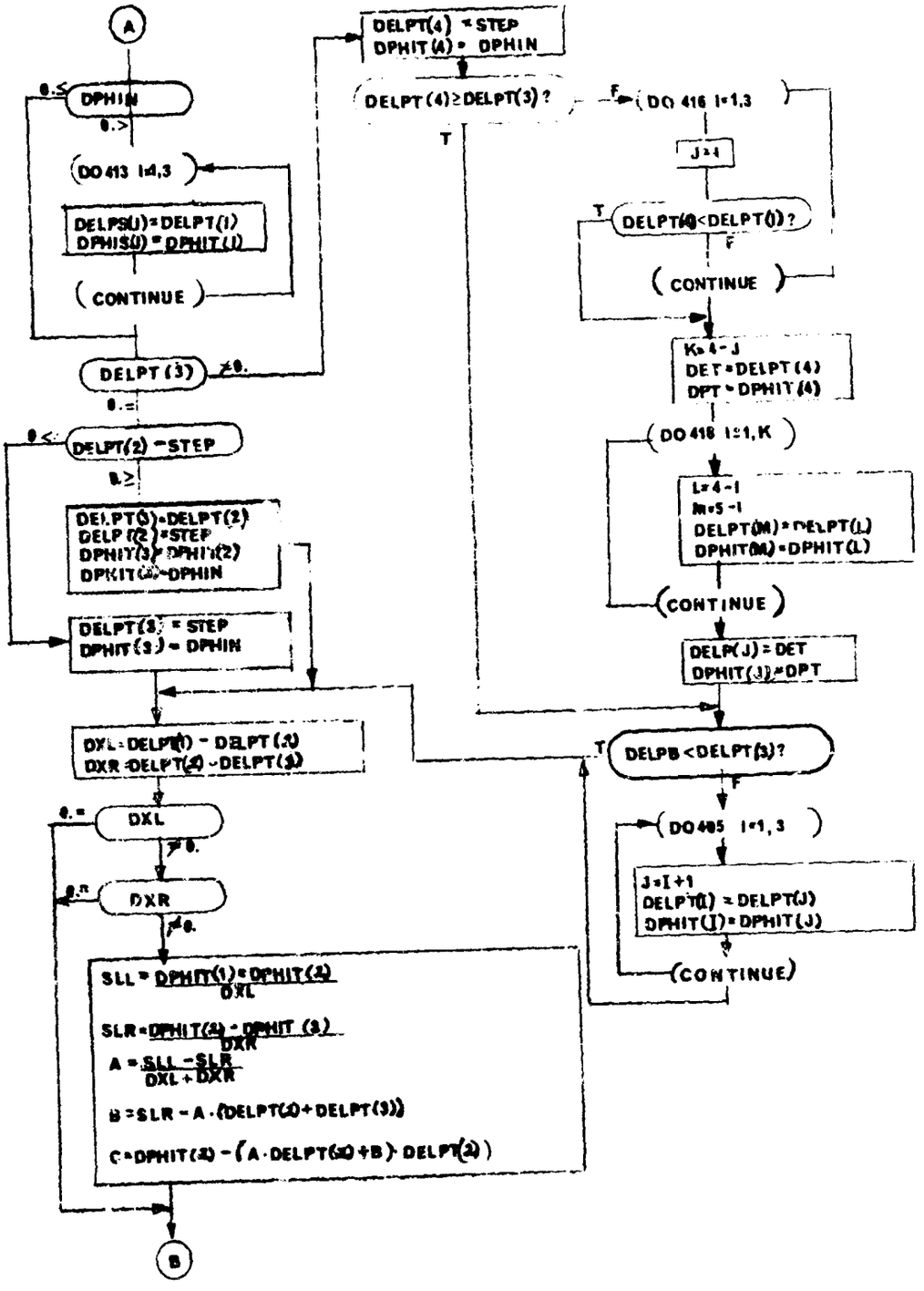
If 10 passes have been made and the peak has not been found the last step will be taken if it is a good step and partials were taken. If it was a bad step, the program will go to the best good step size so far, take another pass with partials on and accept that pass. If no trial so far has produced a good step, the step size is halved and a trial made. Halving will continue until a good step is found or 15 passes have been made and the program terminates. If a good step is found one more pass at that step size will be made with partials on and that pass will be accepted.

Remarks

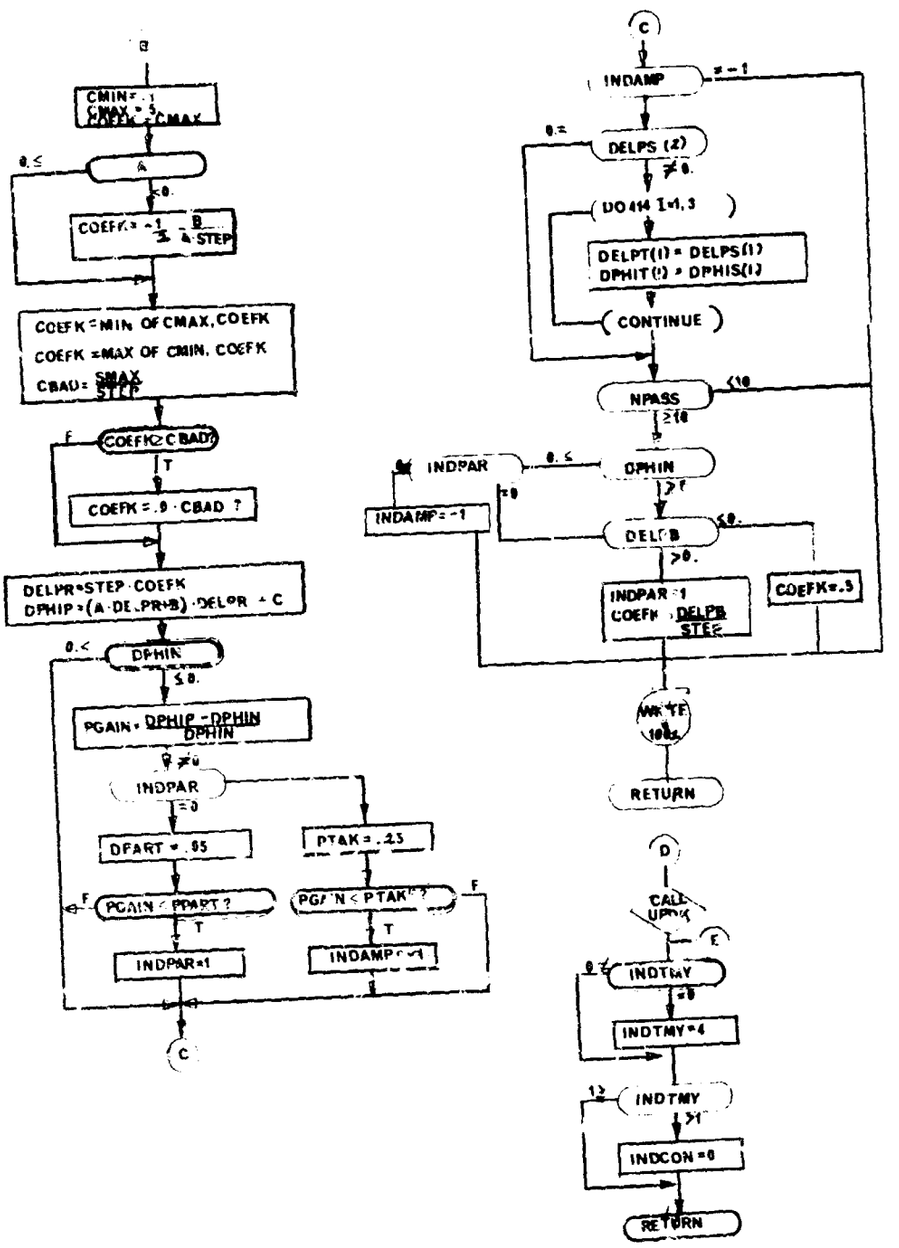
DECID3 is called only by DECIDE which in turn is called only by CTLS2.

DECID3 calls UPDK, WNORM, and STOP.


DECID3 (1)
BEGIN
NPASS
≤ 15
INDAMP=1
15 >
1, 12=
ENTRY
4,7=
CALL STOP
2, 5, 11=
3,6=
9=
8=
=13=10
D
E
RETURN
STEP=DELP2 1/2
SMAX=STEP
COEFK=.5
INDPAR=0
WRITE 6 1003
RETURN
INDBAL=1
0=
ITBAL
≠0
WRITE 6 1005
CALL STOP
RETURN
SMAX=10^9
DO 801 I=1,5
SIFWD(I)=0.
SIBWD(I)=0.
CONTINUE
A=0.
B=0.
C=0.
0=
INDFST
≠0
DELP2=DELP2-RATIO
5=
INDWMA
≠5
MC=NCYCLE-IWCYCL-1
RETURN
MC
0<
≤0
CALL WNORM
WRITE 6 1004
DELP2=DELP2 · WOLD/WNEW
WRITE 6 1004
RETURN
DPHIN=φ*−φ*
STEP=DELP2 1/2
DELPS(2)=0.
DPHIN
<0
SMAX=STEP
0≤
DPHIB-DPHIN
≥0.
DELPB=STEP
DPHIB=DPHIN
0.≤
DELPT(2)
=0.
0.≠
A
C=0.
B=DPHIL/STEP
A=DPHIN/STEP − B
DPHIN
<0
0.≤
DELPT(2)=STEP
DPHIT(2)=DPHIN
B1


A
DPHIN
0.<
0.>
DO 413 I=1,3
DELPS(I)=DELPT(I)
DPHIS(I)=DPHIT(I)
CONTINUE
DELPT(3) ≠0.
0.=
DELPT(2) =STEP
0.≥
DELPT(3)=DELPT(2)
DELPT(2)=STEP
DPHIT(3)=DPHIT(2)
DPHIT(2)=DPHIN
DELPT(3) = STEP
DPHIT(3) = DPHIN
DXL=DELPT(1) - DELPT(2)
DXR=DELPT(2) -DELPT(3)
0.= DXL
≠0.
0.= DXR
≠0.
SLL = DPHIT(1)-DPHIT(2) / DXL
SLR = DPHIT(2) - DPHIT(3) / DXR
A = (SLL - SLR) / (DXL + DXR)
B = SLR - A·(DELPT(2)+DELPT(3))
C = DPHIT(2) - (A·DELPT(2)+B)·DELPT(2)
B
DELPT(4) = STEP
DPHIT(4) = DPHIN
DELPT(4)≥DELPT(3)?
T
F
DO 416 I=1,3
J=I
DELPT(4)<DELPT(I)?
T
F
CONTINUE
K=4-J
DET=DELPT(4)
DPT=DPHIT(4)
DO 418 I=1,K
L=4-I
M=5-I
DELPT(M)=DELPT(L)
DPHIT(M)=DPHIT(L)
CONTINUE
DELP(J)=DET
DPHIT(J)=DPT
DELPB<DELPT(3)?
T
F
DO 485 I=1,3
J=I+1
DELPT(I)=DELPT(J)
DPHIT(I)=DPHIT(J)
CONTINUE

DECID3 (3)



B
CMIN = .1
CMAX = .5
COEFK = CMAX
A
COEFK = -1/2 · B/(A·STEP)
COEFK = MIN OF CMAX, COEFK
COEFK = MAX OF CMIN, COEFK
CBAD = SMAX/STEP
COEFK ≥ CBAD?
COEFK = .9 · CBAD ?
DELPR = STEP · COEFK
DPHIP = (A · DELPR + B) · DELPR + C
DPHIN
PGAIN = (DPHIP - DPHIN)/DPHIN
INDPAR
DPART = .05
PTAK = .25
PGAIN < PPART ?
PGAIN < PTAK ?
INDPAR = 1
INDAMP = -1
C
INDAMP
DELPS (2)
DO 414 I=1,3
DELPT(I) = DELPS(I)
DPHIT(I) = DPHIS(I)
CONTINUE
NPASS
DPHIN
INDPAR
INDAMP = -1
DELPB
INDPAR = 1
COEFK = DELPB/STEP
COEFK = .5
WRITE
RETURN
D
CALL UPDK
E
INDTMY
INDTMY = 4
INDTMY
INDCON = 0
RETURN

17. DECID2 - Arbitrary Decision Routine

This is a dummy routine included so the user may write his own decision subprogram to be used with CTLS2 in the optimization program.

Such a routine must adhere to the entry point pattern established by CTLS2 and make those decisions assigned to it by CTLS2.

18. PACK - Integer Word Conversion Routine

Purpose:

To insert the character to be plotted into the correct position in an integer type word.

Method:

PACK uses DECODE to break a word into 4 words, each containing one character. The (IPOS+1)th word is then set equal to CHAR and ENCODE is used to place the 4 characters back into the word.

Usage:

CALL PACK(WORD, CHAR, IPOS)

where

WORD = the word in which the character is to be placed

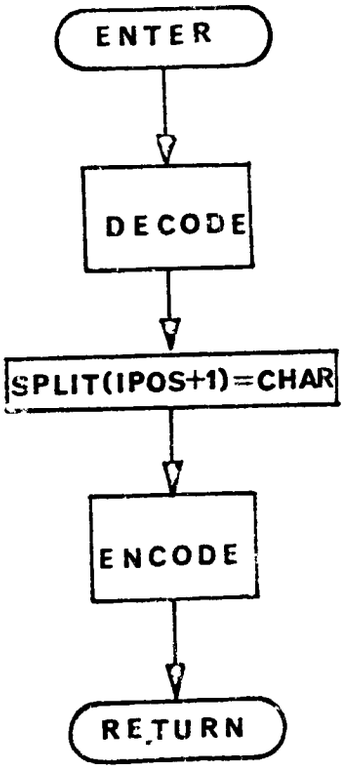
CHAR = the Hollerith character to be inserted into WORD

IPOS = one less than the position in WORD into which CHAR will be inserted

Subroutines Called:

System functions ENCODE and DECODE.

PACK



19. WNORM - Weighting Matrix Norm Calculation

Purpose:

To compute the current weighting matrix norm when using OPTION 5.

Usage:

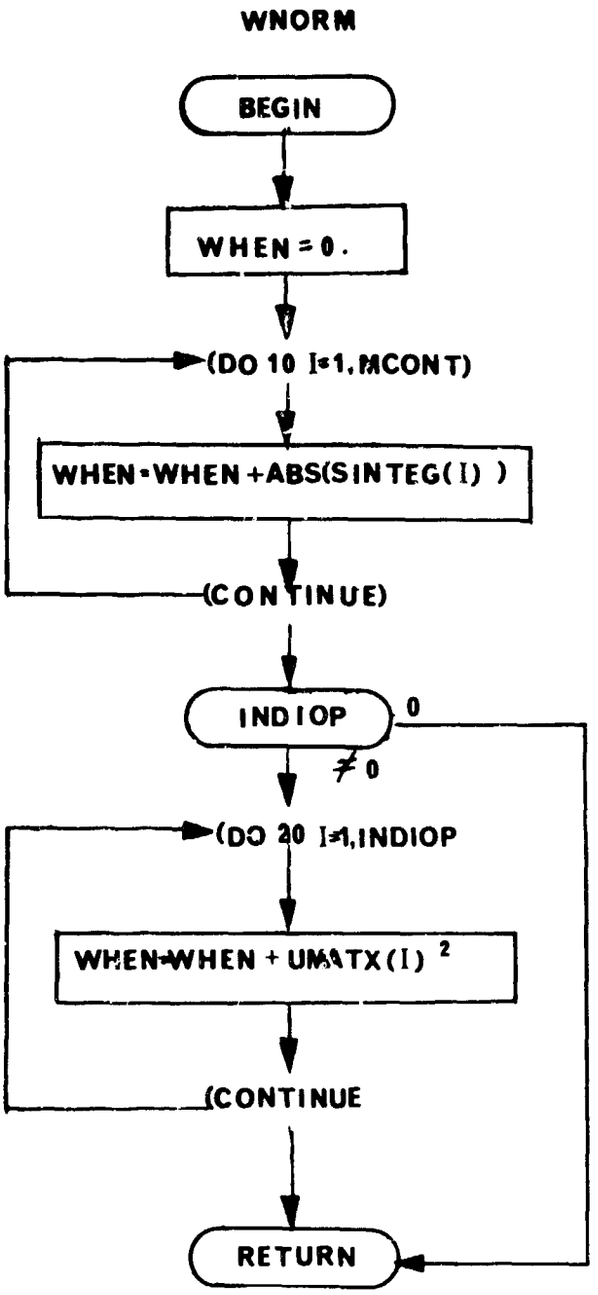
CALL WNORM (WNEW)

$$WNEW = \left\{ \sum_{i=1}^{MCONT} \left| \int_{T}^{o} (W_{i,i}^{-1})^2 \, dt \right| + \sum_{j=1}^{INDIOP} (U_j^{-1})^2 \right\}^{1/2}$$

MCONT = number of control variables

INDIOP = number of initial control variables

T = terminal trajectory time


WNORM
BEGIN
WHEN = 0.
(DO 10 I=1, MCONT)
WHEN=WHEN+ABS(SINTEG(I))
(CONTINUE)
INDIOP
0
≠0
(DO 20 I=1, INDIOP
WHEN=WHEN + UMATX(I) 2
(CONTINUE
RETURN

20. UPDK - Convergence Control Routine

Purpose

To determine when and how to tighten constraint belts, and to perform the tightening; to test for convergence of the optimization process and terminate the program if convergence is obtained, and to recompute $I_{\varphi\varphi}$, φ^*, and the coefficients for the augmented payoff function.

Usage

CALL UPDK

CTLS2, DECID3, UPDK, and SUMOLA are the routines which implement the control system referred to as "CTLS2" in the formulation.

This routine makes those decisions which have most effect on the convergence of the optimization process under "CTLS2".

The philosophy of the routine is given in the formulation manual. The procedure by which SIERR is reduced is referred to as belt tightening. The logic of the routine is given in the accompanying explanatory flowchart.

A tool for investigating the convergence effects of decisions other than those made by the normal logic of this routine is provided. This tool is referred to as "Analyst Control." Using this option, all decisions on tightening and phase selection are superseded by data inputs which completely pre-determine when and how tightening is done and what phase will be entered. This is not intended for normal use.

The normal logic is referred to as "Program Control". Certain parameters used by this option may be modified by data. Modification by data is not recommended.

Parameters for UPDK and their effect on tightening and phase selections are listed under the two options. These parameters may be set to values other than the nominal by input data.

See routine FENAL for a description of the augmented payoff function used by this routine.

Analyst Control

Parameter	Nominal	Effect
INDINT	0	0 selects program control 1 selects analyst control
		For Analyst Control Input A Value of 1
INTCYC	All 1000	When NCYCLE = $\lvert$INTCYC(J)$\rvert$ where $1 \leq J \leq 8$, belts will be tightened. INTCYC is an array of eight values.
INTPHZ	All 0	INTPHZ(J) is the phase to be entered after belt tightening when NCYCLE = $\lvert$INTCYC(J)$\rvert$ where $1 \leq J \leq 8$. INTPHZ is an array of eight values.
ITIT	0	If ITIT > 0 and NCYCLE = $\lvert$INTCYC(J)$\rvert$, where $1 \leq J \leq 8$, and INTCYC(J) > 0, INTCYC(J) will be replaced by INTCYC(J) + ITIT after belt tightening.
TTIT	0	Basic belt tightening factor. When belts are to be tightened, all are multiplied by TTIT. If all constraint errors are still within their respective belts, the belts are again multiplied by TTIT. This continues until at least one error is outside its belt. Then all belts less than 1/2 their respective tolerances are increased to 1/2 those tolerances. After belt tightening, if TTIT is 0., it is set to .1. TTIT may be modified before tightening by TINT and TDUN.
TDUN	0.	If NCYCLE = $\lvert$INTCYC(J)$\rvert$ where $1 \leq J \leq 8$, and INTCYC(J) < 0, TTIT is multiplied by TDUN before belt tightening. Total tightening modification factor.
TINT	All 1.	Partial tightening modification factor. If NCYCLE = $\lvert$INTCYC(J)$\rvert$ where $1 \leq J \leq 8$, and INTCYC(J) > 0, TTIT is multiplied by TINT(J) before belt tightening. TINT is an array of eight values. TINT may be modified by TMOD.
TMOD	0.	If NCYCLE = $\lvert$INTCYC(J)$\rvert$ where $1 \leq J \leq 8$ and INTCYC(J) > 0 and ITIT > 0, TINT(J) is replaced by TINT(J)-TMOD after belt tightening.

Program Control

Parameter	Nominal	Effect
INDINT	0	0 selects program control 1 selects analyst control
INTEX	3	The ratios: $\frac{\varphi^{*} - \bar{\varphi}^{*}_{INTEX}}{\bar{\varphi}^{*}_{INTEX}}$ and $\frac{\bar{\varphi}^{*}_{i} - \bar{\varphi}^{*}_{INTEX}}{\bar{\varphi}^{*}_{INTEX}}$, i = 1, INTEX-1 are checked against TTOL for tightening of belts.
TTOL	.005	If all ratios specified by INTEX are less than or equal to TTOL and belt tightening has not been delayed by INTSTY, a check will be made for convergence and belts will be tightened.
INTSTY	3	Belt tightening is delayed by the INTSTY number of cycles after leaving phase 0 or effecting a belt tightening.
TTIT	0	Same as for Analyst Control.
ITIT	0	Same as for Analyst Control.
INTDUN	3	Total tightening cycle indicator. When ITIT = INTDUN, TTIT is multiplied by TDUN before tightening.
TDUN	0.	Total tightening modification factor for TTIT.
TINT	All 1.	Same as for Analyst Control.
INTPHZ	All 0	Same as for Analyst Control.

Analyst and Program Control

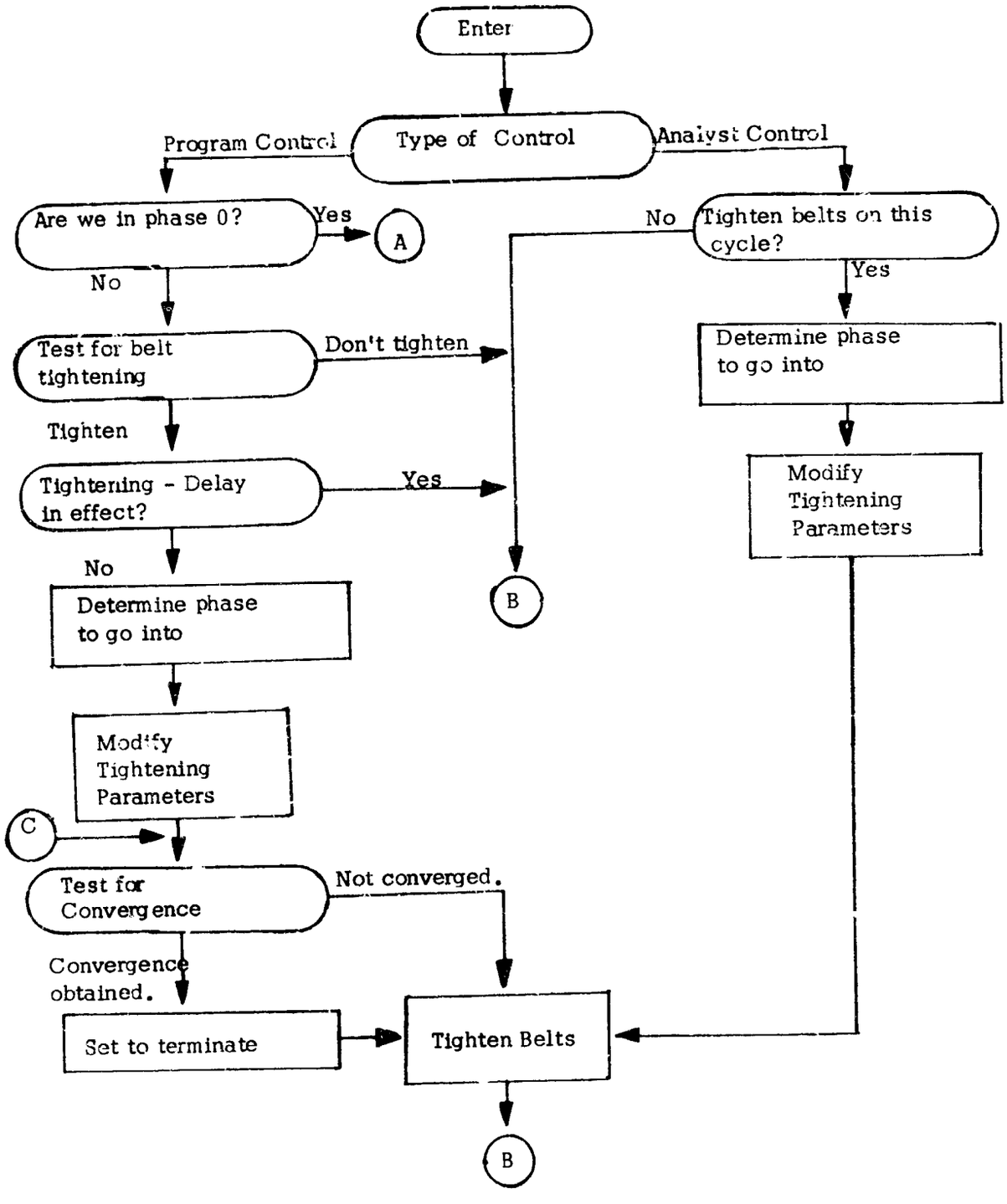
Parameter	Nominal	Effect
ENDCON	All blank	Identifiers for payoff and constraints. ENDCON is an array of ten words. The first is the payoff function, the rest are constraint functions. MAXIM or MINIM must be "PNALTY".
SIBAR	All 0.	Desired constraint values. SIBAR is an array of ten values. The first is a dummy.
SITOL	All 0.	Allowable constraint error. SITOL is an array of ten values. The first is a dummy.
SIERR	All .01	Belt sizes for constraints. SIERR is an array of ten values. The first is a dummy. The rest are belt values for the constraints.
INDTMY	4	Initial phase for CTLS2 control system. 4 is phase 0 3 is phase 1.
P	1.	Exponent used in determining phase 1 mode parameters.
RATIO	2.	Step size multiplicative change factor between successive cycles.

Remarks

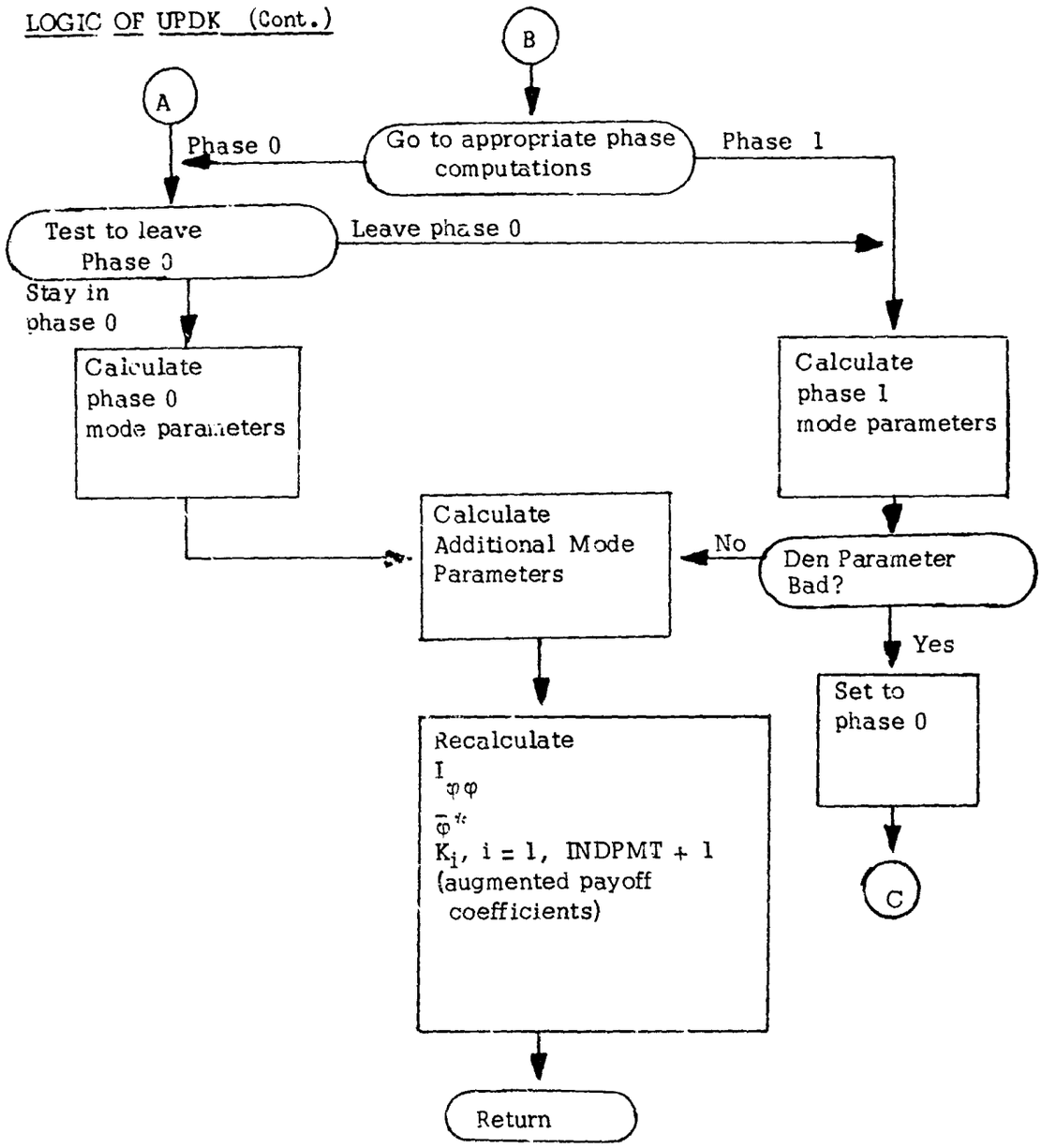
UPDK is called by DECID3.

UPDK calls INVERT and STOP.

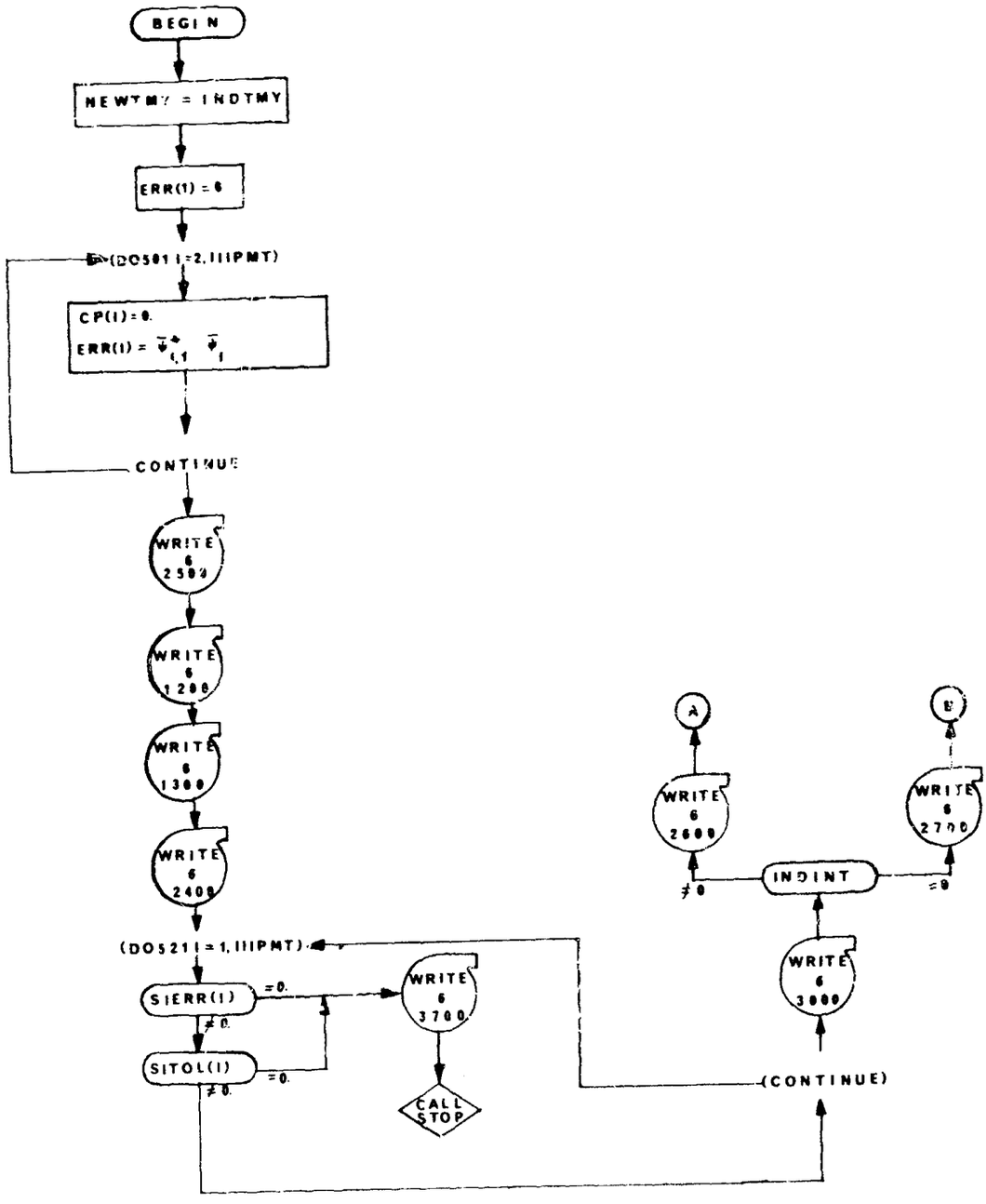
LOGIC OF UPDK

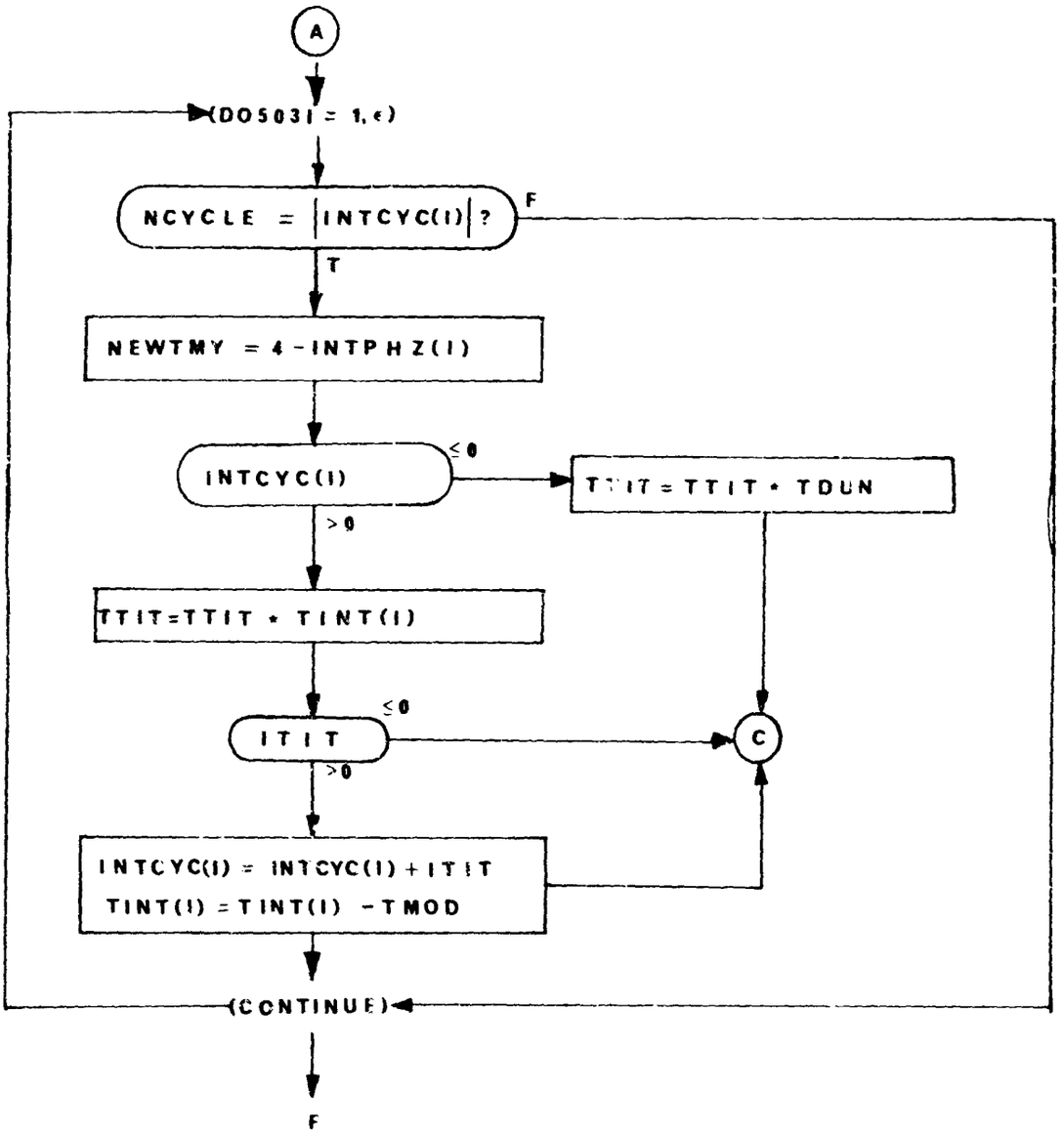


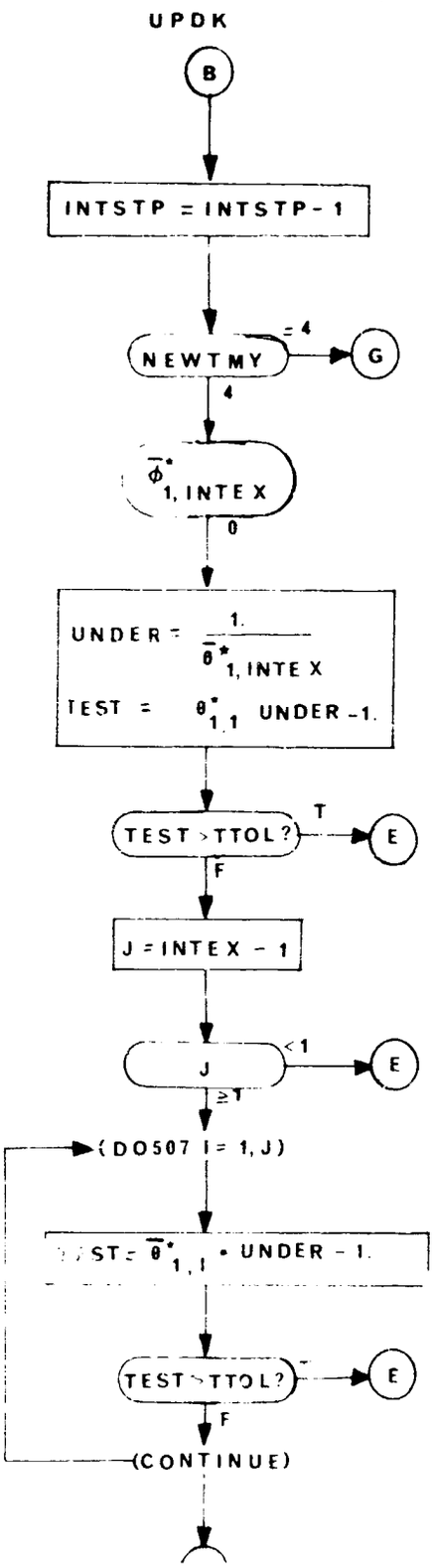
LOGIC OF UPDK (Cont.)

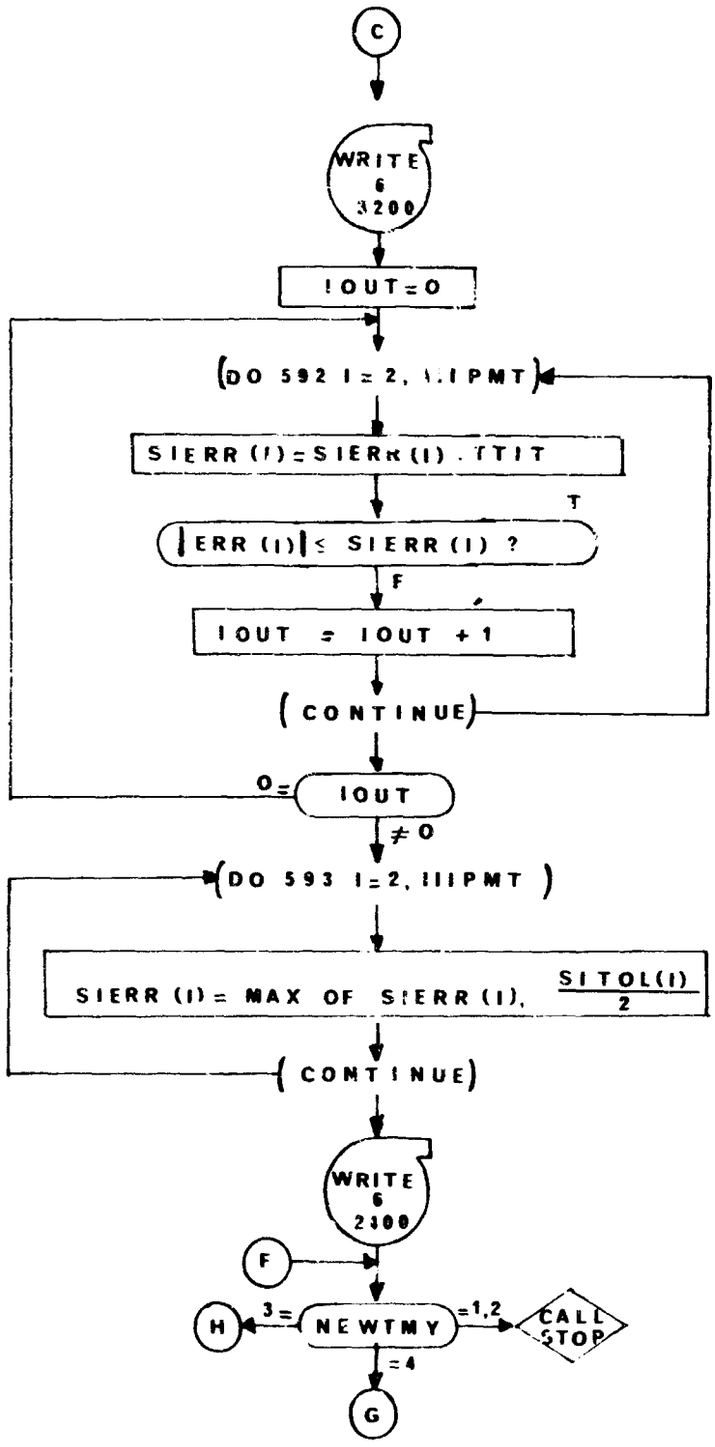


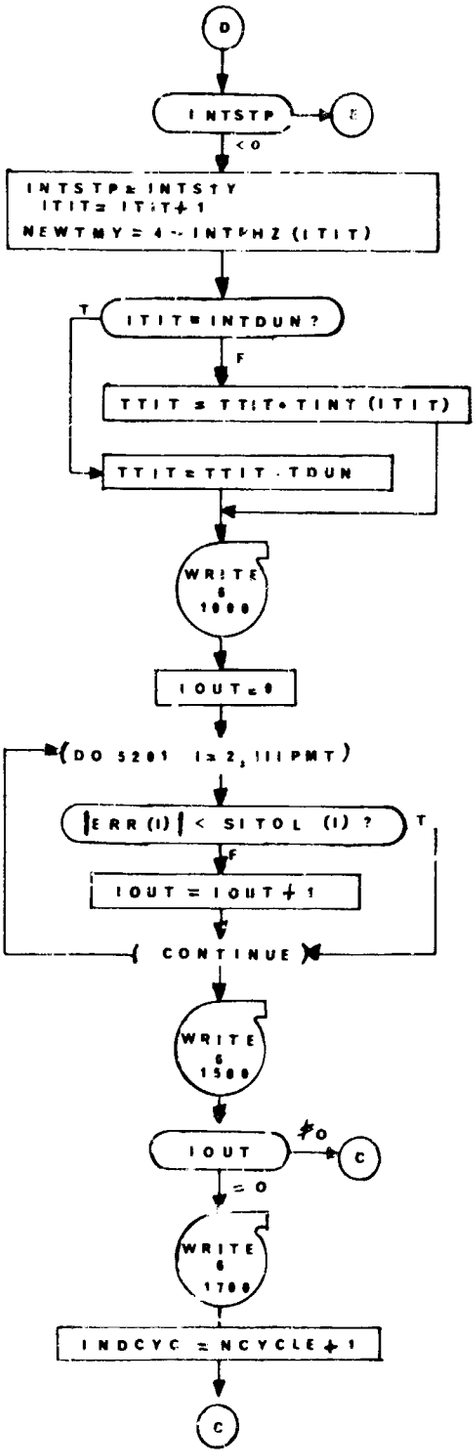
UPDK (1)

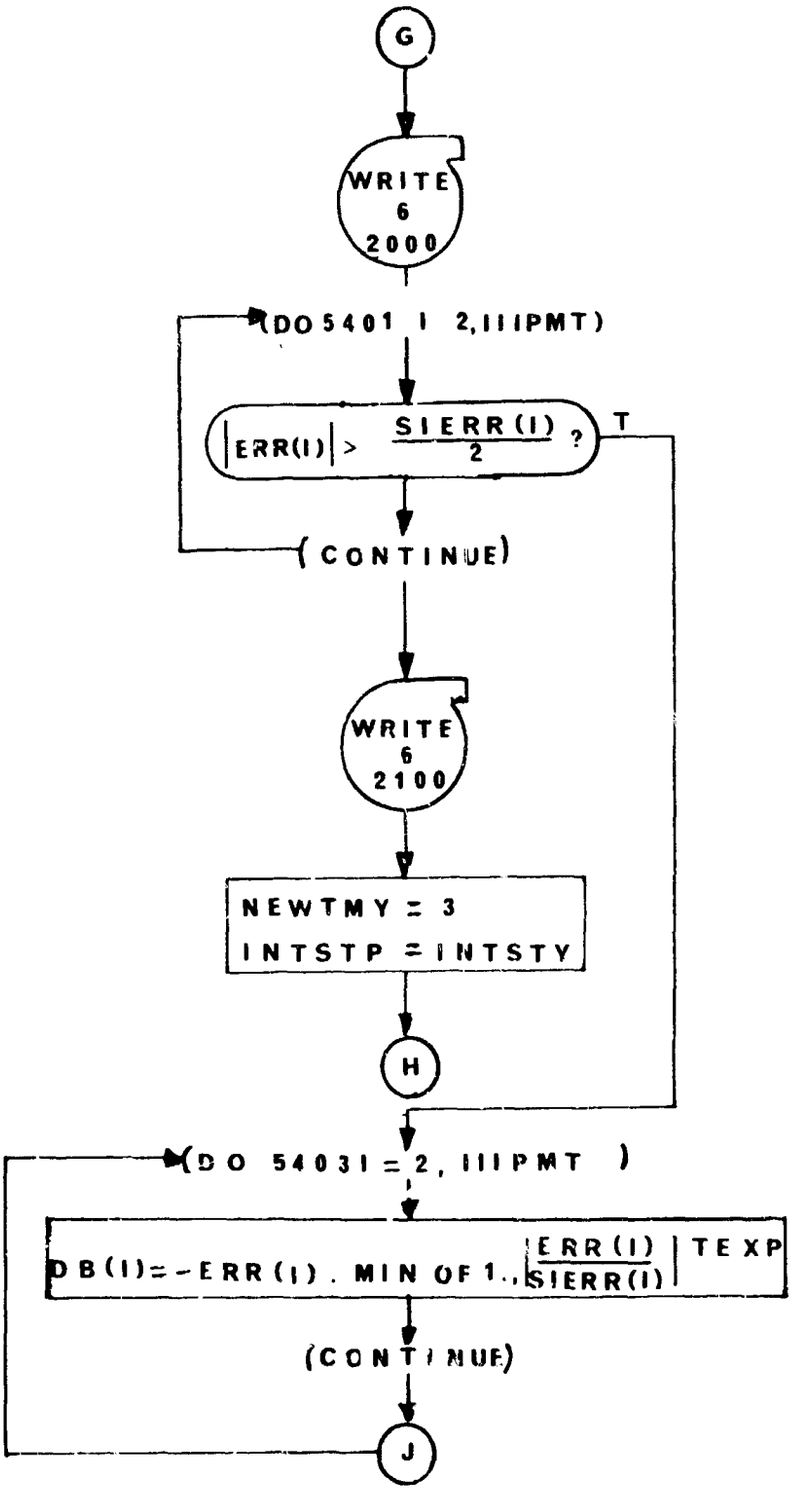

BEGIN
NEWTMY = INDTMY
ERR(1) = 0
(DO501 I=2, IIIPMT)
CP(I) = 0.
ERR(I) = $\bar{\psi}^{*}_{I,1}$ $\bar{\psi}_{I}$
CONTINUE
WRITE 6 2500
WRITE 6 1200
WRITE 6 1300
WRITE 6 2400
(DO521 I = 1, IIIPMT)
SIERR(I)
=0.
≠0.
SITOL(I)
=0.
≠0.
WRITE 6 3700
CALL STOP
(CONTINUE)
WRITE 6 3000
INDINT
≠0
=0
WRITE 6 2600
WRITE 6 2700
A
B

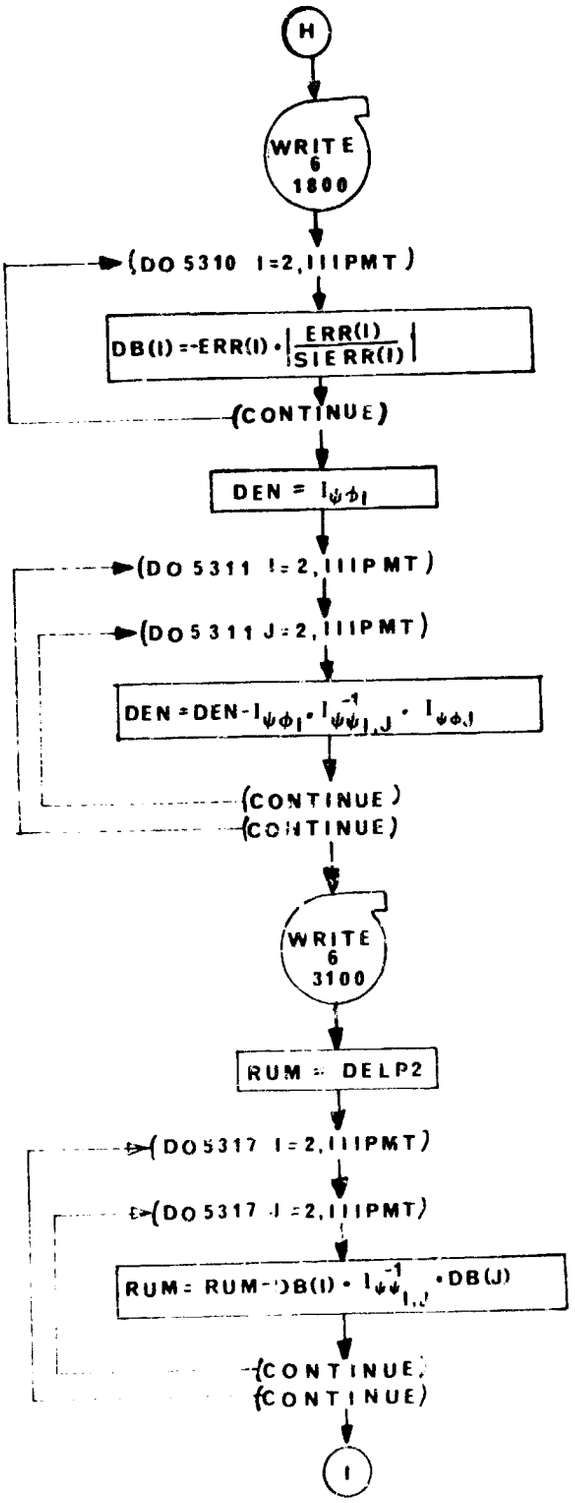

A
(DO 503 I = 1, ϵ)
NCYCLE = |INTCYC(I)| ?
F
T
NEWTMY = 4 - INTPHZ(I)
INTCYC(I)
≤ 0
TTIT = TTIT * TDUN
> 0
TTIT=TTIT * TINT(I)
ITIT
≤ 0
C
> 0
INTCYC(I) = INTCYC(I) + ITIT
TINT(I) = TINT(I) - TMOD
(CONTINUE)
F

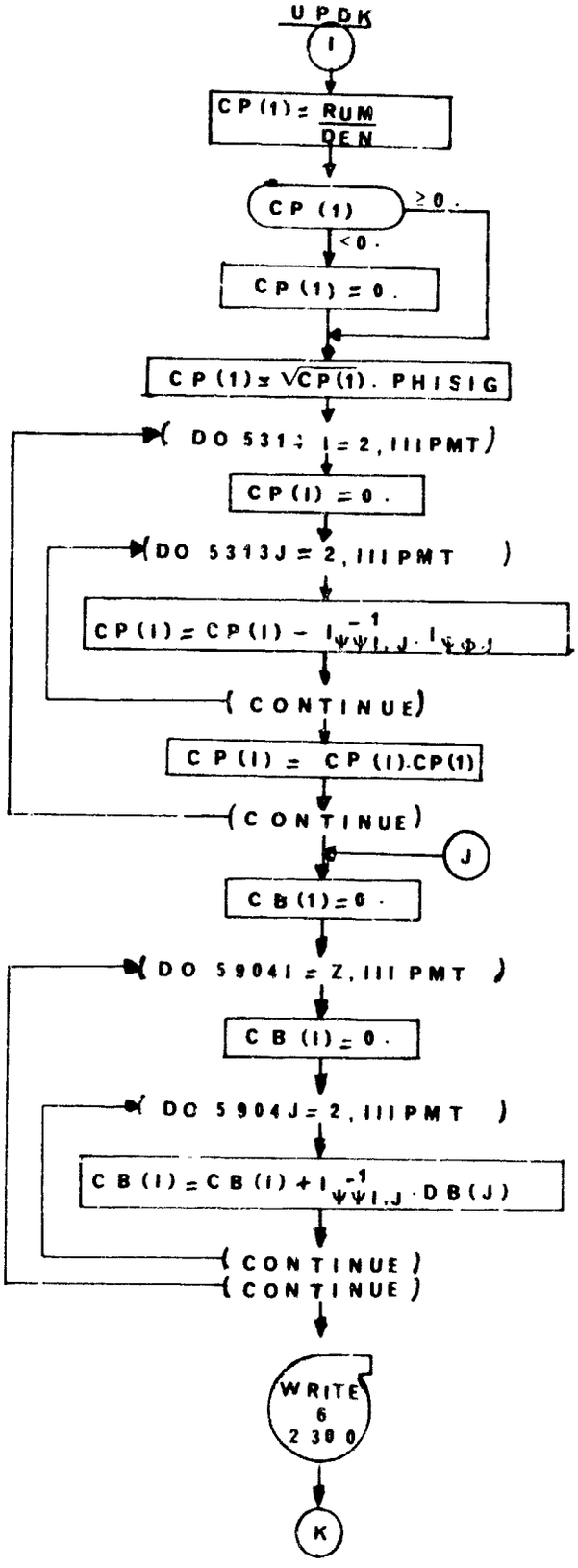

UPDK
B
INTSTP = INTSTP - 1
NEWTMY
= 4
G
4
$\bar{\phi}^*_{1,INTEX}$
0
UNDER = $\frac{1.}{\bar{\theta}^*_{1,INTEX}}$
TEST = $\theta^*_{1,1}$ UNDER - 1.
TEST > TTOL?
T
E
F
J = INTEX - 1
J
< 1
E
≥ 1
(DO507 I = 1, J)
TEST = $\bar{\theta}^*_{1,I}$ * UNDER - 1.
TEST > TTOL?
T
E
F
(CONTINUE)

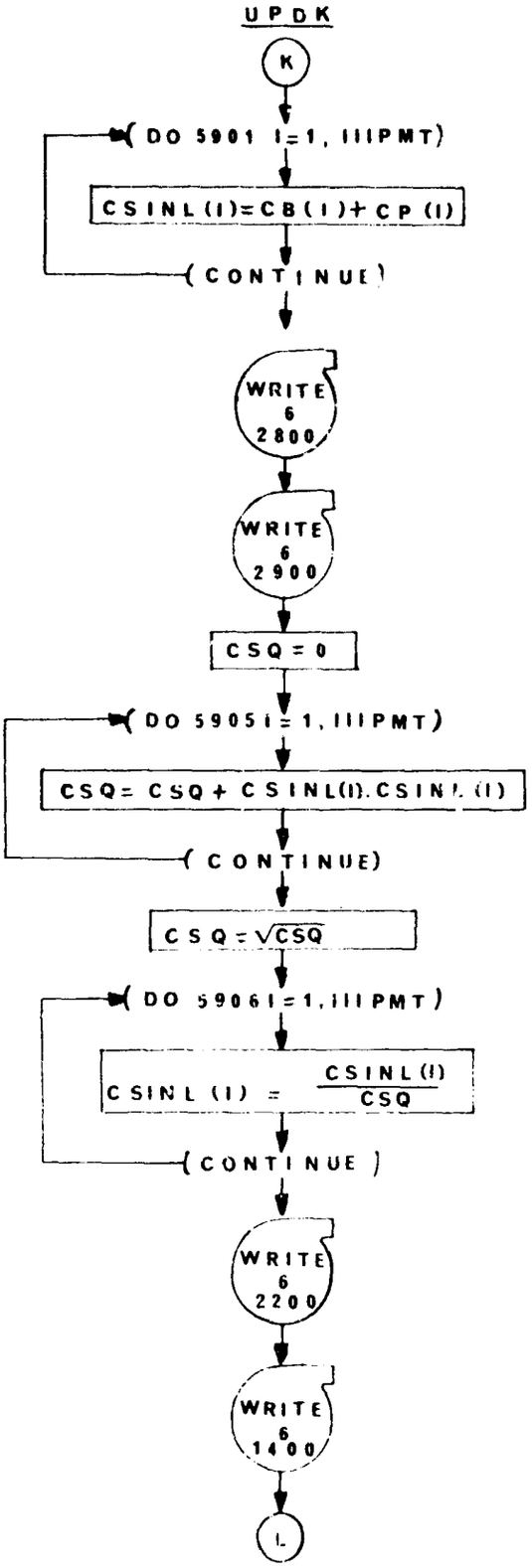

C
WRITE 6 3200
IOUT=0
DO 592 I=2, NIPMT
SIERR(I)=SIERR(I)*TTIT
T
|ERR(I)| ≤ SIERR(I) ?
F
IOUT = IOUT + 1
CONTINUE
0=
IOUT
≠0
DO 593 I=2, NIPMT
SIERR(I) = MAX OF SIERR(I), SITOL(I)/2
CONTINUE
WRITE 6 2300
F
H
3=
NEWTMY
=1,2
CALL STOP
=4
G

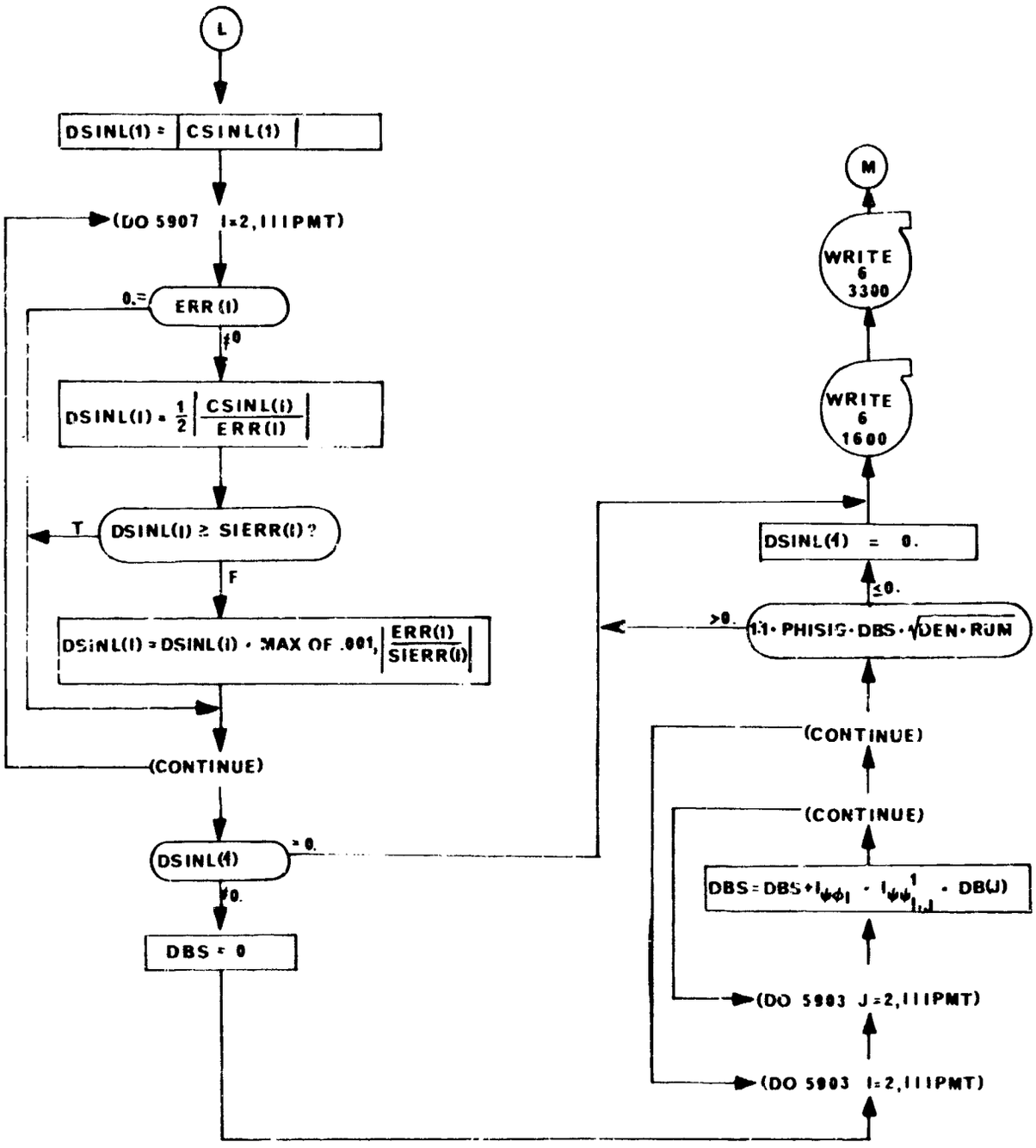

D
INTSTP
< 0
INTSTP = INTSTY
ITIT = ITIT + 1
NEWTMY = 4 ~ INTPHZ (ITIT)
T
ITIT = INTDUN ?
F
TTIT = TTIT + TINT (ITIT)
TTIT = TTIT - TDUN
WRITE 6 1000
IOUT = 0
(DO 5201 I = 2, IIPMT)
|ERR (I)| < SITOL (I) ?
T
F
IOUT = IOUT + 1
CONTINUE
WRITE 6 1500
IOUT
≠0
C
= 0
WRITE 6 1700
INDCYC = NCYCLE + 1
C

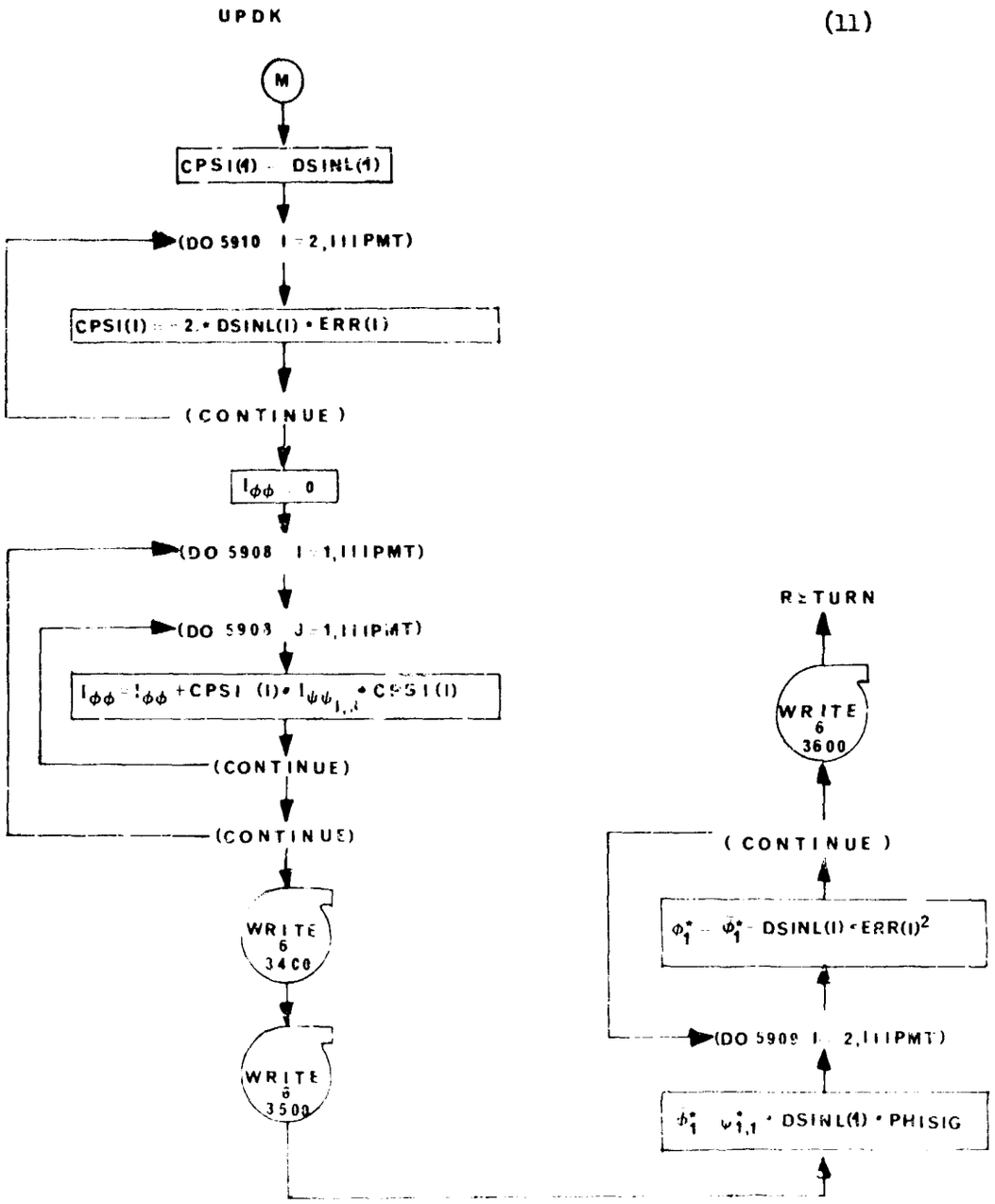

G
WRITE 6 2000
(DO 5401 I 2,IIIPMT)
|ERR(I)| > SIERR(I)/2 ?
T
(CONTINUE)
WRITE 6 2100
NEWTMY = 3
INTSTP = INTSTY
H
(DO 54031 = 2, IIIPMT)
DB(I)=-ERR(I) . MIN OF 1., |ERR(I)/SIERR(I)| TEXP
(CONTINUE)
J


H
WRITE 6 1800
(DO 5310 I=2,IIIPMT)
DB(I)=-ERR(I)•|ERR(I)/SIERR(I)|
(CONTINUE)
DEN = $I_{\psi\phi I}$
(DO 5311 I=2,IIIPMT)
(DO 5311 J=2,IIIPMT)
DEN=DEN-$I_{\psi\phi I}$•$I^{-1}_{\psi\psi I,J}$•$I_{\psi\phi J}$
(CONTINUE)
(CONTINUE)
WRITE 6 3100
RUM = DELP2
(DO5317 I=2,IIIPMT)
(DO5317 J=2,IIIPMT)
RUM=RUM-DB(I)•$I^{-1}_{\psi\psi I,J}$•DB(J)
(CONTINUE)
(CONTINUE)
I


UPDK
I
CP(1) = RUM / DEN
CP (1)
≥ 0.
< 0.
CP (1) = 0.
CP(1) = √CP(1) . PHISIG
DO 5313 I = 2, IIIPMT
CP(I) = 0.
DO 5313J = 2, IIIPMT
CP(I) = CP(I) - I⁻¹ ΨΨI,J · IΨΦ,J
CONTINUE
CP(I) = CP(I).CP(1)
CONTINUE
J
CB(1) = 0.
DO 5904I = 2, IIIPMT
CB(I) = 0.
DO 5904J = 2, IIIPMT
CB(I) = CB(I) + I⁻¹ ΨΨI,J · DB(J)
CONTINUE
CONTINUE
WRITE
6
2300
K


UPDK
K
DO 5901 I=1, IIIPMT
CSINL(I)=CB(I)+CP(I)
CONTINUE
WRITE 6 2800
WRITE 6 2900
CSQ = 0
DO 59051=1, IIIPMT
CSQ= CSQ + CSINL(I).CSINL(I)
CONTINUE
CSQ = √CSQ
DO 59061=1, IIIPMT
CSINL(I) = CSINL(I) / CSQ
CONTINUE
WRITE 6 2200
WRITE 6 1400
L


L
DSINL(1) = |CSINL(1)|
(DO 5907 I=2,IIIPMT)
0.= ERR(I)
≠0
DSINL(I) = 1/2 |CSINL(I) / ERR(I)|
T
DSINL(I) ≥ SIERR(I) ?
F
DSINL(I) = DSINL(I) + MAX OF .001, |ERR(I) / SIERR(I)|
(CONTINUE)
DSINL(1) = 0.
≠0.
DBS = 0
(DO 5903 I=2,IIIPMT)
(DO 5903 J=2,IIIPMT)
DBS = DBS + I_ψφI · I_ψψ(I,J) · DB(J)
(CONTINUE)
(CONTINUE)
>0.
1.1 · PHISIG · DBS · √DEN · RUN
≤0.
DSINL(1) = 0.
WRITE 6 1600
WRITE 6 3300
M


UPDK
M
CPSI(1) = DSINL(1)
(DO 5910 I = 2,IIIPMT)
CPSI(I) = 2.*DSINL(I)*ERR(I)
(CONTINUE)
$I_{\phi\phi} = 0$
(DO 5908 I = 1,IIIPMT)
(DO 5908 J = 1,IIIPMT)
$I_{\phi\phi} = I_{\phi\phi} + CPSI(I) * I_{\psi\psi_{I,J}} * CPSI(I)$
(CONTINUE)
(CONTINUE)
WRITE 6 3400
WRITE 6 3500
RETURN
WRITE 6 3600
(CONTINUE)
$\phi_1^* = \phi_1^* + DSINL(I) * ERR(I)^2$
(DO 5909 I = 2,IIIPMT)
$\phi_1^* = \psi_{1,1}^* * DSINL(1) * PHISIG$

SECTION VIII

PROGRAM REV

This program is the program for calculating the solutions to the adjoint equations. There is a striking similarity between the structure of the EXE segment and the REV segment. In fact, REV proper was programmed by modifying the same logic that was used in EXE.

The input to REV comes from three sources: value in COMMON, information on TAPE12 , and sometimes information on IATAP. The output from REV consists of information on ILTAP and values in COMMON/2/ and values in COMMON.

Large arrays must be dealt with in REV. In order to conserve core, it is assumed that COMMON/1/ and COMMON/5/ are available for use.

The Partial Tape

REV calls the routine UNPART to read the partial tape for everything except the P matrix. To get the P matrix (if it exists) at the beginning of a major stage REV itself reads the partial tape. (If the P matrix does not exist, REV constructs the identity matrix and puts it in P). The reading of the partial tape is the most critical task of the reverse trajectory. The trouble occurs because the information that is put on it, sequentially, in the forward trajectory is needed in the opposite order in the reverse trajectory. This means that the partial tape has to be read in the reverse direction by backspacing, reading, backspacing, etc. The F and G matrices are treated as functions of stage time in the reverse trajectory. The F and G stage time histories exist on the partial tape. Since the table of F and G's for an entire stage would be too large to accommodate in the machine, the table of F and G's which is actually used to do the interpolation is constructed from the two adjacent points which bracket the current value of stage time in the reverse trajectory. REV is responsible for updating the F and G tables as the stage time changes; it does this by shifting the table values and reading a new F and G point from the partial tape (via a call to UNPART) and putting the respective values into the vacated spots of the F and G tables.

Staging and Time Points in REV

ND is an indicator set in EXE and transmitted to REV on the partial tape. It informs REV when the end of a major stage (ND=1) has been reached (i.e., the beginning of a major stage in the forward trajectory), and also whether a P matrix exists on tape at the stage point (ND=3) or if the stage point corresponds to the very first point of the forward trajectory (ND=0)

The NPT values in the array TIMESA are those values of stage time which must be hit during a stage of the reverse integration. These points are determined during the integration of the forward trajectory; NPT and the TIMESA array go out on the last record written on the partial tape at the end of a major stage of the forward trajectory. Hence at the beginning of every major stage of the reverse trajectory this array is read in from TAPE12

(via the call to UNPART). Every stage time value at which there is an F and G matrix on TAPE12 must appear in the TIMESA array. REV keeps track of the next time point that must be hit in the cell TIMPT. When TIMPT is hit (i.e., when TIMES=TIMPT) this point is defined to be a stage point. The type of stage point is reflected in the value given to INDSTG; this is done in the area labeled "STGTST ROUTINE FOR REVERSE" in the listing. The meaning of the respective values of INDSTG follow:

INDSTG	= 0	not yet at a stage point
	= 1	the end of the reverse trajectory
	= 2	the end of a major stage of the reverse trajectory
	= 3	the end of a minor stage of the reverse trajectory (i.e., get new F and G)
	= 4	merely a time point (Throw away all points of the TIMESA array not needed by decrementing NPT by 1)

INTERPOLATION

Before REV calls ADJEQ (3) to do the bulk of the calculation of the derivatives, it computes the instantaneous approximation to the F and G matrices by calls to the interpolation routine TLUREV. Also at this time, REV calls TLUREV to obtain the approximation to the instantaneous values of the control variables. (The control variables are put on TAPE12 at the same time F and G are put on TAPE12). This method of preserving the control variables is not always exact. An alternate method which is better but which involves more effort is available; it involves use of the CRVSR routine.

Integration

It is assumed that only fixed step integration is to be used in REV. This is not too strong an assumption since the density of points of the TIMESA array will dictate the requirement of taking relatively small steps. The integration step to be used in REV is DELTSR. DELTSR is transmitted to REV on the partial tape. Hence DELTSR may appear in major or minor stage data of EXE. If DELTSR is not input EXE initializes DELTSR to DELTS at every major stage.

Numbered COMMON Usage

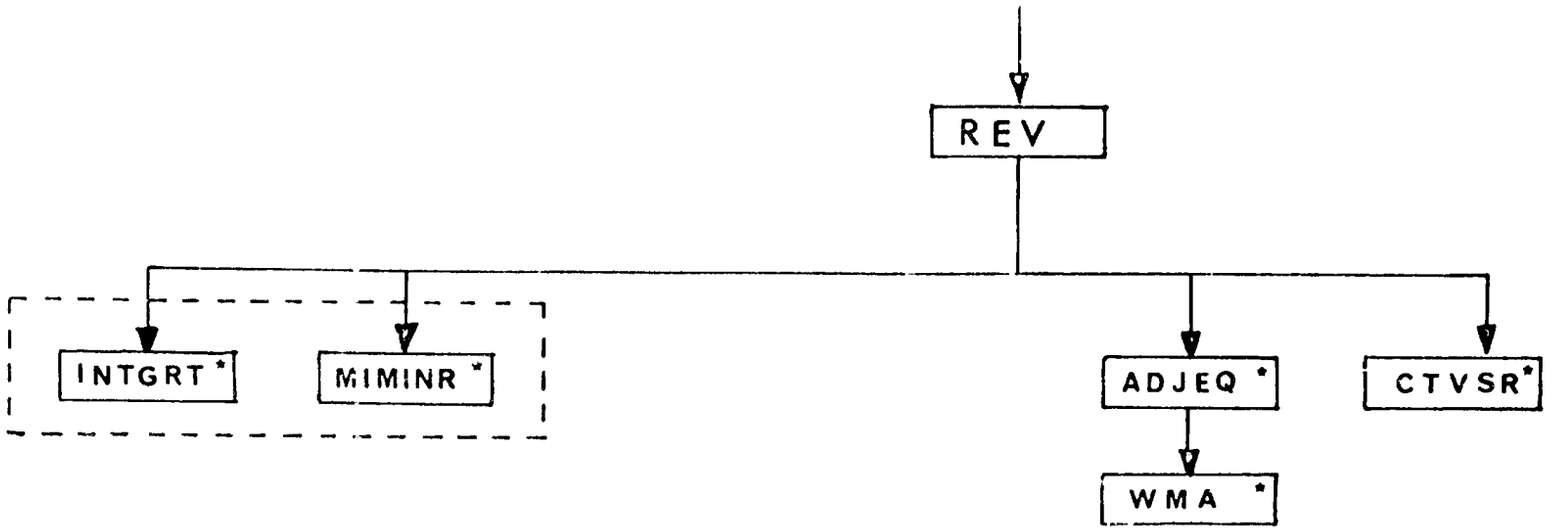
COMMON /1/ is assumed to be free. Many of the large arrays used in REV are in the COMMON /1/ block. COMMON/2/ is for the matrices $I_{\psi\psi}$, $I_{\psi\phi}$, $I_{\phi\phi}$. COMMON/5/ is assumed to be free and will be used whenever any features of the routine CTVSR are used. The core requirements of REV can be cut down considerably if appropriate arrays are equivalenced to one another.

The following is a list of some of the variables used in REV which are important and which may not be described elsewhere since they are not in COMMON:

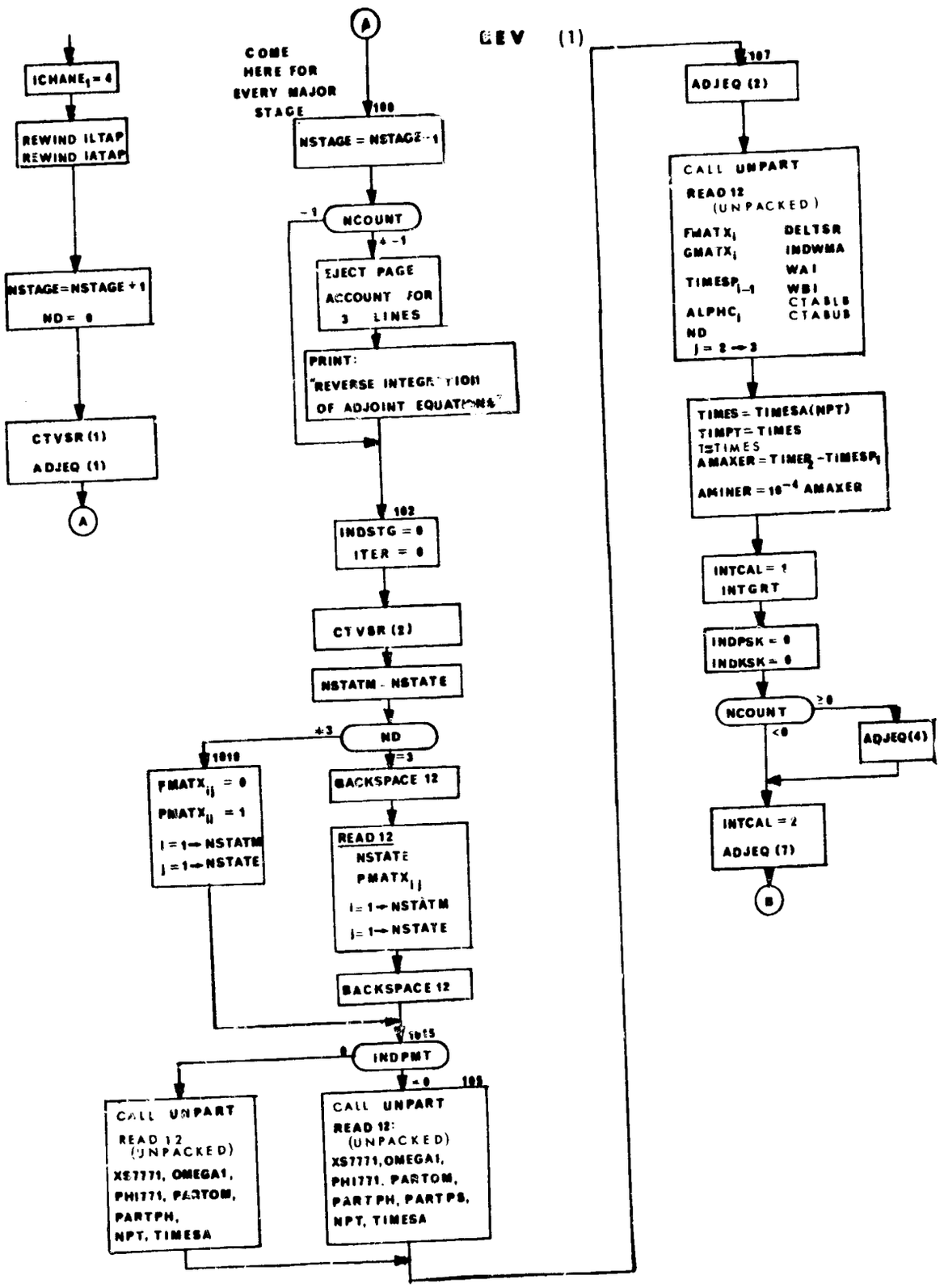
ALPHC(3,6)	2 point table of control variable values and the interpolated values.
FMATX(3,15,15)	2 point table of F matrix values and the interpolated values.

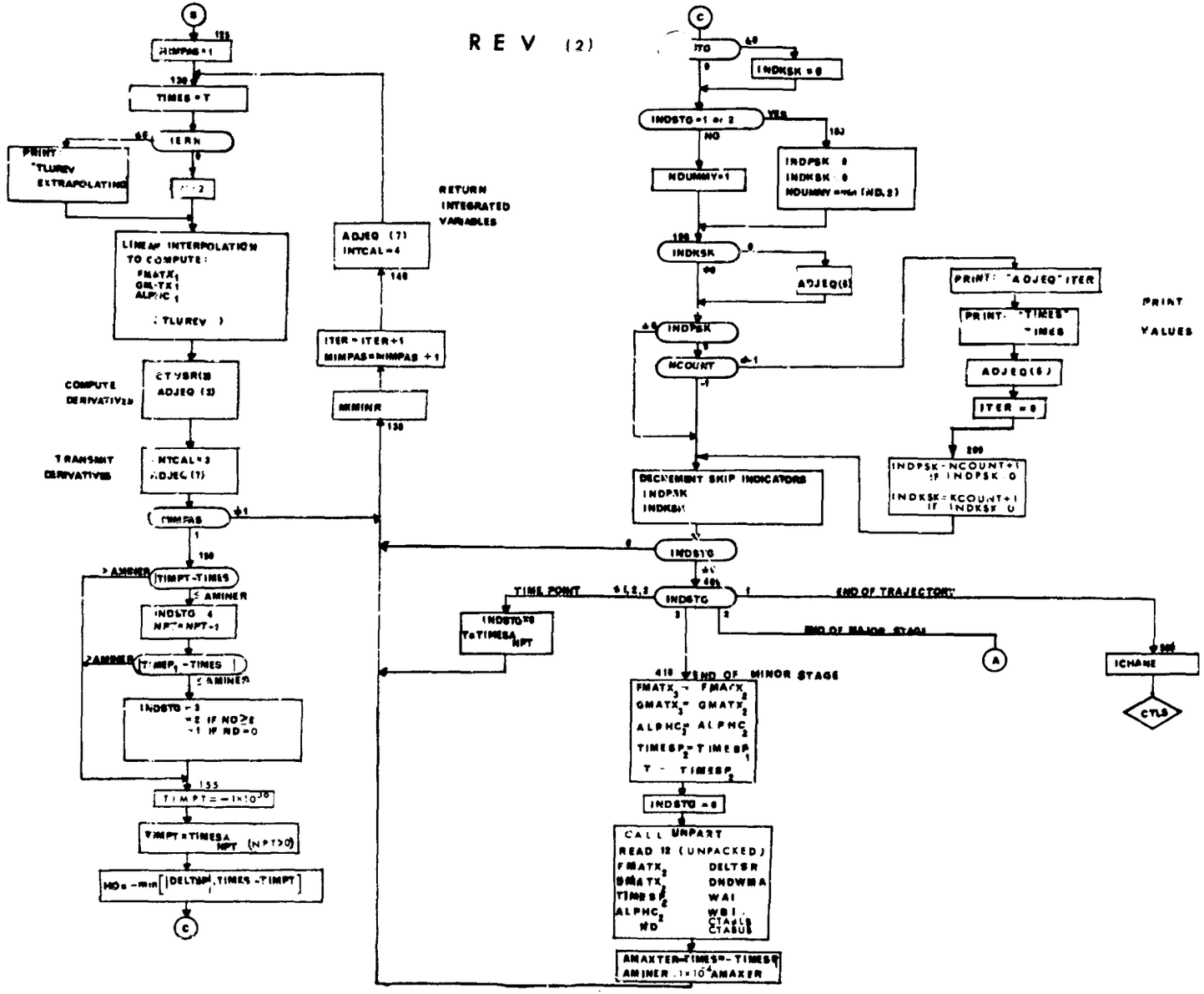
GMATX(3,15,6)	2 point table of G matrix values and the interpolated values.
TIMESP(2)	2 point table of the stage times for the above three tables.
NDUMMY	an indicator which mimics the indicator ND and which is put on tape DLTAP (by ADJEQ(8)) and which serves the same purpose in DALCAL as ND does in REV.
TIMESA (500)	an array of stage time points which must be hit.
NPT	the number of points in the TIMESA array.
TIMPT	the next time point from the TIMESA array which must be hit.

Organizational Chart for REV



*DESIGNATES MULTIPLE ENTRY POINTS


ICHANE$_1$ = 4
REWIND ILTAP
REWIND IATAP
NSTAGE=NSTAGE + 1
ND = 0
CTVSR (1)
ADJEQ (1)
A
COME HERE FOR EVERY MAJOR STAGE
100
NSTAGE = NSTAGE - 1
NCOUNT
EJECT PAGE
ACCOUNT FOR 3 LINES
PRINT: "REVERSE INTEGRATION OF ADJOINT EQUATIONS"
102
INDSTG = 0
ITER = 0
CTVSR (2)
NSTATM - NSTATE
ND
1010
FMATX$_{ij}$ = 0
PMATX$_{ii}$ = 1
i = 1→ NSTATM
j = 1→ NSTATE
BACKSPACE 12
READ 12
NSTATE
PMATX$_{ij}$
i = 1→ NSTATM
j = 1→ NSTATE
BACKSPACE 12
INDPMT
CALL UNPART
READ 12 (UNPACKED)
XS7771, OMEGA1, PHI771, PARTOM, PARTPH, NPT, TIMESA
105
CALL UNPART
READ 12: (UNPACKED)
XS7771, OMEGA1, PHI771, PARTOM, PARTPH, PARTPS, NPT, TIMESA
REV (1)
107
ADJEQ (2)
CALL UNPART
READ 12 (UNPACKED)
FMATX$_i$ DELTSR
GMATX$_i$ INDWMA
WAI
TIMESP$_{i-1}$ WBI
CTABLB
ALPHC$_i$ CTABUB
ND
j = 2 → 3
TIMES = TIMESA(NPT)
TIMPT = TIMES
T=TIMES
AMAXER = TIMER$_2$ - TIMESP$_1$
AMINER = 10^{-4} AMAXER
INTCAL = 1
INTGRT
INDPSK = 0
INDKSK = 0
NCOUNT
ADJEQ(4)
INTCAL = 2
ADJEQ (7)
B


REV (2)
TIMES = T
IERN
PRINT "TLUREV EXTRAPOLATING"
LINEAR INTERPOLATION TO COMPUTE:
COMPUTE DERIVATIVES
TRANSMIT DERIVATIVES
TIMPT-TIMES
INDSTG = 4
NPT = NPT-1
INDSTG = 3
TIMPT = TIMESA NPT (NPT>0)
RETURN INTEGRATED VARIABLES
ADJEQ (7)
INTCAL = 4
ITER = ITER+1
MIMPAS = MIMPAS + 1
MIMINR
INDKSK = 0
INDSTG = 1 or 2
YES
NO
NDUMMY = 1
INDPSK
ADJEQ(5)
NCOUNT
PRINT: "ADJEQ" ITER
PRINT: "TIMES" TIMES
ADJEQ(5)
ITER = 0
INDPSK = NCOUNT+1 IF INDPSK 0
INDKSK = KCOUNT+1 IF INDKSK 0
PRINT VALUES
DECREMENT SKIP INDICATORS
INDPSK
INDKSK
INDSTG
TIME POINT
END OF TRAJECTORY
END OF MAJOR STAGE
END OF MINOR STAGE
ICHANE
CTLS
CALL UNPART
READ 12 (UNPACKED)
DELTSR
DNDWMA
WAI
WBI
CTABLS
CTABUS
AMAXTER = TIMES - TIMESP
AMINER = 1×10^-4 AMAXER

1. CTVSR - Control Variable Routine for REV

Purpose

During the reverse trajectory, to obtain what is needed from the information on IATAP (prepared in DAICAL or MAIN1).

Method

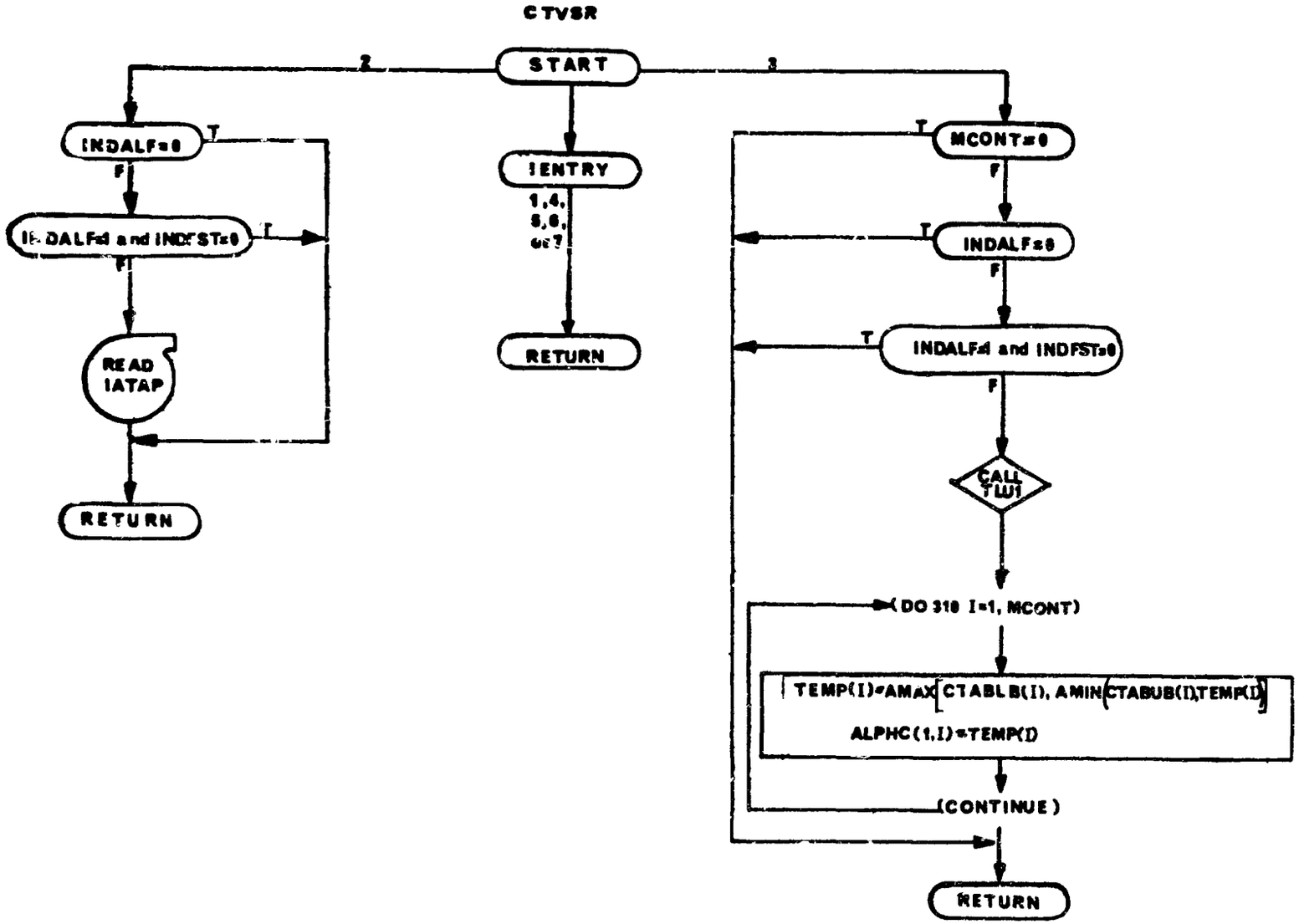
To accomplish reading IATAP during the reverse trajectory, it is necessary to request this in the data (DATA1):

INDALF	0	IATAP is not read in REV.
INDALF	1	IATAP is read in REV on all cycles after the nominal.
INDALF	2	IATAP is first prepared with all the information that REV needs from it (on the nominal) in MAIN1. Then IATAP is read in REV on every cycle including the nominal.

CTVSR(2) reads IATAP if it is needed. If so, then CTVSR(3) computes the control variable values from the CTABLE which was read from IATAP at CTVSR(2) in the same manner that CTVS(3) does for the forward trajectory. One reason for requesting that this be done is to preserve the CTABLE at time points closer than those time points at which partials are computed. (the CTABLE is always preserved at those time points where partials are computed without the need for doing CTVSR calculations.) To request this optional type preservation of CTABLE requires that DELTSA be input as well as INDALF (see CTVS). Another reason for requesting that IATAP be read is to obtain the weighting matrix table that was used on the previous cycle (this is needed if and only if INDWMA = 5).

Remarks

CTVSR is to REV as CTVS is to EXE.


CTVSR
START
2
3
INDALF=0
T
F
IF DALF=1 and INDFST=0
T
F
READ IATAP
RETURN
IENTRY
1,4,
5,6,
or 7
RETURN
MCONT=0
T
F
INDALF=0
T
F
INDALF=1 and INDFST=0
T
F
CALL TW1
DO 310 I=1, MCONT
TEMP(I)=AMAX[CTABLB(I), AMIN(CTABUB(I),TEMP(I)]
ALPHC(1,I)=TEMP(I)
CONTINUE
RETURN

2. ADJEQ - Adjoint Equations Routine

Purpose

This is the basic subprogram of REV. ADJEQ is to REV as DIFEQ is to EXE. ADJEQ contains the equations for computing the derivatives of the adjoint variables at ADJEQ(3) and the transformation of the adjoint variable across major stage points at ADJEQ(2). A description of the notation used follows:

PHIOLA	$\lambda_{\varphi\Omega}$
PSIOLA	$\lambda_{\Psi\Omega}$
PHIOLG	$\lambda_{\varphi\Omega}G$
PSIOLG	$\lambda_{\Psi\Omega}G$
PHOLGW	$\lambda_{\varphi\Omega}GW^{-1}$
PSOLGW	$\lambda_{\Psi\Omega}GW^{-1}$
PHIOL1	$\dot{\lambda}_{\varphi\Omega} = -F' \lambda_{\varphi\Omega}$
PSIOL1	$\dot{\lambda}_{\Psi\Omega} = -F' \lambda_{\Psi\Omega}$
FMATX	F'
GMATX	G
PMATX	P
RMATX	R
PARTOM	$\frac{\partial\Omega}{\partial X}$
PARTPH	$\frac{\partial\varphi}{\partial X}$
PARTPS	$\frac{\partial\Psi}{\partial X}$
PSI771	$\dot{\Psi}$
PHI771	$\dot{\varphi}$
OMEGA1	$\dot{\Omega}$
XS7771	$\dot{X}$

(FMATX, GMATX, PMATX, RMATX, PARTOM, PARTPH, PARTPS, PSI771, PHI771, OMEGA1, XS7771: come from PARTS on TAPE12)

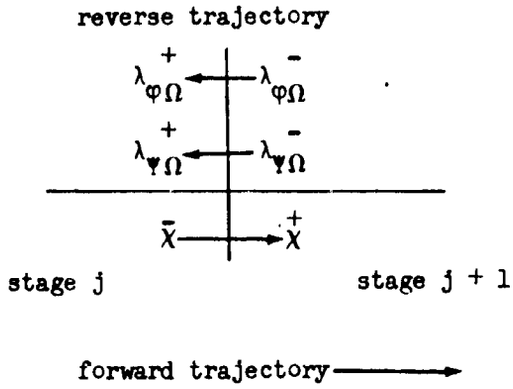
PHJUMP	$\lambda_{\varphi\Omega}P\dot{X}$
PSJUMP	$\lambda_{\Psi\Omega}P\dot{X}$

SISII	$-\int_T^t \lambda_{\psi\Omega} G\, W^{-1} (\lambda_{\psi\Omega} G)' dt = I_{\psi\psi}\ (+ K_{\psi\psi})$
FEFEI	$-\int_T^t \lambda_{\varphi\Omega} G\, W^{-1} (\lambda_{\varphi\Omega} G)' dt = I_{\varphi\varphi}\ (+ K_{\varphi\varphi})$
SIFEI	$-\int_T^t \lambda_{\psi\Omega} G\, W^{-1} (\lambda_{\varphi\Omega} G)' dt = I_{\psi\varphi}\ (+ K_{\psi\varphi})$
SISIK	$\lambda_{\psi\Omega} R\, U^{-1} (\lambda_{\psi\Omega} R)' = K_{\psi\psi}$,
SIFEK	$\lambda_{\psi\Omega} R\, U^{-1} (\lambda_{\varphi\Omega} R)' = K_{\psi\varphi}$
FEFEK	$\lambda_{\varphi\Omega} R\, U^{-1} (\lambda_{\varphi\Omega} R)' = K_{\varphi\varphi}$
PHIOLR	$\lambda_{\varphi\Omega} R$ (or $\lambda_{\varphi\Omega} P$)
PSIOLR	$\lambda_{\psi\Omega} R$ (or $\lambda_{\psi\Omega} P$)
PHOLRU	$\lambda_{\varphi\Omega} R\, U^{-1}$
PSOLRU	$\lambda_{\psi\Omega} R\, U^{-1}$
ALPSI	$\int_T^t \lvert\lambda_{\varphi\Omega} G\rvert\, dt$ or $\int_T^t WW' dt$
ALPSI1	$\lvert\lambda_{\varphi\Omega} G\rvert$ or WW'
ALPHC	α
VALINS	χ_o
SISII1	$-\lambda_{\psi\Omega} G\, W^{-1} (\lambda_{\psi\Omega} G)'$
FEFEI1	$-\lambda_{\varphi\Omega} G\, W^{-1} (\lambda_{\varphi\Omega} G)'$
SIFEI1	$-\lambda_{\psi\Omega} G\, W^{-1} (\lambda_{\varphi\Omega} G)'$
UMATX	U^{-1}
W	W^{-1}
TIMES	t_s (stage time)

Method

ADJEQ(1): The integrated variables are initialized to 0. These variables include $\lambda_{\varphi\Omega}$, $\lambda_{\psi\Omega}$, $I_{\varphi\varphi}$, $I_{\psi\varphi}$, $I_{\psi\psi}$, and ALPSI. This point of the reverse trajectory corresponds to the last point of the forward trajectory.

ADJEQ(2): Entry is made here at the first point of every major stage of the reverse trajectory; this point corresponds to the last point of the corresponding stage of the forward trajectory.



$\lambda_{\varphi\Omega}$ and $\lambda_{\psi\Omega}$ will, in general, be discontinuous at a major stage point. The initial conditions for these variables at the beginning of the new stage ($\lambda_{\varphi\Omega}^{+}$, $\lambda_{\psi\Omega}^{+}$) are expressed in terms of their values at the end of the last stage ($\lambda_{\varphi\Omega}^{-}$, $\lambda_{\psi\Omega}^{-}$) and the values $\dot{\chi}$, $\dot{\Omega}$, $\frac{\partial\varphi}{\partial\chi}$, $\frac{\partial\psi}{\partial\chi}$, $\frac{\partial\Omega}{\partial\chi}$, and P which have been computed at the corresponding stage point of the forward trajectory and have been transmitted on tape to the REV segment. The transformation is:

$$\lambda_{\varphi\Omega}^{+} = \lambda_{\varphi\Omega}^{-} P + \frac{\partial\varphi}{\partial\chi} - \left[\frac{\dot{\varphi} + \lambda_{\varphi\Omega}^{-} P\dot{\chi}}{\dot{\Omega}}\right] \frac{\partial\Omega}{\partial\chi}$$

$$\lambda_{\psi\Omega}^{+} = \lambda_{\psi\Omega}^{-} P + \frac{\partial\psi}{\partial\chi} - \left[\frac{\dot{\psi} + \lambda_{\psi\Omega}^{-} P\dot{\chi}}{\dot{\Omega}}\right] \frac{\partial\Omega}{\partial\chi}$$

ADJEQ(3): At this entry point the derivatives of the integrated variables are computed. In the process, $\lambda_{\phi\Omega} G W^{-1}$ and $\lambda_{\psi\Omega} G W^{-1}$ are computed; these will be put on tape at ADJEQ(8).

ADJEQ(4): This is entered at the beginning of every major stage, after ADJEQ(2) has been called. At this entry point values of PHJUMP and PSJUMP are output. These had been computed in the process of making the transformation at ADJEQ(2).

ADJEQ(5): Not used.

ADJEQ(6): Code printing and value printing are done for the following variables: $\lambda_{\varphi\Omega}$, $\lambda_{\psi\Omega}$, $\lambda_{\varphi\Omega} G$, $\lambda_{\psi\Omega} G$. If the end of the trajectory is at hand, then code and value printing are done also for the following variables: $I_{\varphi\varphi}$, $I_{\psi\varphi}$, $I_{\psi\psi}$, and $\lambda_{\varphi\Omega} R$, $\lambda_{\psi\Omega} R$, $K_{\varphi\varphi}$, $K_{\psi\varphi}$, $K_{\psi\psi}$ (if they have been computed).

ADJEQ(7): This entry point defines the variables that are to be integrated (see ADJEQ(1)).

ADJEQ(8): This entry point is called immediately after each integration step. The purpose is to put $\lambda_{\varphi\Omega}G\,W^{-1}$ and $\lambda_{\Psi\Omega}GW^{-1}$, W^{-1}, t_s, and α on tape ILTAP so that the information may be used later on to compute the perturbation to make to the α history. In addition, if it is the last point of the reverse trajectory (corresponds to first point of forward trajectory), and if initial conditions are being perturbed, then $\lambda_{\varphi\Omega}R$, $\lambda_{\Psi\Omega}R$, $\lambda_{\varphi\Omega}R\,U^{-1}$, $\lambda_{\Psi\Omega}R\,U^{-1}$, $K_{\varphi\varphi}$, $K_{\Psi\varphi}$, $K_{\Psi\Psi}$ are computed. The K's are added to the corresponding I's then $\lambda_{\varphi\Omega}R\,U^{-1}$, $\lambda_{\Psi\Omega}RU^{-1}$ and x_o are put on ILTAP. Except for the last record, ILTAP is prepared by calling the ad-hoc blocking routine DALPACK.

Remarks

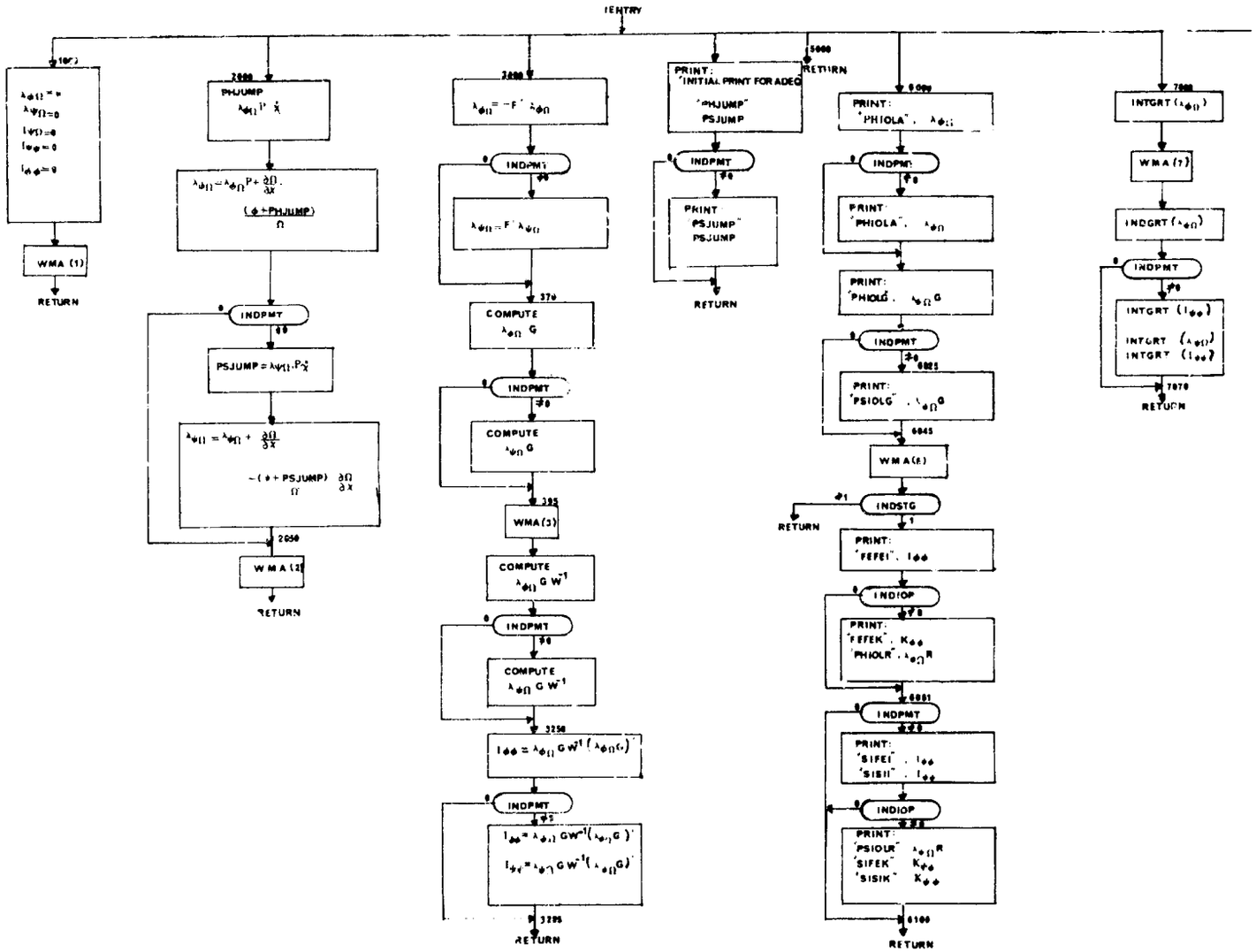
1. Any computations pertaining to the Ψ functions will not be done if INDPMT = 0. INDPMT is the number of Ψ - components in the various vectors and matrices e.g., $\lambda_{\Psi\Omega}$, $I_{\Psi\Psi}$, etc. (The Ψ-functions are those variables listed in the ENDCON array).

2. Computations for initial condition perturbation e.g. $\lambda_{\varphi\Omega}R$, $K_{\varphi\varphi}$, $K_{\Psi\varphi}$, etc. are not done if INDICP = 0.

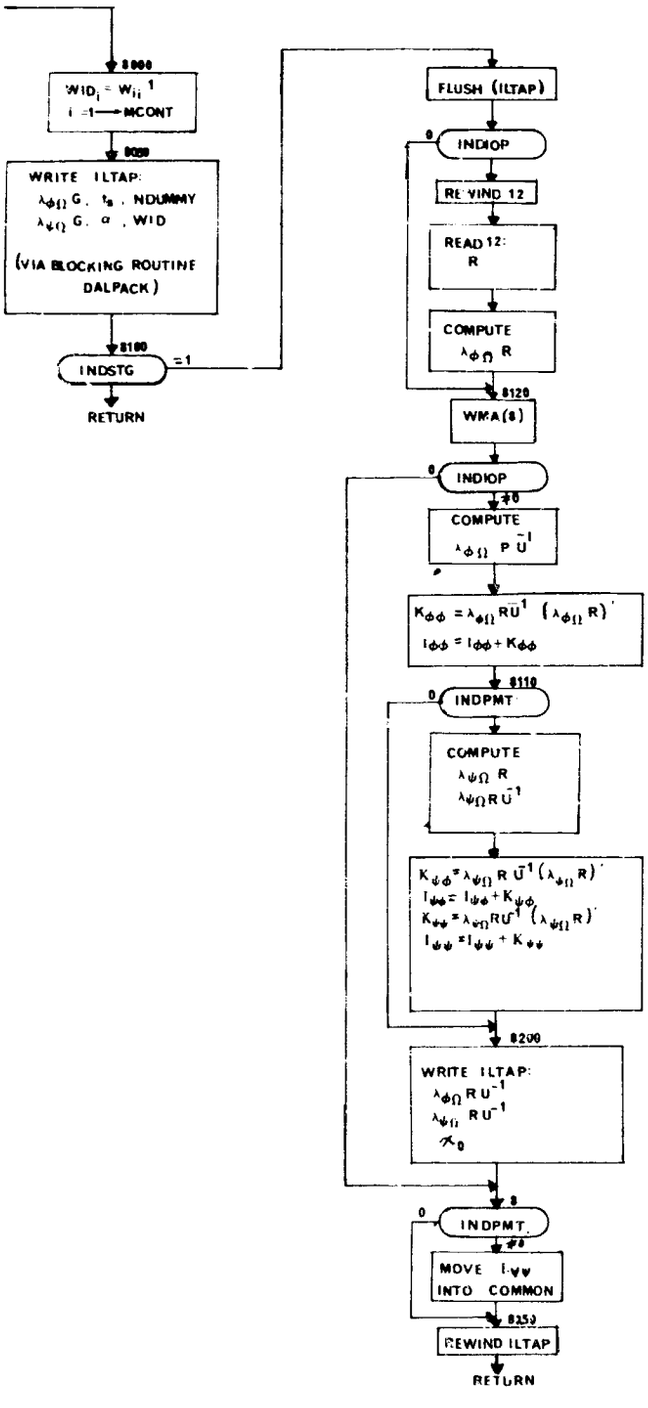
3. It is assumed that MCONT > 0; i.e. there is at least one control variable.

4. The information that ADJEQ needs from PARTS is available when ADJEQ is called. REV proper sees to it that the partial tape is read when it is needed. There is one exception to this: ADJEQ(8) reads the partial tape to obtain the R matrix (if it is needed).

ADJEQ (1)


IENTRY
WMA (1)
RETURN
PHJUMP
INDPMT
PSJUMP = λΨ(1)·P
WMA (2)
RETURN
COMPUTE
WMA (3)
INDPMT
RETURN
PRINT: "INITIAL PRINT FOR ADEQ"
"PHJUMP"
PSJUMP
INDPMT
PRINT "PSJUMP" PSJUMP
RETURN
RETURN
PRINT: "PHIOLA", λφ(1)
INDPMT
PRINT: "PHIOLA", λφΩ
PRINT: "PHIOLG", λφΩ G
INDPMT
PRINT: "PSIOLG", λφΩ G
WMA (4)
INDSTG
RETURN
PRINT: "FEFEI", Iφφ
INDIOP
PRINT: "FEFEK", Kφφ "PHIOLR", λφΩ R
INDPMT
PRINT: "SIFEI", Iφφ "SISII", Iφφ
INDIOP
PRINT: "PSIOR" λφΩ R "SIFEK" Kφφ "SISIK" Kφφ
RETURN
INTGRT (λφ(1))
WMA (7)
INDGRT (λφΩ)
INDPMT
INTGRT (Iφφ) INTGRT (λφ(1)) INTGRT (Iφφ)
RETURN

ADJEQ (2)


8000
WID$_i$ = W$_{ii}^{-1}$
i =1 → MCONT
8050
WRITE ILTAP:
$\lambda_{\phi\Omega}$ G, t_s, NDUMMY
$\lambda_{\psi\Omega}$ G, α, WID
(VIA BLOCKING ROUTINE DALPACK)
8100
INDSTG
=1
RETURN
FLUSH (ILTAP)
0
INDIOP
REWIND 12
READ 12:
R
COMPUTE
$\lambda_{\phi\Omega}$ R
8120
WRA(8)
0
INDIOP
≠0
COMPUTE
$\lambda_{\phi\Omega}$ P $\bar{U}^1$
$K_{\phi\phi} = \lambda_{\phi\Omega} R \bar{U}^1 (\lambda_{\phi\Omega} R)'$
$I_{\phi\phi} = I_{\phi\phi} + K_{\phi\phi}$
8110
0
INDPMT
COMPUTE
$\lambda_{\psi\Omega}$ R
$\lambda_{\psi\Omega} R \bar{U}^{-1}$
$K_{\psi\phi} = \lambda_{\psi\Omega} R \bar{U}^{-1} (\lambda_{\phi\Omega} R)'$
$I_{\psi\phi} = I_{\psi\phi} + K_{\psi\phi}$
$K_{\psi\psi} = \lambda_{\psi\Omega} R \bar{U}^{-1} (\lambda_{\psi\Omega} R)'$
$I_{\psi\psi} = I_{\psi\psi} + K_{\psi\psi}$
8200
WRITE ILTAP:
$\lambda_{\phi\Omega} R U^{-1}$
$\lambda_{\psi\Omega} R U^{-1}$
χ_0
0
INDPMT
≠0
MOVE $I_{\psi\psi}$
INTO COMMON
8250
REWIND ILTAP
RETURN

3. UNPART - Unblocking Routine for Partials

Purpose:

To unpack TAPE12 which has been packed in large arrays to eliminate unnecessary I/O device selection.

Method:

TAPE12 has been prepared in chain EXE. The records only contain non-zero from the output values in chain EXE.

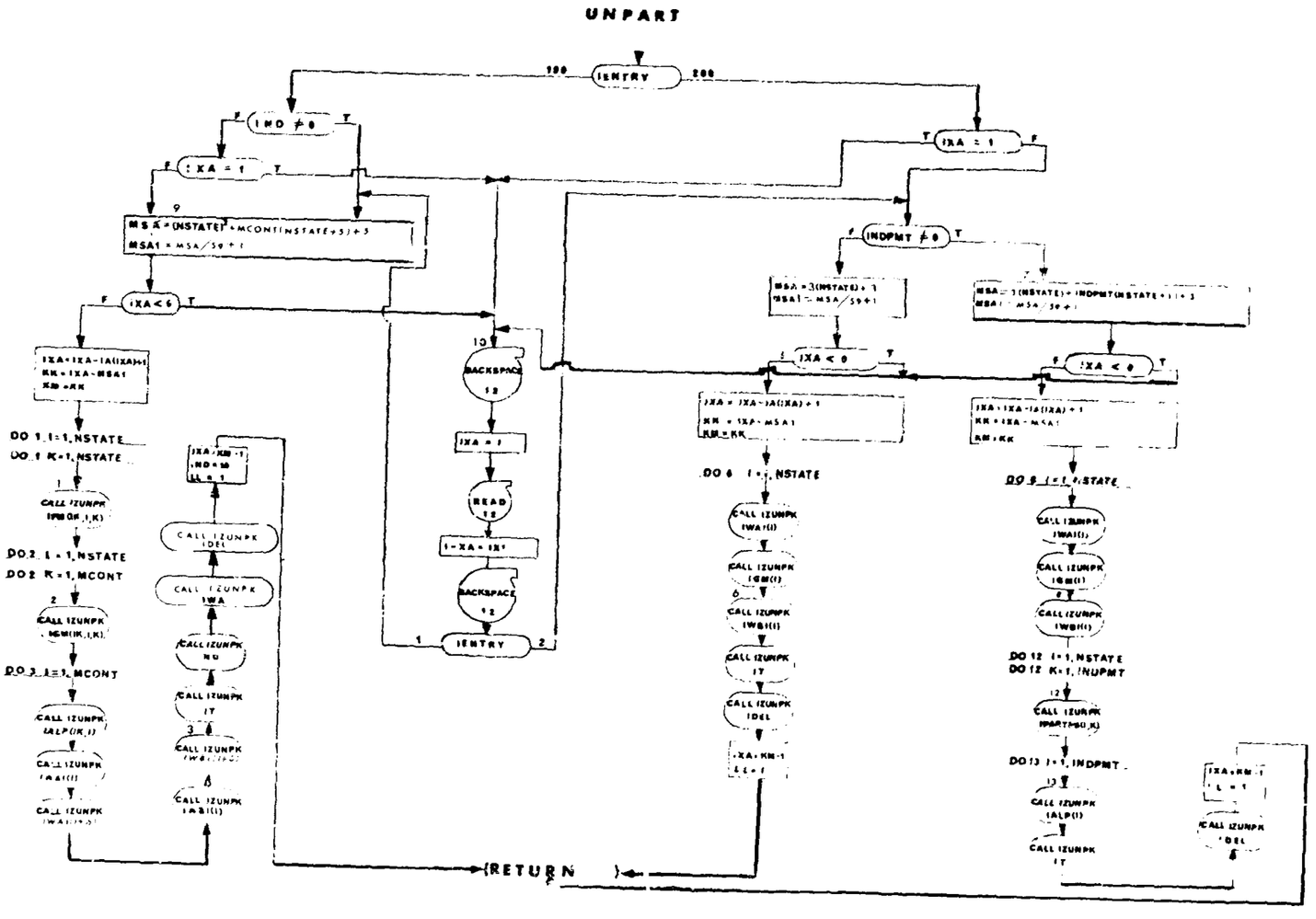
Usage:

Entry is made to the routine with the following statement:

```
CALL UNPART (MCONT,NSTATE,IFM,IGM,IT,ND,IALP,IDEL,IWA,IWAI,
CTABUB,INDPMT,IPARTPS,IK,IENTRY)
```

MCONT	=	Number of control variables.
NSTATE	=	Number of state variables.
IFM =	=	F from chain EXE.
IGM	=	G from chain EXE.
IT	=	Time.
ND	=	Control words.
IALP	=	Control Variables.
IDEL	=	Delt time used in reverse.
IWA	=	Weighting Matrix indicator.
IWAI	=	Weighting Matrix data.
CTABUB	=	Control variable bounds data.
INDPMT	=	Number of constraints.
IPARTPS	=	Terminal partials.
IK	=	Subscript used in storing data.
IENTRY	=	Entry point to subroutine UNPART.

Subroutines called from this routine include IZUNPK and normal FORTRAN I/O routines.


UNPART
100
IENTRY
200
IND ≠ 0
IXA = 1
MSA = (NSTATE) * MCONT(NSTATE+5)+5
MSA1 = MSA/59+1
IXA < 6
IXA = IXA - IA(IXA)+1
KK = IXA - MSA1
KB = KK
DO 1 I = 1, NSTATE
DO 1 K = 1, NSTATE
CALL IZUNPK
DO 2 I = 1, NSTATE
DO 2 K = 1, MCONT
CALL IZUNPK
DO 3 I = 1, MCONT
CALL IZUNPK
CALL IZUNPK
CALL IZUNPK
IXA = KB + 1
ND = 58
LL = 1
CALL IZUNPK
IDEL
CALL IZUNPK
IWA
CALL IZUNPK
ND
CALL IZUNPK
IT
CALL IZUNPK
CALL IZUNPK
IA(I)
10
BACKSPACE
12
IXA = 1
READ
12
I = IXA + IX
BACKSPACE
12
IENTRY
INDPMT ≠ 0
MSA = 3(NSTATE) + 1
MSA1 = MSA/59+1
MSA = 3(NSTATE) + INDPMT(NSTATE+1) + 3
MSA1 = MSA/59+1
IXA < 0
IXA = IXA - IA(IXA) + 1
KK = IXA - MSA1
KB = KK
IXA < 0
IXA = IXA - IA(IXA) + 1
KK = IXA - MSA1
KB = KK
DO 6 I = 1, NSTATE
CALL IZUNPK
IWA(I)
CALL IZUNPK
IGB(I)
CALL IZUNPK
IWB(I)
CALL IZUNPK
IT
CALL IZUNPK
IDEL
IXA = KK - 1
LL = 1
DO 8 I = 1, NSTATE
CALL IZUNPK
IWA(I)
CALL IZUNPK
IGB(I)
CALL IZUNPK
IWB(I)
DO 12 I = 1, NSTATE
DO 12 K = 1, INDPMT
CALL IZUNPK
IPARTMSO,K)
DO 13 I = 1, INDPMT
CALL IZUNPK
IALP(I)
CALL IZUNPK
IT
IXA = KK - 1
LL = 1
CALL IZUNPK
IDEL
RETURN

4. MIMINR - Integration Routine for REV

Purpose

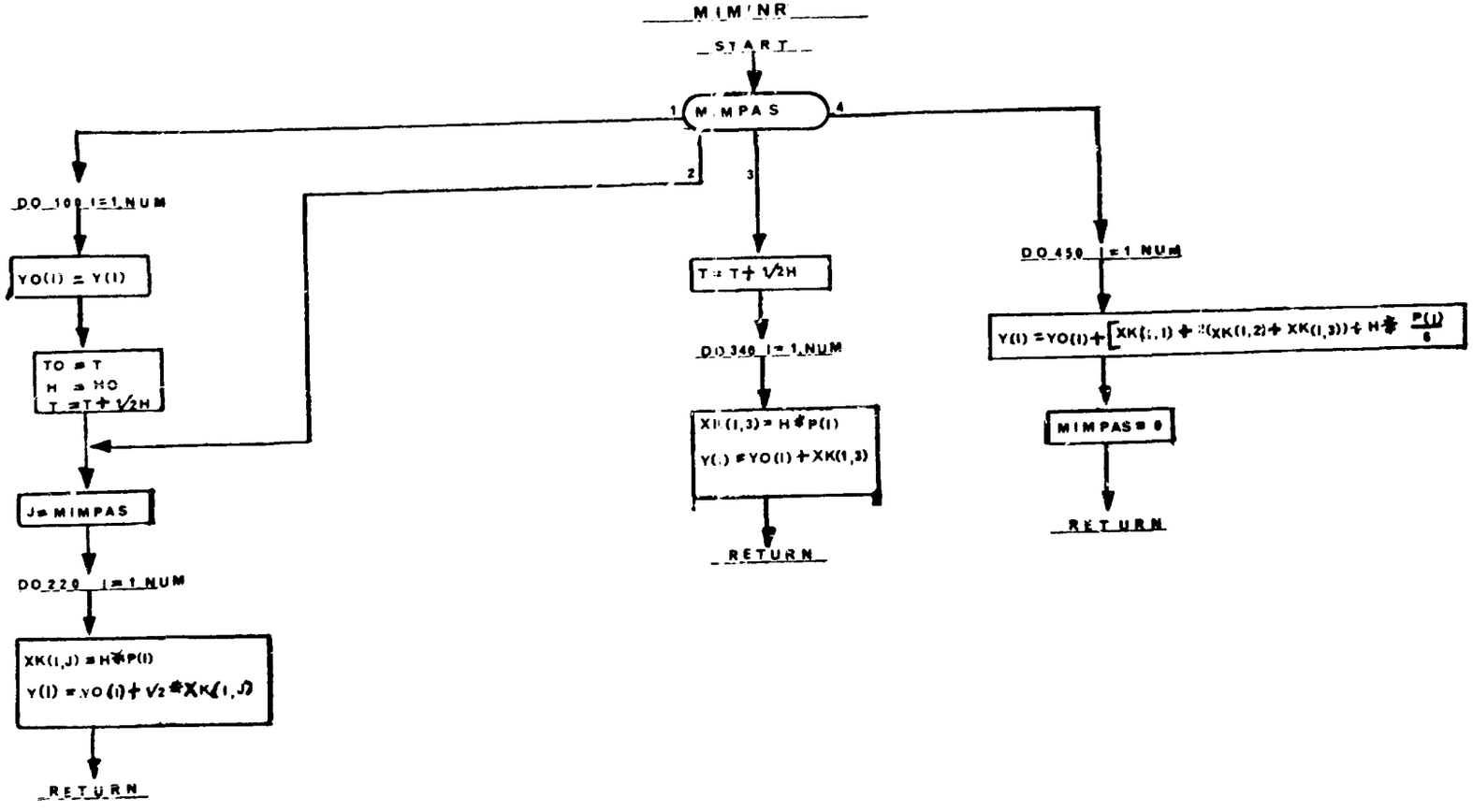
This routine is essentially the same as MIMINF. MIMINR is used for the reverse trajectory. It does not contain entry point 5 for truncation error estimation and step size control since it is assumed that fixed step integration will always be used in the reverse trajectory.

Method

The relationship of MIMINR to REV is analogous to the relationship of MIMINF and EXE.

Remarks

The MIMINF write-up should be consulted for more detailed information.


MIMINR
START
MIMPAS
1
2
3
4
DO 100 I=1,NUM
YO(I) = Y(I)
TO = T
H = HO
T = T + √2H
J = MIMPAS
DO 220 I=1,NUM
XK(I,J) = H*P(I)
Y(I) = YO(I) + √2*XK(I,J)
RETURN
T = T + √2H
DO 340 I=1,NUM
XK(I,3) = H*P(I)
Y(I) = YO(I) + XK(I,3)
RETURN
DO 450 I=1,NUM
Y(I) = YO(I) + [XK(I,1) + 2(XK(I,2) + XK(I,3)) + H* P(I)/6
MIMPAS = 0
RETURN

5. WMA - Weighting Matrix Routine

Purpose

To provide optional methods for computing the inverse weighting matrices (W^{-1}) and (U^{-1})

Usage

CALL WMA (IENTRY)

This routine follows the entry point pattern of ADJEQ. Several options are available. W^{-1} and U^{-1} are determined by the following inputs:

INDICATOR	NOMINAL	DESCRIPTION
INDWMA	0	Basic option indicator.
IWCYCL	0	Starting matrix indicator.
IWDELT	0	Periodic matrix indicator.
WAI	0.	Basic constant input array. WAI is an array of six values.
WBI	0.	Basic constant input array. WBI is an array of six values.
UMATX	1.	Basic initial control variable input array UMATX is an array of fifteen values.

In the following description:

The subscript i runs from 1 through MCONT.

The subscript j runs from 1 through INDIOP.

The subscript K runs from 1 through MCONT.

MCONT = number of control variables.

INDIOP = number of initial control variables.

$SINTEG_i$ = total integral of $\lambda_{\varphi\Omega} G_i$

t = time.

INDWMA = 0

(a) $W_{i,i}^{-1} = 1.$

(b) $U_j^{-1} = UMATX_j$

INDWMA = 1

(a) when NCYCLE ≤ IWCYCL - 1 :

$$W_{i,i}^{-1} = 1.$$

(b) when NCYCLE > IWCYCL - 1, the W^{-1} matrix is time - varying:

$$W_{i,i}^{-1} = WAI_i + WBI_i \cdot t$$

(c) $U_j^{-1} = UMATX_j$

INDWMA = 2

(a) when NCYCLE ≤ IWCYCL - 1 :

$$W_{i,i}^{-1} = 1.$$

(b) when NCYCLE > IWCYCL - 1 and IWDELT ≠ 0 and NCYCLE is not a multiple of IWDELT:

$$W_{i,i}^{-1} = 1.$$

(c) when NCYCLE > IWCYCL - 1 and IWDELT = 0 or IWDELT ≠ 0 and NCYCLE is a multiple of IWDELT and no previous cycle has been run:

$$W_{i,i}^{-1} = 1.$$

(d) when NCYCLE > IWCYCL - 1 and IWDELT = 0 or IWDELT ≠ 0 and NCYCLE is a multiple of IWDELT and a previous CYCLE has been run:

$$W_{i,i}^{-1} = \frac{WAI_j + WBI_i \cdot \sum_K \left|SINTEG_K\right| \; / \; \left|SINTEG_i\right|}{(MCONT+1)}$$

where $SINTEG_K = \int_T^{t_o} \left|\lambda_{\varphi\Omega} G_K\right| \partial t$ from the previous cycle.

(e) $U_j^{-1} = UMATX_j$

INDWMA = 3

(a) when NCYCLE ≤ IWCYCL - 1

$$W_{i,i}^{-1} = 1.$$

(b) when NCYCLE > IWCYCL - 1 and IWDELT ≠ 0 and NCYCLE is not a multiple of IWDELT:

$$W_{i,i}^{-1} = 1.$$

(c) when NCYCLE ≥ IWCYCL - 1 and IWDELT = 0 or INDELT ≠ 0 and NCYCLE is a multiple of IWDELT, the W^{-1} matrix is time-varying:

$$W_{i,i}^{-1} = \frac{\tilde{W}_i \cdot \bar{W}_i}{2 \cdot (MCONT + 1)}$$

where $\tilde{W}_i = WAI_3 + WAI_4 \cdot \frac{B}{h}$,

$$\bar{W}_i = WAI_1 + WAI_2 \cdot \frac{h}{\left|\lambda_{\varphi\Omega} G_i\right|},$$

B = max value of $\sum_K \lambda_{\varphi\Omega} G_K$ on the previous cycle.

h = instantaneous value of $\sum_K \lambda_{\varphi\Omega} G_K$.

(d) $U_j^{-1} = UMATX_j$

INDWMA = 4

(a) when NCYCLE ≤ IWCYCL - 1

$$W_{i,i}^{-1} = 1.$$

(b) when NCYCLE > INCYCL - 1

$$W_{i,i}^{-1} = \delta_i,$$

where $\delta_i = 0$, unless i = [Remainder upon division of ($\frac{NCYCLE}{IWDELT}$) by MCONT] + 1, in which case $\delta_i = 1$.

This allows the user to work with one control variable at a time, staying with each for IWDELT cycles at a time.

(c) $U_j^{-1} = UMATX_j$

INDWMA = 5

(a) when NCYCLE ≤ IWCYCL - 1, or no previous cycle has been run, the W^{-1} matrix is time-varying:

$$W_{i,i}^{-1} = WAI_i + WBI_i \cdot t$$

(b) when NCYCLE > IWCYCL - 1, and a previous cycle has been run, the W^{-1} matrix is time-varying:

The calculation of this W^{-1} matrix is somewhat complex.

For each new cycle a normalizing factor W' is computed by the formula:

$$W' = \left\{ \sum_i \left| \int_T^{t_o} (W_{i,i}^{-1})^2 \, dt \right| + \sum_j (U_j^{-1})^2 \right\}^{1/2}$$

where $W_{i,i}^{-1}$ and U_j^{-1} are from the previous cycle.

For each stage the quantities $\bar{W}_i$ are computed as an average value for $W_{i,i}^{-1}$ during the stage on the previous reverse.

At each point the quantity $W_i^* = \max\left\{ .2, \min\left\{ 5., \left| 1. - \frac{\lambda_{\varphi\Omega G_i}}{\lambda_i^*} \right| \right\} \right\}$ is computed, where λ_i^* is the value of $\lambda_{\varphi\Omega G_i}$ at this time during the previous reverse. (If CTABUB or CTABLB is being used to bound a control variable and it lies on its boundary and λ_i^* and $\lambda_{\varphi\Omega G_i}$ agree in sign, W_i^* is set to 1.).

For the first point computed in the stage:

$$W_{i,i}^{-1} = \frac{\bar{W}_i}{W' \cdot (W_i^*)^{1/2}}$$

For the (j + 1)st point computed in the stage:

$$W_{i,i}^{-1} = \tilde{W}_i + \frac{W_i^* \cdot \bar{W}_i}{\frac{W'}{j+1}} - \tilde{W}_i$$

where $\tilde{W}_i = W_{i,i}^{-1}$ at the jth point computed.

(c) when NCYCLE < IWCYCL - 1

U_j^{-1} is computed by a formula analogous to that for $W_{i,i}^{-1}$ at the first point in a stage.

INDWMA = 6

(a) $W_{i,i}^{-1} = WAI_i$

(b) $U_j^{-1} = UMATX_j$

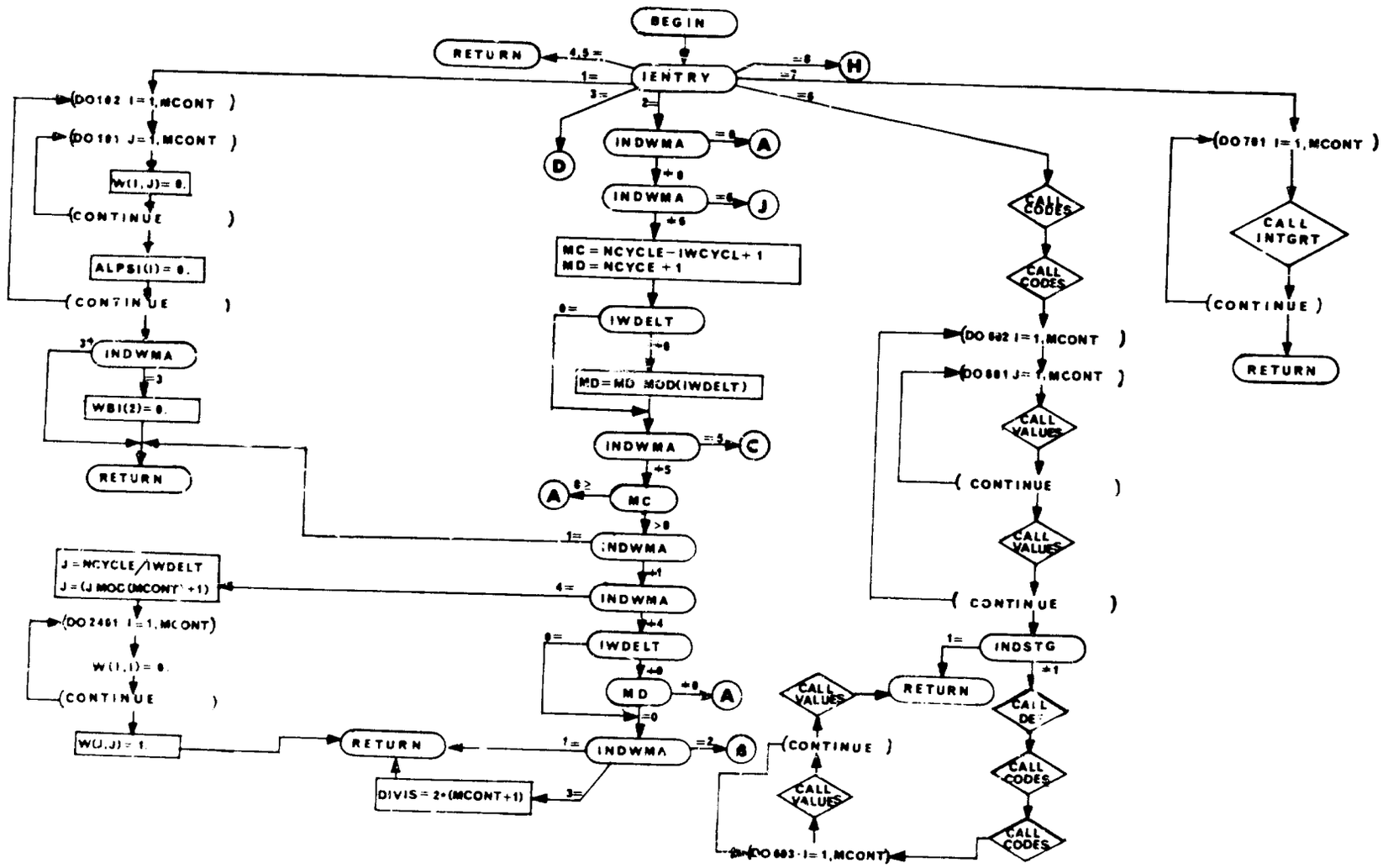
Remarks

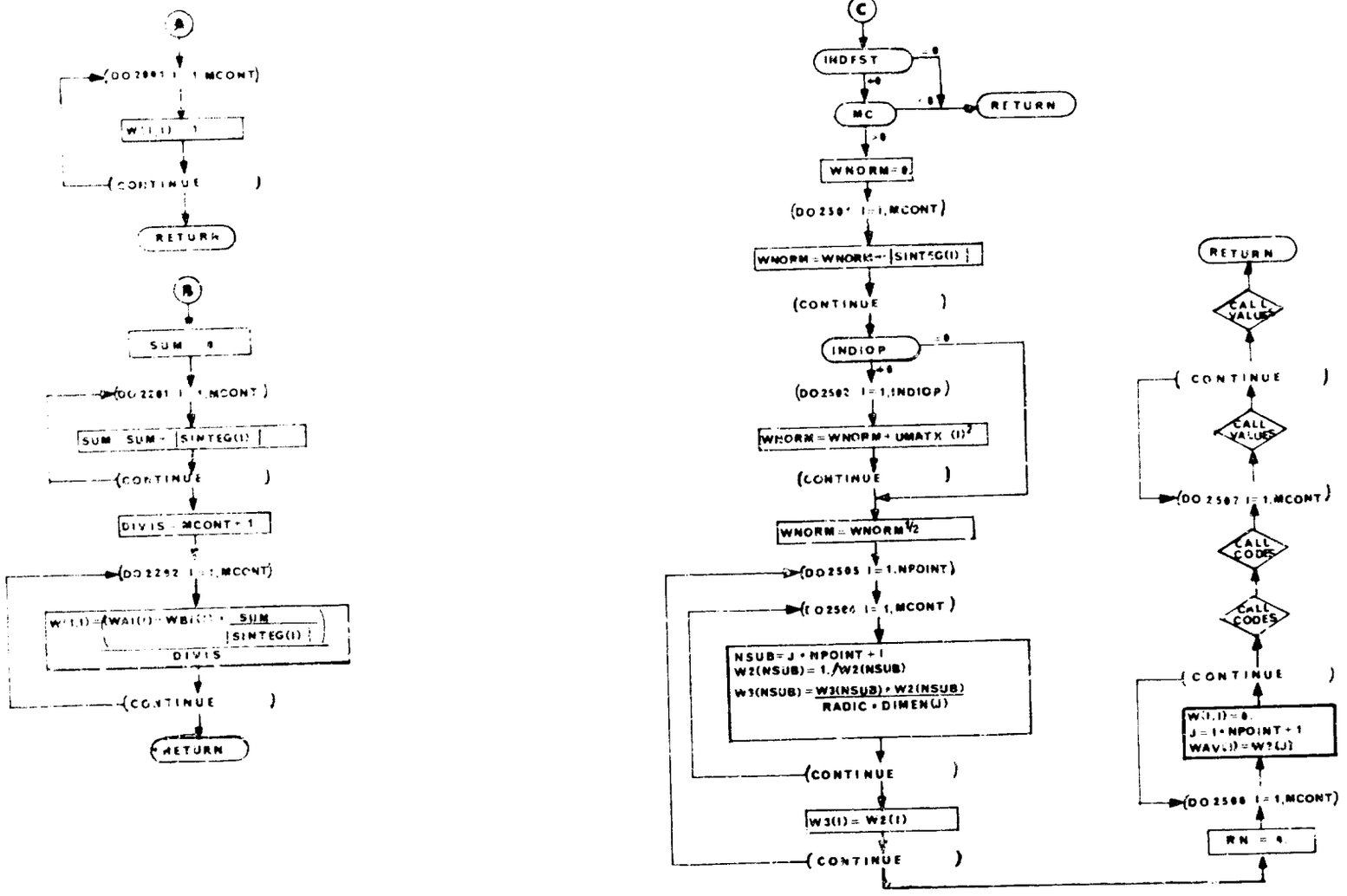
The weighting matrices for INDWMA = 5 are designed for use with the CTLS2 control system only.

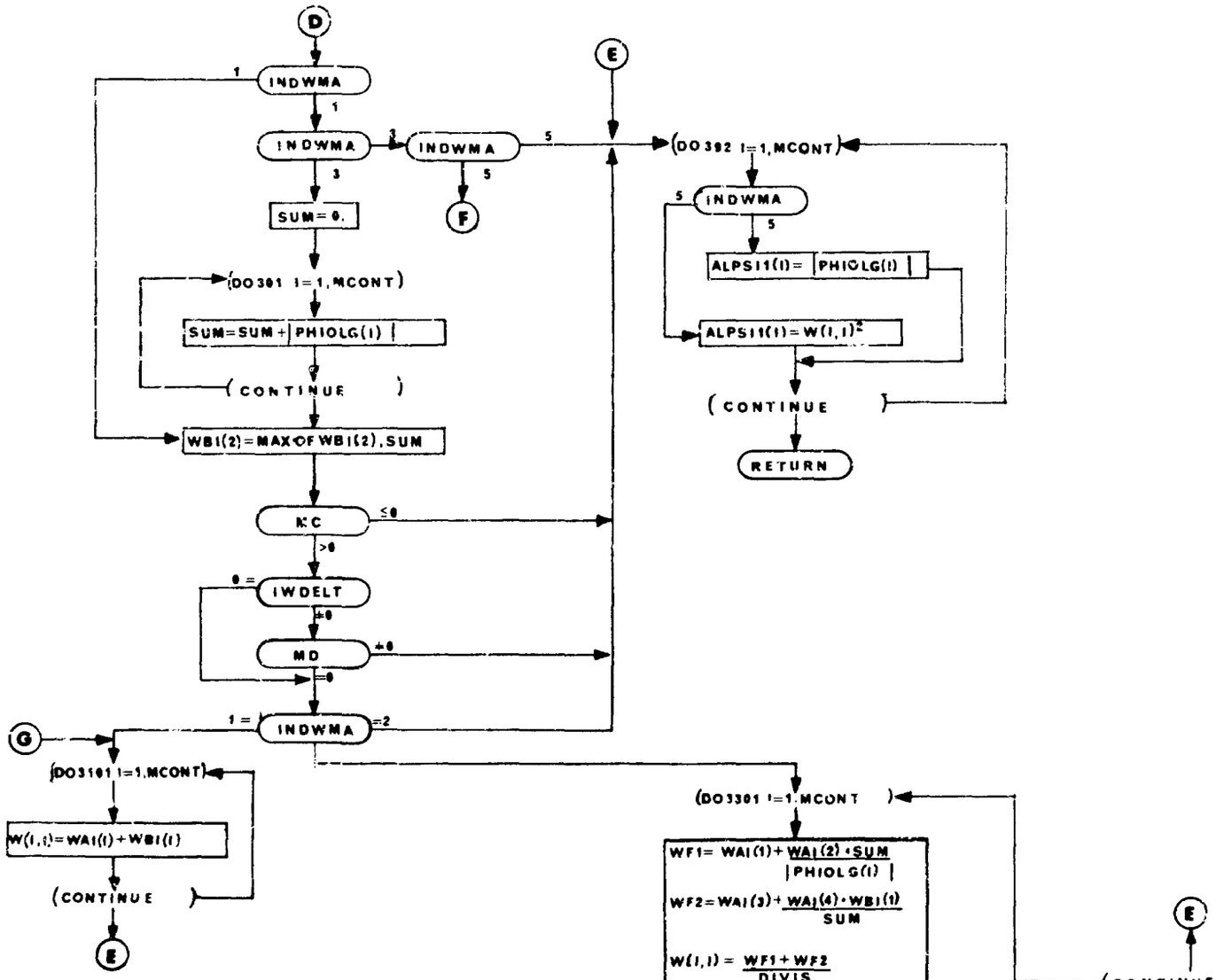
WMA is called by ADJEQ.

WMA calls CODES and VALUES.

WMA (1)


BEGIN
IENTRY
RETURN
INDWMA
MC = NCYCLE-IWCYCL+1
MD = NCYCE +1
IWDELT
MD=MD MOD(IWDELT)
MC
MD
DIVIS = 2*(MCONT+1)
J = NCYCLE/IWDELT
W(I,J)= 0.
ALPSI(I)= 0.
WBI(2)= 0.
W(J,J)= 1.
CALL CODES
CALL VALUES
CONTINUE
INDSTG
CALL INTGRT
(DO 102 I=1,MCONT)
(DO 101 J=1,MCONT)
(DO 2401 I=1,MCONT)
(DO 701 I=1,MCONT)
(DO 602 I=1,MCONT)
(DO 601 J=1,MCONT)
(DO 603 I=1,MCONT)


A
(DO 2001 I = 1,MCONT)
W(I,1) = 1
(CONTINUE)
RETURN
B
SUM = 0
(DO 2201 I = 1,MCONT)
SUM = SUM + |SINTEG(I)|
(CONTINUE)
DIVIS = MCONT + 1
(DO 2292 I = 1,MCONT)
W(I,1) = (WAI(I) + WBI(I)) * SUM / |SINTEG(I)| / DIVIS
(CONTINUE)
RETURN
C
INDFST = 0
MC = 0
RETURN
WNORM = 0
(DO 2501 I = 1,MCONT)
WNORM = WNORM + |SINTEG(I)|
(CONTINUE)
INDIOP = 0
(DO 2502 I = 1,INDIOP)
WNORM = WNORM + UMATX (I)**2
(CONTINUE)
WNORM = WNORM**1/2
(DO 2505 I = 1,NPOINT)
(DO 2504 I = 1,MCONT)
NSUB = J * NPOINT + I
W2(NSUB) = 1./W2(NSUB)
W3(NSUB) = W3(NSUB) * W2(NSUB) / RADIC * DIMEN(J)
(CONTINUE)
W3(I) = W2(I)
(CONTINUE)
RETURN
CALL VALUE
(CONTINUE)
CALL VALUE
(DO 2507 I = 1,MCONT)
CALL CODES
CALL CODES
(CONTINUE)
W(I,I) = 0.
J = I * NPOINT + 1
WAV(J) = W2(J)
(DO 2508 I = 1,MCONT)
RN = 0


D
INDWMA
INDWMA
INDWMA
F
SUM=0.
DO 301 I=1,MCONT
SUM=SUM+|PHIOLG(I)|
CONTINUE
WBI(2)=MAX OF WBI(2), SUM
NC
IWDELT
MD
INDWMA
G
DO 3101 I=1,MCONT
W(I,1)=WAI(I)+WBI(I)
CONTINUE
E
E
DO 392 I=1,MCONT
INDWMA
ALPS11(I)=|PHIOLG(I)|
ALPS11(I)=W(I,1)²
CONTINUE
RETURN
DO 3301 I=1,MCONT
WF1=WAI(1)+WAI(2)·SUM/|PHIOLG(I)|
WF2=WAI(3)+WAI(4)·WBI(1)/SUM
W(I,1)=(WF1+WF2)/DIVIS
CONTINUE
E

6. DALPACK - Blocking Routine for λ's

Purpose:

To store data in large arrays to eliminate unnecessary I/O.

Method:

Data is stored in a large array until it has been filled before being output on tape.

Usage:

Linkage to this routine is accomplished via the statement

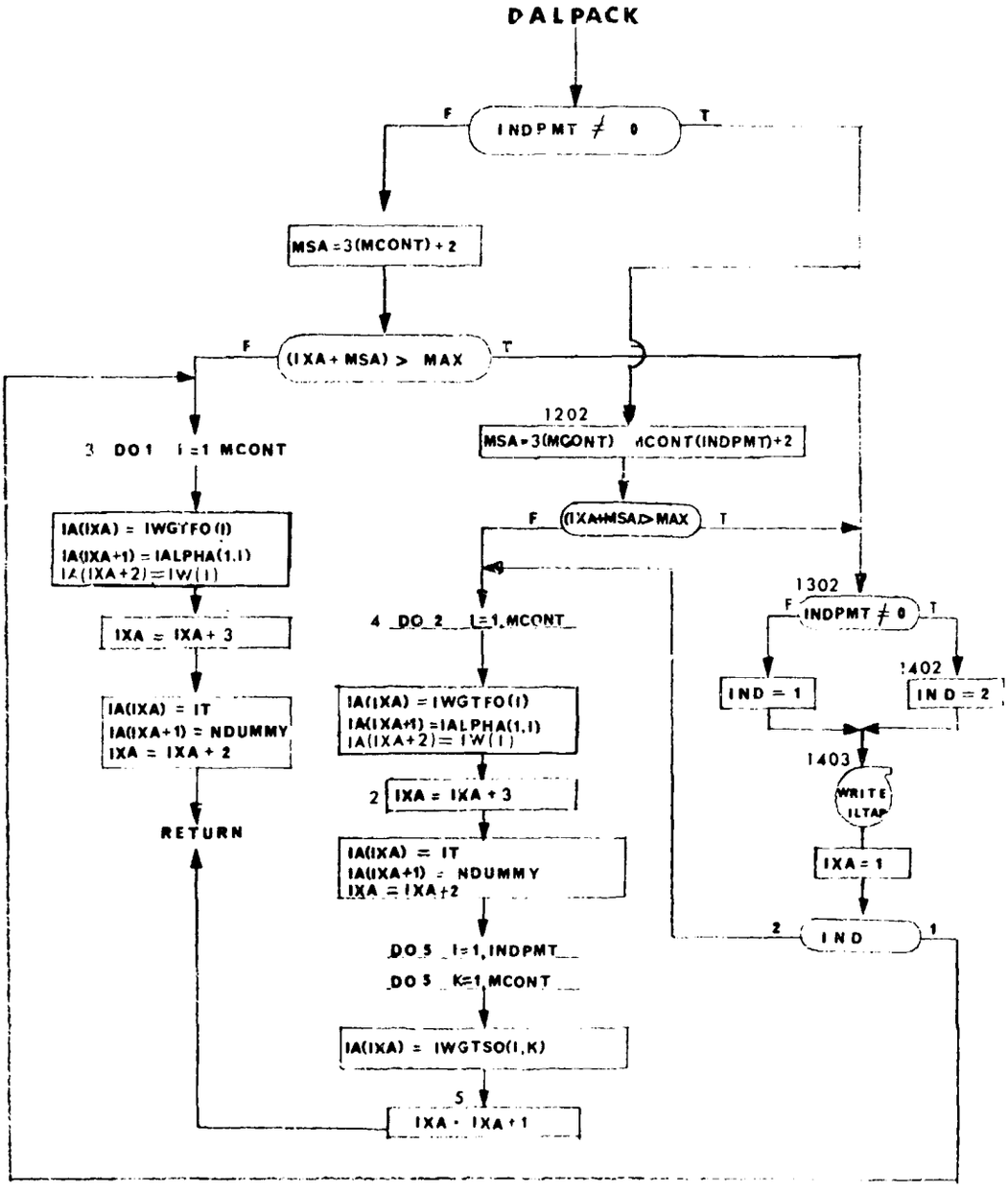
CALL DALPACK (IWGTFO,IT,NDUMMY,INDPMT,MCONT,IWGTSO,IALPHA)

IWGTFO	arrays used in the control program.
IWGTSO	arrays used in the control program.
IT	TIME.
NDUMMY	control word generated in reverse
INDPMT	number of constraints.
MCONT	number of control variables.
IALPHA	alpha point in CTABLE.

Remarks:

IW	Weighting matrix
ILTAP	Tape being used

Normal I/O routines are called.


DALPACK
INDPMT ≠ 0
F
T
MSA=3(MCONT)+2
(IXA+MSA) > MAX
F
T
3 DO 1 I=1 MCONT
IA(IXA) = IWGTFO(I)
IA(IXA+1) = IALPHA(1,I)
IA(IXA+2)=IW(I)
IXA = IXA+3
IA(IXA) = IT
IA(IXA+1) = NDUMMY
IXA = IXA + 2
RETURN
1202
MSA=3(MCONT) MCONT(INDPMT)+2
(IXA+MSA)>MAX
4 DO 2 I=1,MCONT
IA(IXA) = IWGTFO(I)
IA(IXA+1) =IALPHA(1,I)
IA(IXA+2)= IW(I)
2 IXA = IXA+3
IA(IXA) = IT
IA(IXA+1) = NDUMMY
IXA = IXA+2
DO 5 I=1,INDPMT
DO 5 K=1,MCONT
IA(IXA) = IWGTSO(I,K)
5 IXA = IXA+1
1302
INDPMT ≠ 0
1402
IND = 1
IND = 2
1403
WRITE ILTAP
IXA = 1
IND
2
1

7. FLUSH - BUFFER FLUSH ROUTINE

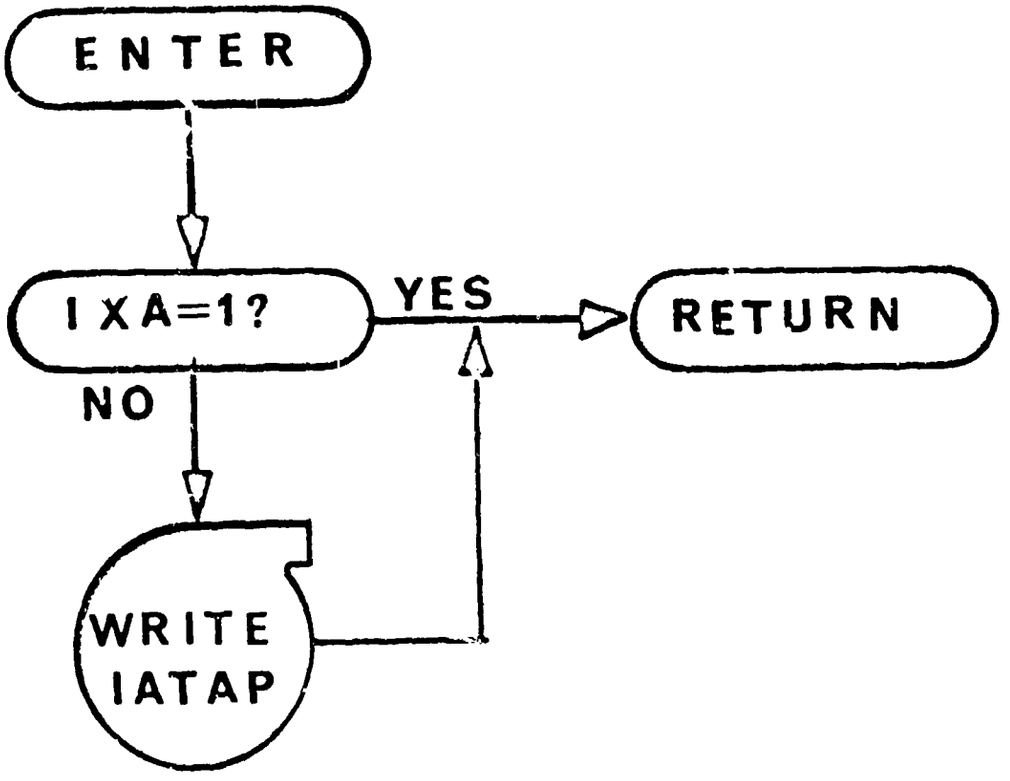
Purpose

To flush the ILTAP buffer at the end of the reverse trajectory.

Usage

CALL FLUSH(ILTAP)

FLUSH



8. IZUNPK - Switch Testing Routine

Purpose:

The routine tests a set of switches to see if the argument should be set to zero or a number should be pulled out of an array.

Method:

A set of switches has been preset by another routine and IZUNPK tests these switches and resets all arrays in proper order.

Usage:

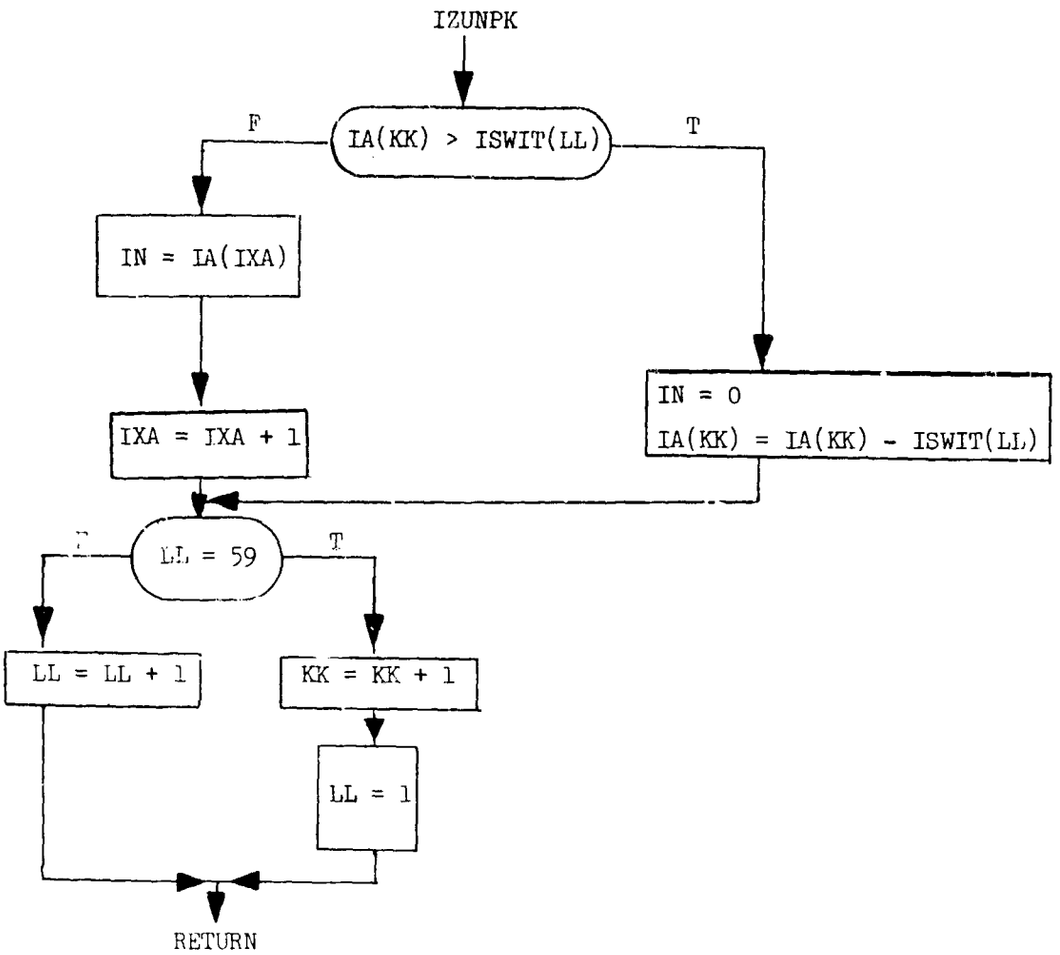
Entry is made to this routine by the following statement:

```
CALL IZUNPK(IA)
```

where

IA = The argument to be set by the routine

No other routines are called by this subroutine


IZUNPK
IA(KK) > ISWIT(LL)
F
T
IN = IA(IXA)
IXA = IXA + 1
IN = 0
IA(KK) = IA(KK) - ISWIT(LL)
LL = 59
F
T
LL = LL + 1
KK = KK + 1
LL = 1
RETURN

SECTION IX

PROGRAM GRAPH

A program has been set up in the program as a data gathering chain for plotting purposes. The CDC 280 plotting package set up at AFWL was used in checking out the program. The basic part of the segment is still in the program. To use the plotting equipment, it is only necessary to provide the additional subroutines, normally available on the 600C series computers, which output information to the 280 for photographing and displaying at the console. A preliminary report on the CDC280 has been included, which gives detailed descriptions of the routines, including purpose, operation, usage and examples.

CDC280

PRELIMINARY REPORT

Some of the routines herein described were originally developed for use at AFWL. Most of the report was taken from "CRT Plotting Routines in use at LRL-Livermore" written by Judith D. Ford and Marilyn J. Welsch (UCRL-14427-T), since many of the routines are the same in name, as well as effect, as those in use at LRL.

ABSTRACT

This report describes a syste of plotting routines. These FORTRAN routines provide a flexible package for point, line, and character plotting via a CDC280 display device.

INTRODUCTION

This paper describes the subroutines available on the 6600 which output o the 280. These routines generate data in a buffer in the 6600 central memory. A peripheral processing unit (PPU) clears the buffer and stores the data on file named -FILMPL-. At the completion of the run a PPU sends this file to the CDC280. The information is then photographed at the recording console (or displayed at the operators option). The film is then developed and made ready for distribution.

This report gives detailed descriptions of the 280 routines, including purpose, operation usage, and examples. The routines are separated into the following classes:

1. Mapping Routines:

These routines set up scale factors for converting the users coordinates to the 280 raster point coordinates (Raster point defined later). These routines may also draw scales with grid lines or short marks along the axes.

2. Arrow, Line, and Point Plotting Routines:

These routines provide the facility for plotting various types of curves.

3. Character Plotting Routines:

These routines provide the facility for plotting alphanumeric information.

4. Absolute Plotting Routines:

These routines position the beam independent of the scaling defined by the mapping routines.

5. Utility Routines:

These routines give the facilities for framing, and initializing the plot package.

6. Internal Routines:

Internal routines perform various functions necessary to the operation of the system, and the user is normally not aware of their existence.

The CDC280 plane is defined to be a (1024 by 1024) square of addressable points on the face of a cathode ray tube (CRT). These points are called raster points. Information is displayed by unblanking the CRT beam. The beam may be moved to a new position without unblanking (i.e. without plotting a line). Points may only be positioned at a raster point. Lines may only be drawn between two raster points (i.e. the beam unblanked between these two raster points may or may not intersect other raster points.

In the following description of the 280 routines, it is assumed that all arguments are given in the same mode as the dummy arguments, using the standard FORTRAN conventions for the names of integer and floating point variables. The dummy arguments spelled -DUM- are not used by the routine. These arguments are reserved in some cases for future options.

For the purposes of these routines this 280 plane is regarded as having the usual X, Y cartesian coordinates, both of which range from 0. to 1. with the origin at the lower left corner. If no mapping routine is called all coordinates for the plotting routines are assumed to be between 0. and 1.

MAPPING ROUTINES

This group of routines makes it unnecessary for the user to scale his own numbers for plotting on the 280. This is accomplished by establishing a mapping from the users coordinate plane onto some portion of the 280 plane. This, by the way, allows more than one graph to be plotted on a frame. CALL MAP (XMIN, XMAX, YMIN, YMAX, XMI, XMA, YMI, YMA). XMIN, XMAX, YMIN, YMAX are the users maximum and minimum cartesian coordinates.

XMI, XMA, YMI, YMA are the maximum and minimum coordinates of the 280 plane desired to be used.

This description encompasses a group of twelve routines, each of which established a mapping from the rectangle in the users plane with corners (XMIN,YMIN), (XMAX,YMAX) onto the rectangle in the 280 plane with corners (XMI,YMI), (XMA,YMA). Unless reset, this mapping applies to all subsequent plotting, except the absolute plotting routines.

Linear mappings are established by -MAP-,-MAPG-, and -MAPS-.

MAP established a mapping only

MAPG plots a grid with scale numbers.

MAPS plots a rectangle with scale numbers and short marks along the axes.

The suffixes -LL-, -SL- and -LS- may be used with any of -MAP-, -MAPG- or -MAPS- to modify the mapping as follows:

LL establishes a log-log mapping.

SL establishes a semi-log mapping with the X-axis linear.

LS establishes a semi-log mapping with the Y-axis linear.

The cycles are determined automatically. Examples:

CALL MAP (0.,1.,0.,1.,0.,1.,0.,1.)
sets up a linear-linear mapping,

CALL MAPSLL (1.,10.,1.,100000.,.1,.999,.1,.999)
sets up a 1 cycle by 5 cycle scale, and

CALL MAPGSL (.100.,10.,1.,100.,.1,.5,.1,.999)
sets up a linear by 2 cycle grid.

The mapping function is initially set

XMIN = YMIN = XMI = YMI = 0. and XMAX = YMAX = XMA = YMA = 1.

The scale numbers will overplot the grid lines if SMI or YMI is less than .078125 for linear scaling or .043 for logarithmic scaling.

Plotting routines specifying point(s) out of the defined user domain are handled in two ways:

1. If the scaled coordinate is within the 280 range then the routine is executed at the scaled coordinate.

2. If the scaled coordinate is outside of the 280 range then this coordinate is projected on the nearest extreme edge and the routine executed there.

An error message is printed whenever a mapping routine is called with XMIN ≥ XMAX, YMIN ≥ YMAX, XMI ≥ XMA, YMI ≥ YMA or a log mapping is called with a non-positive argument.

CALL MAPP(RMAX, XMI, XMA, YMI),

RMAX is the maximum radius for the user's polar coordinates.

XMI, XMA, YMI are the same as in -MAP- above.

-MAPP- establishes a mapping from the circle of radius RMAX in the user's polar coordinate plane into the square in the 280 plane with corners (XMI,YMI), (XMA,YMA) where YMA = YMI + (XMA - XMI).

Vertical and horizontal reference axes will be plotted, with scale numbers along the zero-degree axis, and with the origin at the center of the square. All (X,Y) pairs given in later plotting routines will be interpreted as polar coordinates (R,Θ) until another mapping routine is called.

CALL MAPX (XMIN, XMAX, YMIN, YMAX, XMI, XMA, YMI, YMA, I)

I is an integer $1 \leq I \leq 13$.

The remaining arguments are the same as in -MAP- above.

-MAPX- allows the mapping to be specified at execution time, according to the value of I. A call to -MAPX- is equivalent to a call to one of the above mapping routines, with the integers 1-13 corresponding to these routines in the following order:

MAP, MAPSL, MAPLS, MAPLL, MAPG, MAPGSL, MPAGLS, MAPGLL, MAPS, MAPSSL, MAPSLS, MAPSLL, MAPP.

When I = 13 the arguments in MAPX correspond to MAPP as follows: CALL MAPX (DUM,RMAX,DUM,DUM,XMI,XMA,YMI,DUM,13).

ARROW, LINE AND POINT PLOTTING ROUTINES

Order Alphabetically

These routines may be used to display and/or photograph data in graphic form. The user's (X,Y) coordinates in these plotting routines are scaling the scale factors set up by a mapping routine. If no mapping routine has been called, these coordinates are assumed to be in the range 0. to 1.

CALL SETBEAM (X,Y)

X is the abscissa at which the beam is to be positioned.

Y is the ordinate at which the beam is to be positioned.

-SETBEAM- causes the beam to be positioned at (X,Y) without unblanking.

CALL LINEOPT (DUM,IN TEN)

DUM is a dummy argument.

INTEN is the intensity at which all arrows, lines, points, and vectors will be plotted.

0 low intensity (fine line).

1 high intensity (heavy line).

-LINEOPT- is initially set to low intensity.

The mapping routines reset -LINEOPT- from within.

CALL POINT (X,Y)

X and Y are the coordinates of a point.

-POINT- will plot a point at (X,Y) with intensity set by -LINEOPT-.

CALL LINE (X1,Y1,X2,Y2)

(X1,Y1) and (X2,Y2) are the coordinates of two points. -LINE- will sweep a line from (X1,Y1) to (X2,Y2) with intensity set by -LINEOPT-.

CALL LINEP (X1,Y1,X2,Y2,K)

(X1,Y1) and (X2,Y2) are the coordinates of two points.

K is an integer.

-LINEP- plots a line consisting of every Kth raster point between (X1,Y1) and (X2,Y2). Intensity is set by LINEOPT

CALL VECTOR (X2,Y2)

(X2,Y2) is the coordinate of a point.

-VECTOR- sweeps a line from the current beam position to (X2,Y2) with intensity set by LINEOPT.

CALL POINTS (X,Y,N)

X and Y are the names (first word addresses) of arrays of the X and Y coordinates of points.

N is the number of points.

-POINTS- plots the N points given by the arrays X and Y. The intensity is set by -LINEOPT-.

CALL ARROW (X1, Y1, X2, Y2, I)

(X1,Y1) and (X2,Y2) are coordinates of two points.

I is an integer ≥ 1.

-ARROW- sweeps a line from (X1,Y1) to (X2,Y2) and draws an arrowhead at (X2,Y2). The arrowhead measures I raster points in length. I = 10 is a normal size arrowhead. The intensity is set by -LINEOPT-. The final beam position is (X2,Y2).

CALL LINES (X,Y,N)

X and Y are the names (first word addresses) of arrays of the X and Y coordinates of points.

N is the number of points.

-LINES- connects the N points given by the arrays X and Y with line segments. The final beam position is (X(N),Y(N)). The lines are swept with intensity as set by -LINEOPT-.

Order Alphabetically

CHARACTER PLOTTING ROUTINES

This group of routines allows the plotting of alphanumeric information, either to label the various curves, lines, etc., produced by the point and line plotting routines, or as more versatile alternative to an off-line printer (this is distinct from the -FILMPR- option. -FILMPR- merely simulates the printer). This versatility derives from:

1. The capability of positioning a line of alphanumeric information anywhere on the current frame (vs. the top-to-bottom progression of a page printer).

2. The two orientations, two intensities and four character sizes that are available, and

3. The expanded character set, which includes many non-key punchable characters. (not immediately available)

CALL CHAROPT (DUM,DUM,ISIZE,IOR,DUM)

ISIZE = 0 miniature

1 small

2 medium

3 large

IOR = 0 horizontal (0°)

1 vertical (90°)

-CHAROPT- specifies the size (ISIZE) and orientation (IOR) of all characters to be plotted. The option is changed by a second call to -CHAROPT-.

The maximum string length and line limits for the various sizes are:

	Symbols/Line	Lines/Frame
miniature	137	64
small	86	43
medium	64	32
large	43	22

In the character plotting routines, the 280 plane is considered to be a grid of rectangles, each containing one character of the chosen size. The number and dimensions of these rectangles depend on the character size and orientation. Characters are drawn within the rectangle. The rectangle is positioned such that the current beam position is in the lower left corner of the rectangle. After the character has been drawn the beam is positioned in the lower right corner of the rectangle.

CALL SYMBOL (A) or CALL SYMBOL (MH...$.)

A is the first word of BCD data. The end of string is designated by $.

MH...$. is a Hollerith text of M characters. The last two characters must be $., which designates the end of string.

-SYMBOL- encodes BCD data into the 280 character set and plots it starting at the current beam position with options as given by -CHAROPT-.

If $. does not appear at the end of string -SYMBOL- attempts to plot words up to the field length.

CALL NUMBER (X,F)

X is a variable (fixed or floating).

F is any allowable FORTRAN format ≤ 10 characters.

-NUMBER- converts the variable X under the given format, determines the field width and plots the resulting characters as -SYMBOL- would.

Example:

```
.
.
.
X = 1.E5

CALL NUMBER (X,5HE10.2)
.
.
.
```

would plot

```
bbb1.00E05
```

and

```
.
.
.
I = 42
CALL NUMBER (1, 9H4HIN = ,I3)
.
.
.
```

would plot

```
INb = b42
```

ABSOLUTE PLOTTING ROUTINES

Order Alphabetically

These routines position the beam independently of the defined mapping function. The arguments range from 0. to 1. Out of range points are projected on the nearest extreme edge of the plotting areas.

```
CALL ABSBEAM (X,Y)
```

X,Y are coordinates of a point.

-ABSBEAM- causes the beam to be positioned at (X,Y) without unblanking.

```
CALL ABSVECT (X,Y)
```

X,Y are coordinates of a point,

-ABSVECT- draws a vector from the last beam position to (X,Y).

```
CALL ABSLINE (X1, Y1, X2, Y2)
```

(X1,Y1) , (X2,Y2) are coordinates of two points.

-ABSLINE- draws a line from (X1,Y1) to (X2,Y2).

```
CALL ABSPT (X,Y)
```

X,Y are coordinates of a point.

-ABSPT- plots a point at (X,Y).

UTILITY ROUTINES

CALL INIT280

-INIT280- initializes the 280 routines and must be called before any of the plotting routines.

CALL FRAME

-FRAME- advances the microfilm to the next blank frame.

INTERNAL ROUTINES

These routines are essential to the plotting routines, but are not called directly by the user, only by other routines in the system. -GRID80- is called by the mapping routines which draw scale marks or grid lines and label them with scale numbers.

-GTRF-, -GEQF-, -EQLF-, -SEQF-, -SMLF-, -UNQF-, and -ZGTRF- are functions which are used in -GRID80-. Each has two arguments and returns a value of 1 if the first argument stands in the indicated relation to the second, a value of 0 otherwise.

FUNCTION	RELATION
GTRF	greater than
GEQF	greater than or equal to
EQLF	equal to
SEQF	less than or equal to
SMLF	less than
UNQF	not equal to

These functions call -ZGTRF- to establish the value.

-TEST- is called by the mapping routines to establish legal arguments.

-ADJUST- is called by some of the plotting routines to convert non-linear arguments to linear before scaling.

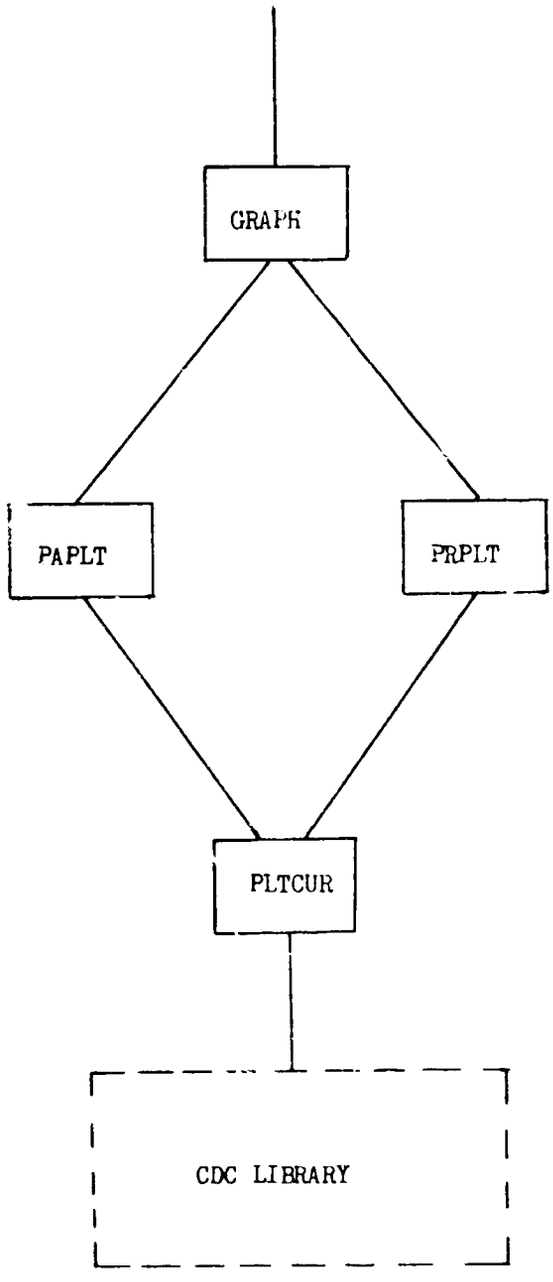
-LENGTH- is called by number to count the number of characters to be plotted.

-STREND- called by symbol to test for end of string symbol.

-PSCALE- is called by the mapping routines to establish the scaling.

-PLOTQ- is called by the plotting routines and forms the 280 instructions.

ORGANIZATIONAL CHART FOR GRAPH



Usage

This is the segment for plotting curves from the forward trajectory on microfilm. GRAPH is called by CTLS before the call is made to REV. Therefore when GRAPH is finished it calls REV. However CTLS calls GRAPH if and only if there are plots to be made. The number of core locations reserved for setting up the tables for plotting may vary depending on the available core. In general, this array (POINTS) should be dimensioned as large as possible, so that GRAPH takes up almost as much core as the largest of the remaining segments. MAXPOT is the value at which POINTS is dimensioned. To vary the dimension of POINTS it is necessary only to change the dimension statement for POINTS and the data statement for MAXPOT in GRAPH proper.

GRAPH drives the plotting computations by calling PAPLT to plot the regular trajectory curves (those specified in the PLOTS array) and PRPLT to plot the F and/or G time history.

DATA FOR PLOTTING

Two types of plotting are possible — printing along with the normal output, or by plotting on microfilm.

Printer Plotting: (done in CTLS segment from DALCAL)

Example: INDPLT INT 39,39

The α and/or $\delta\alpha$ stage time history are plotted for each major stage for the respective control variables if the corresponding two digit integer of the INDPLT array is non-zero and if the second digit is a multiple of the cycle number. The first digit may be either 1, 2, or 3:

1 - α history only (and W^{-1} history if INDWMA = 5)
2 - $\delta\alpha$ history only (and W^{-1} history if INDWMA = 5)
3 - both α and $\delta\alpha$ history (and W^{-1} history if INDWMA = 5)

Microfilm Plotting: (done in GRAPH SEGMENT)

Example:

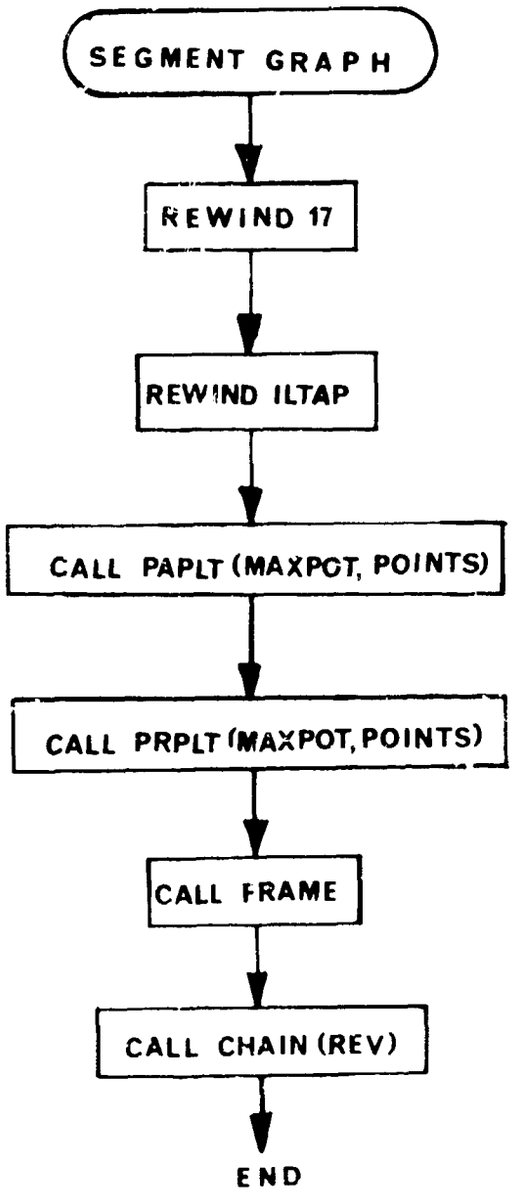
PLOTS BCD 6 TIME^{bb}HGC77F^{b}ALPHD^{b}AMACH^{b}TIME^{bb}GAM7D

INDFAG 3

IGCONT 2

The definition of the abscissa and ordinate variable of each plot are listed successively on the PLOTS card (abscissa, ordinate, abscissa, ordinate, etc.). A plot of up to six passes of each cycle will be made for a given curve, all passes for a cycle for a given curve will appear on the same graph. Values are saved on tape for plotting at every (IGCONT+1)th valid integration step.

INDFAG
0 do not plot partials
1 plot the F matrix vs. time
2 plot the G matrix vs. time
3 plot both the F matrix and the G matrix vs. time.


SEGMENT GRAPH
REWIND 17
REWIND ILTAP
CALL PAPLT (MAXPOT, POINTS)
CALL PRPLT (MAXPOT, POINTS)
CALL FRAME
CALL CHAIN (REV)
END

7. PAPLT - Parameter Collection Routine

Purpose:

To read TAPE17, assemble the points to be plotted into arrays and call the routine to do the actual plotting (PLTCUR).

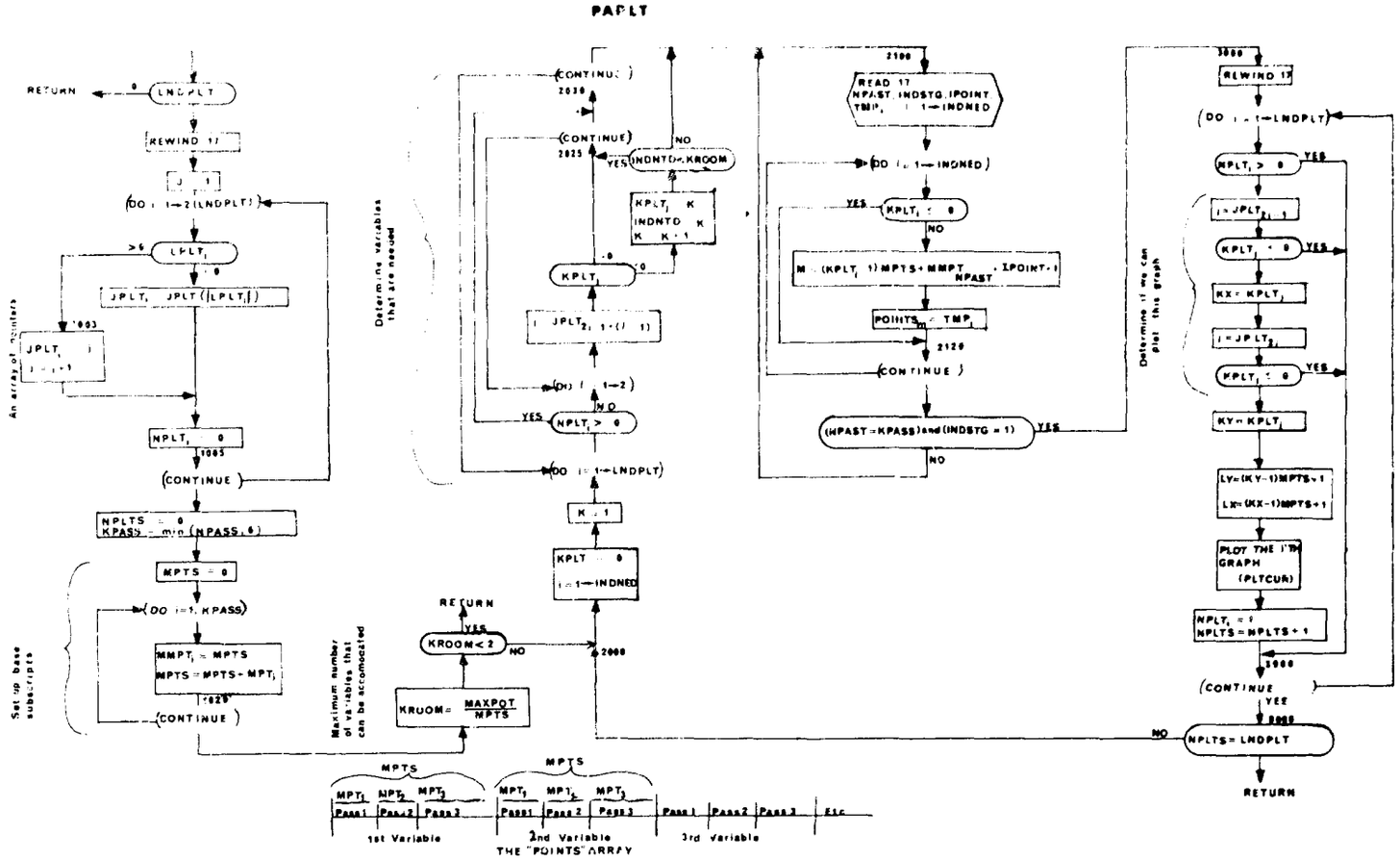
Method:

TAPE17 will be rewound and re-read if there is not enough room to accommodate all the plots at one time. Hence, if there is room to accommodate just one plotting array, all plots will eventually be made; if there is not room for one plot then no plots will be made. PAPLT must be compatible with PLTS of the EXE segment and with the initialization done in MAIN1 for the plotting variables. The format of the POINTS array appears on the flow chart; PAPLT fills this array as it reads TAPE 17 until the last point of the last pass of the forward trajectory is reached. PAPLT does not fill up the POINTS array with those plots that have already been made (i.e. on previous passes through TAPE17). Also, PAPLT does not fill up the POINTS array with those plots for which it does not have room on the current pass through TAPE17.

The following is a description of the important variables that are used internally to the PAPLT routine:

INDNTD	the number of plots to be made on a given read through TAPE17
JPLT	an array of indicators; a function to the LPLT array
KROOM	the maximum number of plots that will fit into the POINTS array
MMPT	an array of base subscripts for each pass (for the points array)
NPAST	the pass number read from tape
IPOINT	the point number read from tape
KPASS	min (NPASS,6)
NPLT	an array to indicate if the i'th plot has been made
KPLT	array of indicators; may change for each read through TAPE17
NPLTS	the number of plots that have been made

For each plot that is made, there will be KPASS curves appearing on the same graph, one for each pass of the forward trajectory. The plots to be made are defined completely in the data on the PLOTS card.


PAPLT
RETURN
LNDPLT
REWIND 17
DO I = 1→2(LNDPLT)
LPLT
JPLT
An array of pointers
NPLT = 0
CONTINUE
NPLTS = 0
KPASS = min (NPASS, 6)
MPTS = 0
DO I=1, KPASS
MMPT = MPTS
MPTS = MPTS + MPT
CONTINUE
Set up base subscripts
CONTINUE
Determine variables that are needed
INDNTO = KROOM
KPLT = K
INDNTO = K
K = K + 1
NPLT > 0
DO I = 1→LNDPLT
K = 1
KPLT = 0
I = 1→INDNED
RETURN
KROOM < 2
KROOM = MAXPOT / MPTS
Maximum number of variables that can be accomodated
READ 17
NPAST, INDSTG, IPOINT
TMP, I = 1→INDNED
DO I = 1→INDNED
KPLT = 0
M = (KPLT - 1) MPTS + MMPT + IPOINT - 1
NPAST
POINTS = TMP
CONTINUE
(NPAST = KPASS) and (INDSTG = 1)
Determine if we can plot this graph
REWIND 17
DO I = 1→LNDPLT
NPLT > 0
I = JPLT
KPLT = 0
KX = KPLT
I = JPLT
KPLT = 0
KY = KPLT
LY = (KY - 1) MPTS + 1
LX = (KX - 1) MPTS + 1
PLOT THE I'TH GRAPH (PLTCUR)
NPLT = 0
NPLTS = NPLTS + 1
CONTINUE
NPLTS = LNDPLT
RETURN
MPTS
MPT1 MPT2 MPT3
Pass 1 Pass 2 Pass 3
1st Variable
2nd Variable
3rd Variable
Etc
THE "POINTS" ARRAY

8. PRPLT - Partials Collection Routine

Purpose:

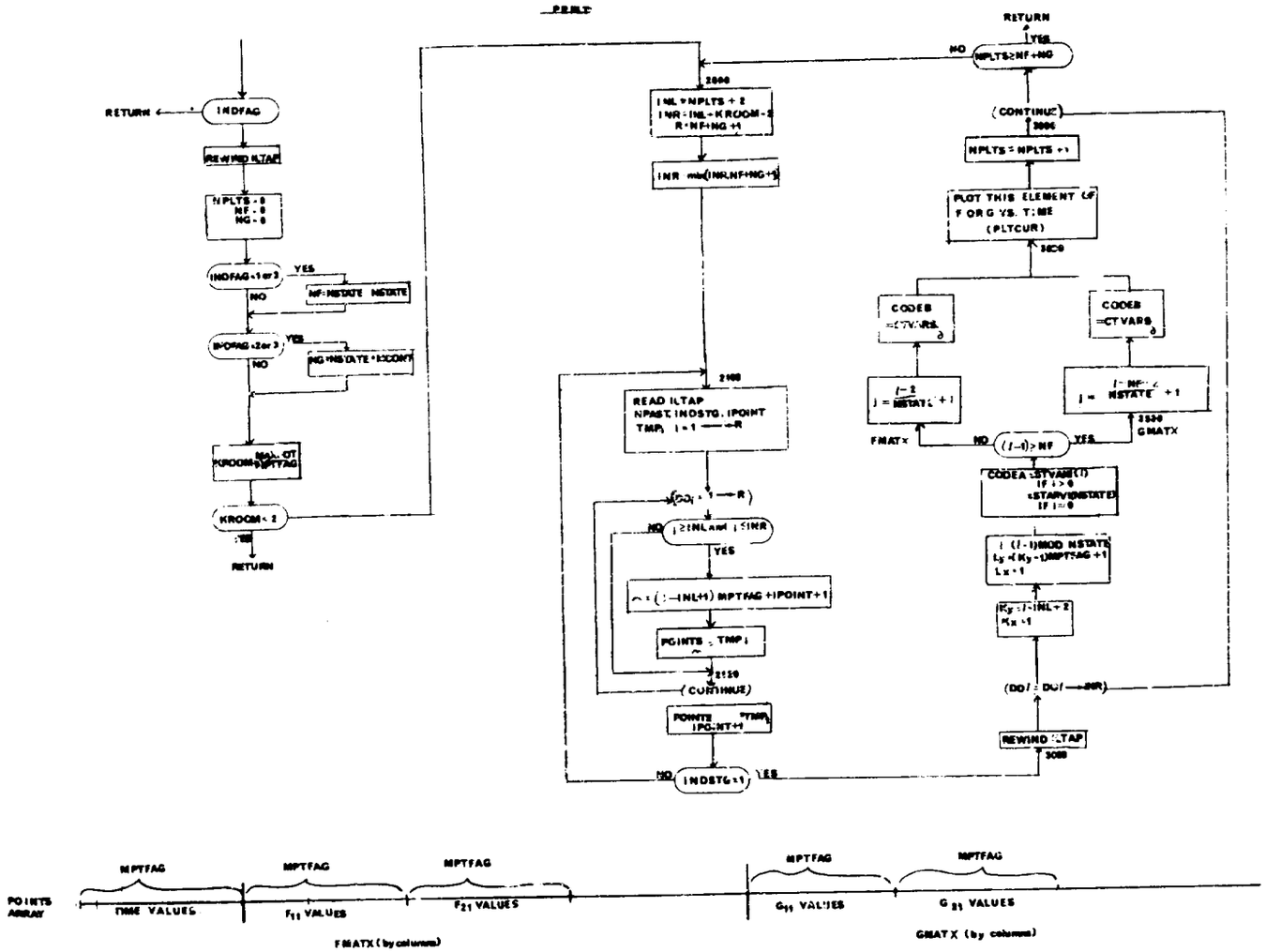
To read ILTAP, assemble the points to be plotted into arrays, and call the routine to do the actual plotting (PLTCUR). ILTAP contains the time histories of the F and/or G matrices.

Method:

The organization of this routine is very similar to that of PAPLT. The differences that do exist make this the less complicated routine. The elements of the F and/or G matrices are always plotted as a function of time. PRPRLT will make several passes through tape ILTAP in order to get all the plots if it is necessary to do so. For each element of the F and G matrix one plot will be made (consisting of just one curve). The format of the POINTS array appears on the flow chart; it is different than that from the PAPLT routine.

In addition to the variables described for PAPLT, the following variable definitions are important:

NF	number of plots to make from the F matrix.
NG	number of plots to make from the G matrix.
INL, INR	respectively, the left and right limit subscripts for the array that is read from tape; any variable whose subscript in the array lies within the interval [INL,INR] will be plotted for the current read through tape ILTAP.


PRINT
RETURN
INDFAG
REWIND ILTAP
NPLTS = 0
NF = 0
NG = 0
INDFAG = 1 or 3
YES
NO
NF = NSTATE NSTATE
INDFAG = 2 or 3
NG = NSTATE * MCONT
KROOM = MAX. OT MPTFAG
KROOM < 2
RETURN
INL = NPLTS + 2
INR = INL + KROOM - 2
R = NF + NG + 1
INR = min(INR, NF + NG + 1)
READ ILTAP
NPAST, INDSTG, IPOINT
TMP, I = 1 → R
(DO I = 1 → R)
I ≥ INL and I ≤ INR
YES
(I - INL + 1) MPTFAG + IPOINT + 1
POINTS = TMP I
(CONTINUE)
POINTS = TMP I
IPOINT + 1
INDSTG = 1
NO
YES
RETURN
NPLTS ≥ NF + NG
(CONTINUE)
NPLTS = NPLTS + 1
PLOT THIS ELEMENT OF F OR G VS. TIME (PLTCUR)
CODEB = CTVARS
CODEB = CTVARS
FMATX
GMATX
(I - 1) > NF
CODEA = STVAR(I)
IF I > 0
= STARV(NSTATE)
IF I = 0
I = (I - 1) MOD NSTATE
Ly = (Ky - 1) MPTFAG + 1
Lx = 1
Ky = I - INL + 2
Kx = 1
(DO I = DO I → INR)
REWIND ILTAP
POINTS ARRAY
MPTFAG
TIME VALUES
F11 VALUES
F21 VALUES
FMATX (by columns)
G11 VALUES
G21 VALUES
GMATX (by columns)

9. PLTCUR - Microfilm Plotting Routine

Purpose:

To plot a set of curves on one graph; the output is to be to microfilm.

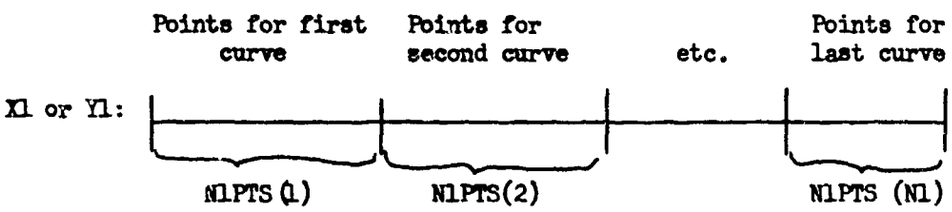
Usage:

CALL PLTCUR (N1,N1PTS,X1,Y1,CODEX,CODEYA,CODEYB)

where

N1	Number of curves to be plotted on a graph.
N1PTS	An array containing the number of points for the respective curves.
X1	An array containing the abscissa values for each curve.
Y1	An array containing the ordinate values for each curve.
CODEX	A Hollerith code word for identifying the abscissa; this will be placed under the abscissa axis on the graph.
CODEYA, CODEYB	Hollerith code words for identifying the ordinate; these will be placed along the ordinate axis on the graph

The format of the X1 and Y1 arrays must conform to the following standard:

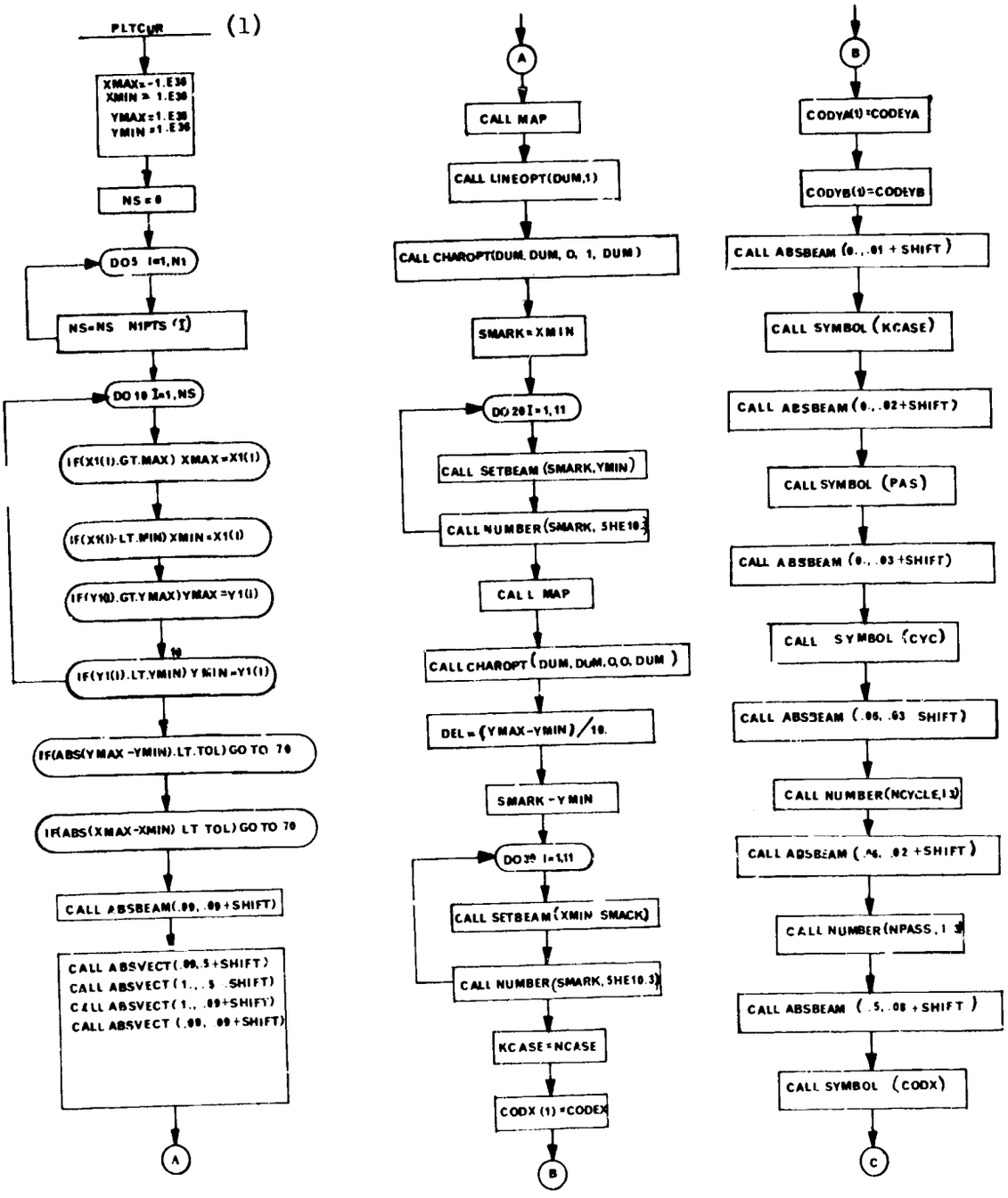


In addition to identifying each plot with the Hollerith code words from the calling sequence, PLTCUR will put the cycle number, the pass number and the case code in the lower left hand corner of the graph. PLTCUR will put two graphs on a frame; for any given call to PLTCUR the graph for that call will be put at one of these positions in the frame depending on the value of the variable SHIFT. If SHIFT=0 the graph is placed in the first half of a frame; if SHIFT=.5 the graph is placed in the second half. PLTCUR updates SHIFT to the appropriate value and changes frames when necessary just before the return is made to the calling program.

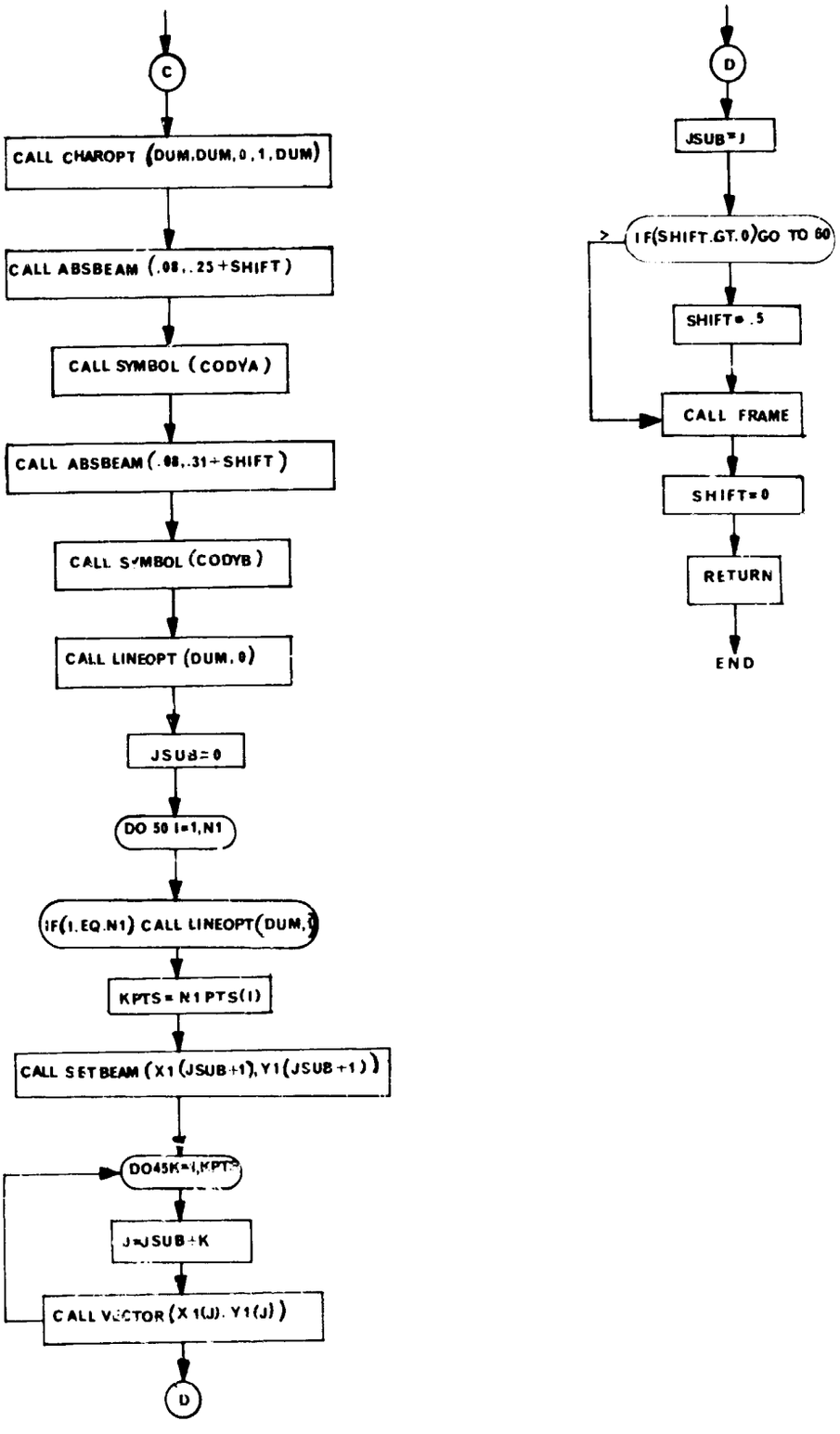
All curves on a given graph will be made in low intensity except the last which will appear in high intensity.

PLTCUR calls a number of routines which must exist in the library "microfilm plot package."

A graph will not be plotted by PLTCUR if the maximum deviation within the X1 or Y1 arrays is less then 10^{-4}.


PLTCUR
(1)
XMAX=-1.E36
XMIN = 1.E36
YMAX=1.E36
YMIN =1.E36
NS = 0
DO 5 I=1,N1
NS=NS N1PTS (I)
DO 10 I=1,NS
IF(X1(I).GT.MAX) XMAX=X1(I)
IF(X1(I)-LT.MIN) XMIN=X1(I)
IF(Y1(I).GT.YMAX)YMAX=Y1(I)
10
IF(Y1(I)).LT.YMIN) YMIN=Y1(I)
IF(ABS(YMAX-YMIN).LT.TOL) GO TO 70
IF(ABS(XMAX-XMIN) LT TOL) GO TO 70
CALL ABSBEAM(.09, .09+SHIFT)
CALL ABSVECT(.09,.5+SHIFT)
CALL ABSVECT(1.,.5 .SHIFT)
CALL ABSVECT(1., .09+SHIFT)
CALL ABSVECT (.09, .09+SHIFT)
A
A
CALL MAP
CALL LINEOPT(DUM,1)
CALL CHAROPT(DUM, DUM, 0, 1, DUM)
SMARK=XMIN
DO 20 I=1,11
CALL SETBEAM (SMARK,YMIN)
CALL NUMBER(SMARK, 5HE10.3)
CALL MAP
CALL CHAROPT (DUM, DUM, 0,0, DUM)
DEL=(YMAX-YMIN)/10.
SMARK=YMIN
DO 30 I=1,11
CALL SETBEAM(XMIN SMACK)
CALL NUMBER(SMARK, 5HE10.3)
KCASE=NCASE
CODX (1) =CODEX
B
B
CODYA(1)=CODEYA
CODYB(1)=CODEYB
CALL ABSBEAM (0.,.01 + SHIFT)
CALL SYMBOL (KCASE)
CALL ABSBEAM (0.,.02+SHIFT)
CALL SYMBOL (PAS)
CALL ABSBEAM (0., .03+SHIFT)
CALL SYMBOL (CYC)
CALL ABSBEAM (.06, .03 SHIFT)
CALL NUMBER(NCYCLE,I3)
CALL ABSBEAM (.06, .02 +SHIFT)
CALL NUMBER(NPASS, I 3)
CALL ABSBEAM (.5, .08 +SHIFT)
CALL SYMBOL (CODX)
C

PLTCUR (2)


C
CALL CHAROPT (DUM,DUM,0,1,DUM)
CALL ABSBEAM (.08,.25+SHIFT)
CALL SYMBOL (CODYA)
CALL ABSBEAM (.08,.31+SHIFT)
CALL SYMBOL (CODYB)
CALL LINEOPT (DUM,0)
JSUB=0
DO 50 I=1,N1
IF(I.EQ.N1) CALL LINEOPT(DUM,
KPTS=N1PTS(I)
CALL SETBEAM (X1(JSUB+1),Y1(JSUB+1))
DO45K=1,KPTS
J=JSUB+K
CALL VECTOR (X1(J),Y1(J))
D
D
JSUB=J
IF(SHIFT.GT.0)GO TO 60
SHIFT=.5
CALL FRAME
SHIFT=0
RETURN
END

10. Dummy Plot Routines

The following dummy routines are included in the deck so that it may be compiled on a machine without attached CDC 280. When using a machine with attached CDC 280 these routines should be replaced by the library routines.

```
      SUBROUTINE ABSBEAM(A,B)                                          ABSB0001
      RETURN                                                           ABSB0002
      END                                                              ABSB0003


      SUBROUTINE ABSVECT(A,B)                                          ABSV0001
      RETURN                                                           ABSV0002
      END                                                              ABSV0003


      SUBROUTINE MAP(A,B,C,D,E,F,G,H)                                  MAP00001
      RETURN                                                           MAP00002
      END                                                              MAP00003


      SUBROUTINE LINEOPT(A,B)                                          LINE0001
      RETURN                                                           LINE0002
      END                                                              LINE0003


      SUBROUTINE CHAROPT(A,B,C,D,E)                                    CHAR0001
      RETURN                                                           CHAR0002
      END                                                              CHAR0003
```

```
      SUBROUTINE NUMBER(A,B)                                          NUMB0001
      RETURN                                                          NUMB0002
      END                                                             NUMB0003


      SUBROUTINE SYMBOL(A)                                            SYMB0001
      RETURN                                                          SYMB0002
      END                                                             SYMB0003


      SUBROUTINE SETBEAM(A,B)                                         SETB0001
      RETURN                                                          SETB0002
      END                                                             SETB0003


      SUBROUTINE FRAME                                                FRAM0001
      RETURN                                                          FRAM0002
      END                                                             FRAM0003


      SUBROUTINE VECTOR(A,B)                                          VECT0001
      WRITE(6,1000) A,B                                               VECT0002
 1000 FORMAT( 1H ,10X,1PE15.7,5X,1PE15.7)                             VECT0003
      RETURN                                                          VECT0004
      END                                                             VECT0005
```

SECTION X

DIRECTORY

Purpose:

To provide a method of reading input symbolically. This subroutine (Blocked Data) has no executable instructions in it. It is designed merely as a means of getting data symbols loaded with the program. No flow chart accompanies this write-up.

Method:

The format of the directory is described here. The directory is divided into two parts: (1) one list is a list of the BCD symbols. (2) the second part is made up of a set of subscripts which corresponds to the BCD words.

For each variable name desired in the directory there must be a corresponding subscript. Thus, each variable in the directory must be in COMMON in order that a subscript may be assigned at assembly time.

Usage:

This subroutine may not be "USED" by the programmer. Rather, it is referred to by subroutines.

1. CTAE - Parameter Optimization Control Program

Purpose:

To control trajectory and optimization calculation sequence when the parameter optimization option is used.

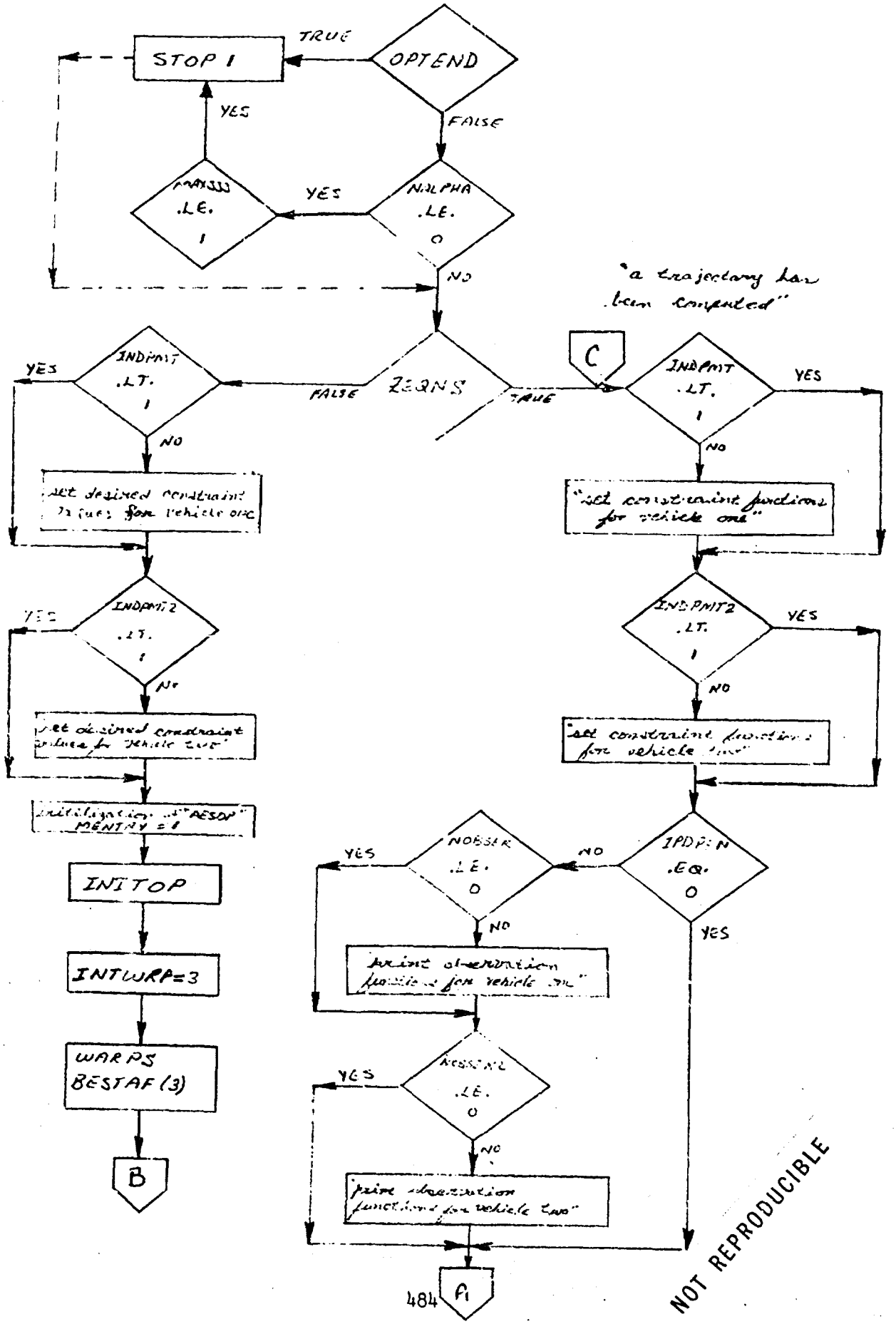
Method:

The parameter optimization algorithms define a control parameter vector. The control parameter vector elements are equated to specific trajectory data input parameters which have been flagged through the "PAR" input option in READA. A trajectory is generated by a call to CHAIN(2), and the resulting trajectory payoff and constraint functions are retrieved for use in the parameter optimization procedure.

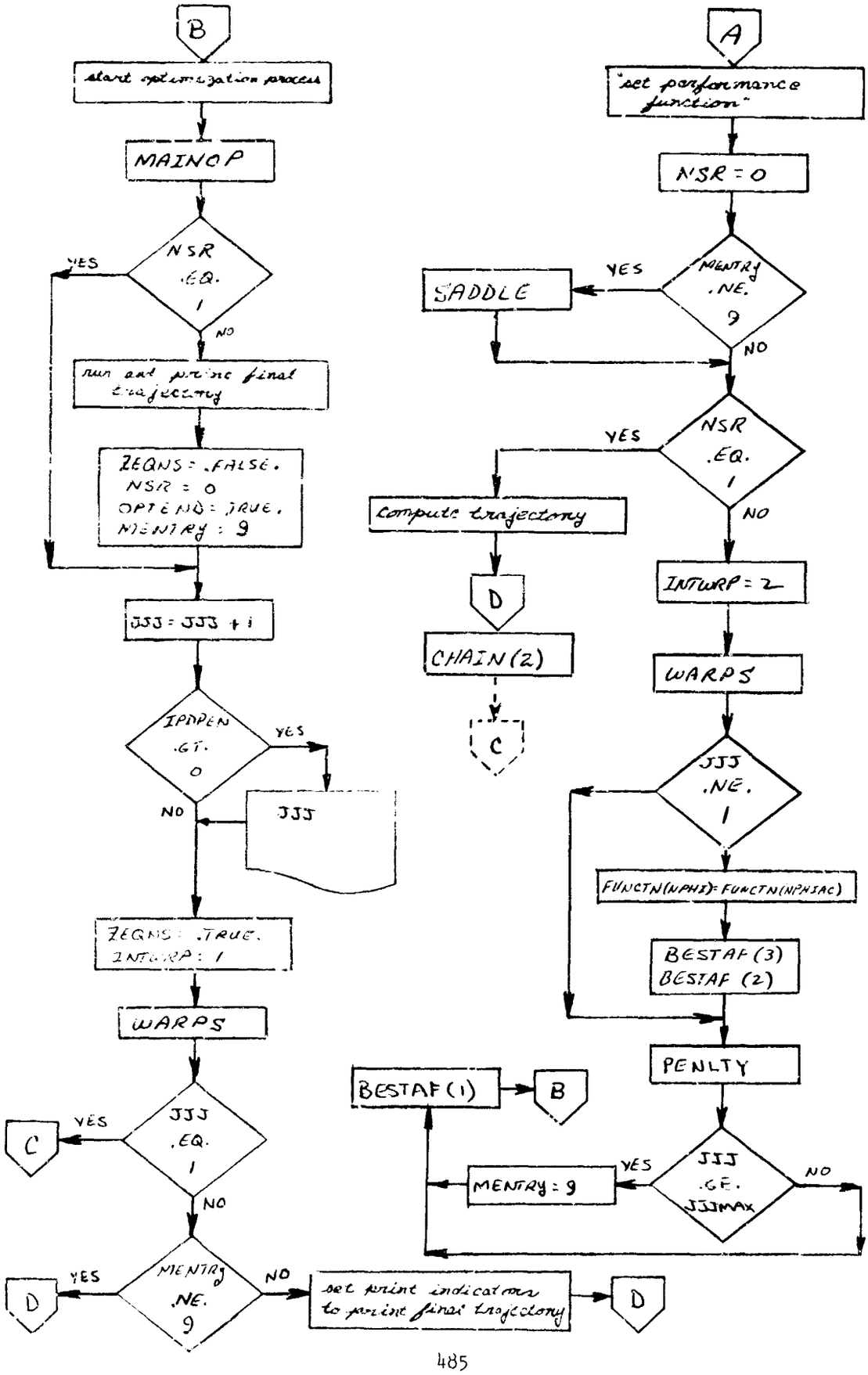
CTAE also controls a multiple extremal search procedure through subroutine WARPS, and a saddle-point search procedure for solution of "mini-max" problems through subroutine SADDLE.

Remarks:

All subroutines employed in the parameter optimization procedure are separately documented in Volume IV of the present report.



NOT REPRODUCIBLE


B
start optimization process
MAINOP
NSR .EQ. 1
YES
NO
run and print final trajectory
ZEQNS = .FALSE.
NSR = 0
OPTEND = .TRUE.
MENTRY = 9
JJJ = JJJ + 1
IPRPEN .GT. 0
YES
NO
JJJ
ZEQNS = .TRUE.
INTERP = 1
WARPS
JJJ .EQ. 1
YES
C
NO
MENTRY .NE. 9
YES
D
NO
set print indicators to print final trajectory
D
A
"set performance function"
NSR = 0
MENTRY .NE. 9
YES
SADDLE
NO
NSR .EQ. 1
YES
compute trajectory
D
CHAIN(2)
C
NO
INTERP = 2
WARPS
JJJ .NE. 1
FUNCTN(NPHI) = FUNCTN(NPHIAC)
BESTAF(3)
BESTAF(2)
PENLTY
JJJ .GE. JJJMAX
YES
MENTRY = 9
NO
BESTAF(1)
B

SECTION XI

ALPHABETIC INDEX OF SUBROUTINES

Subroutine Name	Page No.
ABSBEAM	480
ABSVECT	480
ACOS	167
ADJEQ	436
ANGLES	196
ANGLES2	196
ANITR	293
ANITR2	293
ASIN	169
ASRCH	295
ASRCH2	295
ATAN2	172
ATMS	290
ATMS2	290
ATMS59	333
ATMS592	333
ATMS62	335
ATMS622	335
ATTAC1	255
ATTAC2	255
ATTAC11	315
ATTAC21	315
ATTAC12	316
ATTAC22	316
BAESOP	56
BAITR	294
BAITR2	294
BDATA1	29
BDATA2	29
BDATA3	29
BDATA4	29
BDATA5	29
BDATA6	29
BDATA7	29
BIBLOCK	72
CARDS	367
CHAIN	30
CHAROPT	480
CHEMP	323
CHEMP2	323
CODES	131
CODES2	131
COMBAT	135
COMBAT2	135
CONV	326
CRATE	199
CRATE2	199
CTAE	483
CTLOPT	318
CTLITR	219
CTLITR2	219
CTLS	345
CTLS1	347
CTLS2	357
CTVS	90
CTVS2	90
CTVSR	434
DALCAL	369
DALPACK	453
DECIDE	384
DECID2	406
DECID3	400
DEF	59
DEF2	59
DEFEN1	246
DEFEN2	246
DEFEN11	301
DEFEN21	301
DEQACI	220
DEQACI2	220
DEQBCI	214
DEQBCI2	214
DEQCOD	225
DEQCOD2	225
DEQINI	213
DEQINI2	213
DEQHT	228
DEQHT2	228
DEQIV	227
DEQIV2	227
DEQPRE	206
DEQPRE2	206
DEQSIP	224
DEQSIP2	224
DEQVAL	226
DEQVAL2	226
DGAMES	188
DETECT	188
DETECT2	82
DIFEQ	154
DIFEQ1	154
DIFEQ2	164
DIFEQ3	

Subroutine Name	Page No.
DIFEQ4	164
DIFEQ5	164
DIFEQ6	164
DIPLAC	57
DISPLAY	362
DORDER	45
DSERCH	47
DSERCH2	47
ERROR	229
ERROR2	229
EVADE1	249
EVADE2	249
EVADE11	308
EVADE21	308
EXE	75
EXE2	75
EXERR	51
EXTRAN	125
EXTRAN2	125
FILTER	120
FILTER2	120
FIRFUN	207
FIRFUN2	207
FIXEDR	307
FIXEDR2	307
FLUSH	455
FLUSH1	184
FLUSH12	184
FPPG	222
FPPG2	222
FPPS	215
FPPS2	215
FRAME	481
GAMA91	211
GAMA92	211
GET	338
GRAPH	458
GRIDXY	297
GVSP	291
GVSP2	291
HETS	269
HETS2	269
HIHO	262
HIHO2	262
HLIMIT	194
HLIMIT2	194

Subroutine Name	Page No.
IFCS	93
IFCS2	93
IMAINOP	341
ISELECT	343
IPICK	300
ITEMS	133
ITEMS2	133
IZERO	235
IZERO2	235
INTGRT	128
INTGRT2	128
INTGRTR	128
INVERT	379
IZUNPK	456
KCALC	374
LATS	288
LATS2	288
LINCOM	104
LINCOM2	104
LINEOPT	480
LINES	62
LINES2	62
MAIN	27
MAIN2	27
MAIN1	33
MAIN12	33
MANTGT	84
MANTGT2	84
MAP	480
MATINV	398
MIMINF	143
MIMINF2	143
MIMINR	444
MISCUT	151
MISCUT2	151
MSGONE	32
MSGTWO	32
MULT31	268
MULT33	344
NDILU	320
NUMBER	481
OALPBA	258
OALPBA2	258
OBSFUN	115
OBSFUN2	115

Subroutine Name	Page No.
OFFEN1	252
OFFEN2	252
OFFEN11	312
OFFEN21	312
OFFEN12	313
OFFEN22	313
OFFEN13	314
OFFEN23	314
OFFSW	385
ONETWO	218
ONLINED	153
OPTBA	339
OPTBA2	339
PACBCD	64
PACK	407
PACKL	46
PACKR	66
PAPLT	472
PARTS	96
PAPERP	187
PASSV1	244
PASSV2	244
PENAL	109
PENAL2	109
PLCPTS	299
PLOT	392
PLTCUR	476
PLTS	112
PLTS2	112
PPLNLN	237
PRPACK	182
PRPLT	474
PSUBR	179
PTBEQN	231
PTBEQN2	231
PUT	337
READA	37
READA2	37
READB	54
READB2	54
READ31	68
REV	428
ROLE1	190
ROLE2	190
SACS	279
SACS2	279
SEARCH	363
SENSOR	239
SENSOR2	239
SETBEAM	48
SETGRD	186
SHELL	74
SLACK	107
SLACK2	107
SPRANG	29
STGTST	122
STGTST2	122
STOP1	49
SUMOLA	390
SVI	70
SVI2	70
SYMBOL	481
TABRE	60
TABRE2	60
TSRCH	52
TSRCH2	52
TFFM	329
TFFM2	329
TFFS	276
TFFS2	276
TIMID	147
TIMID2	147
TIM0001	204
TIM0012	204
TIMREV	177
TIMREV2	177
TLU	170
TLU2	170
TLU1	174
TLUU	386
TLUREV	165
TMTX	264
TWOONE	234
TRNPOS	267
UNBLOCK	388
UNPART	442
UPDK	411
VALUES	149
VALUES2	149
VECTOR	481
VISION2	242
WMA	446
WNORM	409

*U.S.Government Printing Office: 1971 — 759-078/131