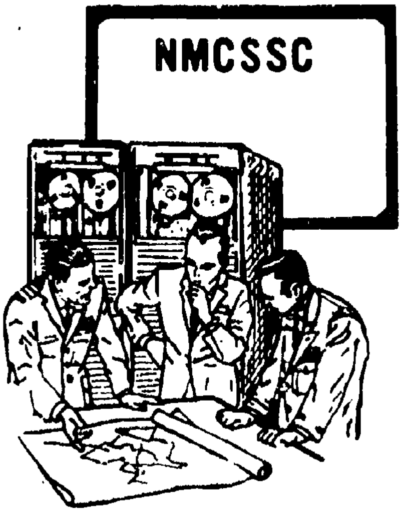
NATIONAL MILITARY COMMAND SYSTEM SUPPORT CENTER

AD 742786

COMPUTER SYSTEM MANUAL
CSM PSM 9A-67
VOLUME II, PART B
29 FEBRUARY 1972



THE NMCSSC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK)

PLAN GENERATION SUBSYSTEM

Vol. II - Pt. A
AD742785

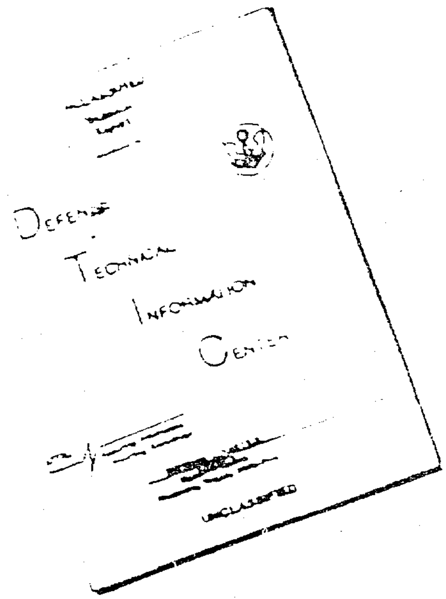
PROGRAMMING SPECIFICATIONS MANUAL

DEFENSE COMMUNICATIONS AGENCY

THIS DOCUMENT HAS BEEN APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

667

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

THIS DOCUMENT CONTAINED BLANK PAGES THAT HAVE BEEN DELETED

REPRODUCED FROM BEST AVAILABLE COPY

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) National Military Command System Support Center (NMCSSC) Defense Communications Agency (DCA) The Pentagon Washington, DC 20301	2a. REPORT SECURITY CLASSIFICATION
	2b. GROUP

3. REPORT TITLE
The NMCSSC Quick-Reacting General War Gaming System (QUICK) Programming Specifications Manual, Volume II, Plan Generation Subsystem

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)
N/A

5. AUTHOR(S) (First name, middle initial, last name)
NMCSSC: Robert R. Hardiman, Yvonne Mapily, Donald F. Webb
Lambda Corp: Paul D. Flanagan, Patricia M. Parish, Jack A. Sasseen

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
29 February 1972	1133	4

8a. CONTRACT OR GRANT NO. DCA 100-70-C-0065	9a. ORIGINATOR'S REPORT NUMBER(S) NMCSSC COMPUTER SYSTEM MANUAL CSM PSM 9A-67
b. PROJECT NO. NMCSSC Project 631	
c.	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)
d.	None

10. DISTRIBUTION STATEMENT
This document is approved for public release; its distribution is unlimited.

11. SUPPLEMENTARY NOTES	12. SPONSORING MILITARY ACTIVITY
	National Military Command System Support Center/Defense Communications Agency The Pentagon, Washington, DC 20301

13. ABSTRACT

This is one of three volumes describing computer programs of the QUICK-Reacting General War Gaming System (QUICK). These volumes complement other NMCSSC Computer System Manuals on QUICK by discussing the programs from a computer programming point of view. This volume, in six parts, concentrates on the Plan Generation Subsystem of QUICK. Other volumes are available for the Input Subsystem and Simulation Subsystem. Collectively, these volumes provide a good basis for maintenance activity on the QUICK System.

Based upon a suitable data base, and user control parameters, QUICK will generate individual bomber and missile plans suitable for war gaming. The generated plans are of a form suitable for independent review and revision. Subsequently, execution of the planned events can be simulated. Various statistical summaries can be produced to reflect the results of the war game. A variety of force postures and strategies can be accommodated.

QUICK is documented extensively in a set of Computer System Manuals (series 9-67) published by the National Military Command System Support Center (NMCSSC), Defense Communications Agency (DCA), The Pentagon, Washington, DC 20301.

DD FORM 1 NOV 65 1473 REPLACES DD FORM 1473, 1 JAN 64, WHICH IS OBSOLETE FOR ARMY USE.

UNCLASSIFIED
Security Classification

NATIONAL MILITARY COMMAND SYSTEM SUPPORT CENTER

Computer System Manual Number CSM PSM 9A-67

29 February 1972

THE NMCSSC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK)

Programming Specifications Manual

Volume II - Plan Generation Subsystem

Part B (Chapters 6 through 11)

Submitted by:

DONALD F. WEBB
Major, USAF
Project Officer

REVIEWED BY:

R. E. HARSHBARGER
Technical Director
NMCSSC

APPROVED BY:

BRUCE MERRITT
Colonel, USA
Commander, NMCSSC

Copies of this document may be obtained from the Defense Documentation Center, Cameron Station, Alexandria, Virginia 22314.

ACKNOWLEDGMENT

This document was prepared under the direction of the Chief for Development and Analysis, NMCSSC, in response to a requirement of the Studies, Analysis and Gaming Agency (SAGA), Organization of the Joint Chiefs of Staff. Technical support was provided by Lambda Corporation under Contract Number DCA 100-70-C-0065.

CONTENTS

PART A

PART B

Chapter | Page

Chapter		Page

Part C

Part D

Part E

Part F

ILLUSTRATIONS (PART B)

TABLES (PART B)

ABSTRACT

The computerized Quick-Reacting General War Gaming System (QUICK) will accept input data, automatically generate global strategic nuclear war plans, simulate the planned events, and provide statistical output summaries. QUICK has been programmed in FORTRAN for use on the NMCSSC CDC 3800 computer system.

The QUICK Programming Specifications Manual (PSM) consists of three volumes: Volume I, Data Input Subsystem; Volume II, Plan Generation Subsystem; Volume III, Simulation and Data Output Subsystems. The Programming Specifications Manual complements the other QUICK Computer System Manuals to facilitate maintenance of the war gaming system. This volume, Volume II, provides the programmer/analyst with a technical description of the purpose, functions, general procedures, and programming techniques applicable to the programs of the Plan Generation subsystem. This volume is in six parts: Parts A and B provide a description of the programs which make up the Subsystem: Parts C through F contain the associated program listings. Companion documents are:

1. GENERAL DESCRIPTION
   Computer System Manual CSM GD 9A-67
   A nontechnical description for senior management personnel

2. ANALYTICAL MANUAL
   Computer System Manual CSM AM 9A-67 (three volumes)
   Provides a description of the system methodology for the non-programmer analysts

3. USER'S MANUAL
   Computer System Manual CSM UM 9-67
   Provides detailed instructions for applications of the system

4. OPERATOR'S MANUAL
   Computer System Manual CSM OM 9A-67
   Provides instructions and procedures for the computer operators

CHAPTER 6
PROGRAM FOOTPRNT

PURPOSE

The purpose of program FOOTPRNT is to divide the set of targets assigned to a MIRV group into subsets, each of which is assigned to one booster in the group. This division is constrained by the limitations of the MIRV systems so that the acceptable booster assignments lie within a geographic pattern known as a footprint. The program is divided into two modules, the test module and the assignment module.

The test module receives as input a potential booster assignment. Using footprint constraint parameters supplied by the user, this module determines if the target set is a feasible footprint for the MIRV system.

The assignment module attempts to assign as many targets as possible to a booster within booster loading constraints specified by the user. Various loading options give the user flexibility in determining the type of loading to be employed.

This program receives the TMPALOC file from program ALOCOUT and prepares the ALOCGRP file. This latter file contains the weapon-target allocation ordered by weapon groups. Program FOOTPRNT processes only those groups with a MIRV capability. The target assignments to those groups are divided into subassignments, each of which is a feasible MIRV booster assignment.

There are two types of user input, algorithm control and footprint assignment parameters. The first type controls the performance of the heuristic algorithm which comprises the assignment module. (These parameters are discussed in Volume II of the User's Manual, Chapter 3, Plan Generation Subsystem, Program FOOTPRNT, Input.) The footprint assignment parameters define the nature of feasible footprints. These parameters define the fuel used in delivering a series of re-entry vehicles and decoys in a specific geographic pattern. The required user-input parameters are a set of coefficients to equations used to model the physical MIRV systems. (These equations are discussed in the Analytical Manual, Volume II, Plan Generation Subsystem, Basic Sortie Generation - MIRV Missile Plans.)

In addition to the parameters which define feasible footprints, the user specifies one of three options for loading the boosters. The first option, free loading, allows the algorithm to load the booster subject only to a maximum load (number of re-entry vehicles) constraint. The second option forces the algorithm to attempt to meet a minimum load constraint as well as the maximum load constraint. The third option requires the algorithm to meet both the minimum and maximum load constraints.

462 546 O - 72 - 2

INPUT FILES

There are three basic sources of input to program FOOTPRNT: the TMPALOC file, the BASFILE file, and the user-input parameter cards. The TMPALOC file is produced by program ALOCOUT. This file contains the assignment of weapons to targets, ordered by group and corridor. That is, all targets assigned to weapons from the same group are placed together on the TMPALOC file. Within each group, the targets assigned to the same corridor are also placed together. Common /STRKSUM/ on this file describes the ordering of corridors and strikes for each group. For missile groups, only corridor 0 is used, so this latter ordering by corridor has no effect. Program FOOTPRNT processes only those missile groups with a MIRV capability.* Therefore, all information on the TMPALOC file which does not deal with MIRV weapons is merely copied verbatim to the ALOCGRP file. For MIRV weapons, FOOTPRNT reads from the TMPALOC file the information contained in common blocks /STRKSUM/ and /3/ as output by ALOCOUT. These data are transferred to common blocks /STRKSUM/, /RAIDATA/, and /4/ in program FOOTPRNT. This information defines the possible targets which can be assigned to the weapons in each MIRV group. Program FOOTPRNT performs the assignment of targets to individual boosters and outputs the information on the ALOCGRP file in a form such that program POSTALOC can prepare the basic sorties for each MIRV missile booster.

The data retrieved from the BASFILE include: plan size information (/MASTER/), file information (/FILES/), weapon group information (from blocks /PAYLOAD/, /WPNTYPE/, /WPNGRP/ on BASFILE), and excess weapon assignment information (from /EXCESS/ on BASFILE).

The user-input parameters define the program control variables and the footprint constraint definition variables. The former set of variables governs the performance of the assignment module of the program. The latter set of variables defines the footprint constraints to be applied in the test module.

OUTPUT FILES

Program FOOTPRNT produces the ALOCGRP file. (This file is written using the QUICK filehandler. The description of those routines describes the format of the physical makeup of this file.) The file is output to the CDC 814 disk unit for use by program POSTALOC. If there are no MIRV weapons in the plan, program FOOTPRNT copies the TMPALOC file onto the ALOCGRP file. If there are MIRV weapons in the plan, the information for those groups without MIRV weapons is copied verbatim onto the ALOCGRP file.

---

* See program FOOTPRNT in Subroutines section of this chapter.

For those weapon groups with a MIRV capability, the data on the TMPALOC file are read into core. The program creates the individual booster assignments and outputs the strike information onto the ALOCGRP file. For MIRV groups, the data are organized as follows (see table 23). The index number of each target that receives the first re-entry vehicle (RV) from a booster is set negative. The strikes are ordered such that targets which receive successive RVs from the same booster are listed in that order. (See also following section: Concept of Operation.)

CONCEPT OF OPERATION

Program ALOCOUT prepares the TMPALOC file on which information concerning target assignments is sorted by group. For those groups with a payload containing multiple independently-targetable re-entry vehicles (MIRV), program FOOTPRNT performs further processing. The inclusion of a MIRV capability into the QUICK system is based upon the assumption that the MIRV weapons can be allocated to targets without regard to the "footprint" constraints. (These constraints define the geographic area into which the ordered set of re-entry vehicles from a single booster must be targeted.) This design approach considers that if a certain amount of extra or excess strikes are included in the allocation, the footprint constraints can be imposed later without the loss of payoff. Since imposition of the constraints may show that a certain number of strikes contained in the unconstrained allocation are not capable of inclusion in a feasible footprint, the extra strikes are added so that the final assignment contains the correct number of strikes.

This program prepares the ALOCGRP file for use by program POSTALOC. This file is very similar to the TMPALOC file. For those weapon groups with a MIRV capability, the data set on ALOCGRP differs from that on TMPALOC in the following ways:

1. The "extra" strikes have been removed
2. The index number (INDEXNO) of the target which receives the first re-entry vehicle (RV) from each booster is set negative
3. The strikes are ordered such that:
    a. Within each booster load (i.e., between minus signs) the strikes are ordered in order of their delivery by the missile
    b. The booster loads are ordered by decreasing value (as defined by the sum of the relative damage values (RVAL) for all targets assigned to the booster).

Table 23. Format for MIRV Group Records on ALOCGRP File

ASSOCIATED COMMON	VARIABLE OR ARRAY	LENGTH	DESCRIPTION
STRKSUM	KGROUP	1	Group number
↑	NTSTRK	1	Total number of strikes for this group
↓	NCORR	1	Number of corridors for this group (=1)
STRKSUM	NSTRK	30	Number of strikes assigned to each corridor
RAIDATA	NT	1	Total number of targets assigned to group
↑	JGROUP	1	Group number
	JCORR	1	Corridor number (=0)
	INDEX	NT	Index numbers of targets (negative if first target assigned to booster)
	TGTLAT	NT	Target latitude (degrees)
	TGTLONG	NT	Target longitude (degrees)
	RVAL	NT	Relative value of strike
	DLAT	NT	Offset latitude (degrees)
↓	DLONG	NT	Offset longitude (degrees)
RAIDATA	LLFIX	(NT/32)+1	Fixed assignment indicator
4	DESIG	NT	Target designator code
↑	TASK	NT	Target task code
↓	CNTRYLOC	NT	Target country location code
4	FLAG	NT	Target flag code

The only other data file required by program FOOTPRNT is the BASFILE. The program reads from this file:

1. Plan size information (/MASTER/)

2. Logical file units (/FILES/)

3. Weapon group information (/WPNGRPX/ in FOOTPRNT; /PAYLOAD/, /WPNTYPE/, /WPNGRP/ on BASFILE)

4. Excess assignment information (/EXCESS/ on BASFILE).

The program operates on a group-by-group basis. Each group is considered independently of all other groups. Hence, the discussion of all subroutines except the main program will consider only the operations required for the current group. For non-MIRV weapon groups, the TMPALOC data are copied onto ALOCGRP. For MIRV groups, further processing is required.

The program consists of two modules, the assignment module and the testing module. The assignment module determines the ordered subset of the total strike set that is to be assigned to each booster. The testing module determines the feasibility of any single booster assignment. The assignment module calls the testing module many times during construction of the subsets. Figure 81 displays a functional diagram of this program.

The card input data for each module are discussed in detail in the subroutine section (subroutine RDCARDF for assignment module, subroutine TABLINPT for testing module). The assignment module data include control information on which groups to process, the degree of effort expended in forming footprints, the booster loading option, and other parameters which govern the operation of the heuristic algorithm which subsets the target list. The testing module data describe the footprint constraints. These data contain information on fuel loads, maximum ranges, fuel consumption rates, and distance ratios.

In essence, the operation of this program is a reordering of a list. The input is an unordered list of strikes assigned to the group. The required processing is to subset and reorder this list such that each sublist is a feasible booster assignment. Since much of the processing involves lists of various kinds, it is useful here to describe some of the basic lists that are involved in processing (input, RAIDATA, POTENT).

The first list is the input data, contained in common /RAIDATA/ and common /4/. The data consist of several lists, each containing one element for each target assigned to the group. The lists contain index number (INDEXNO), target latitude (TGTLAT), target longitude (TGTLONG), relative damage value (RVAL), offset latitude (DLAT), offset longitude (DLONG), fixed assignment indicator (LLFIX), target designator code (DESIG), target

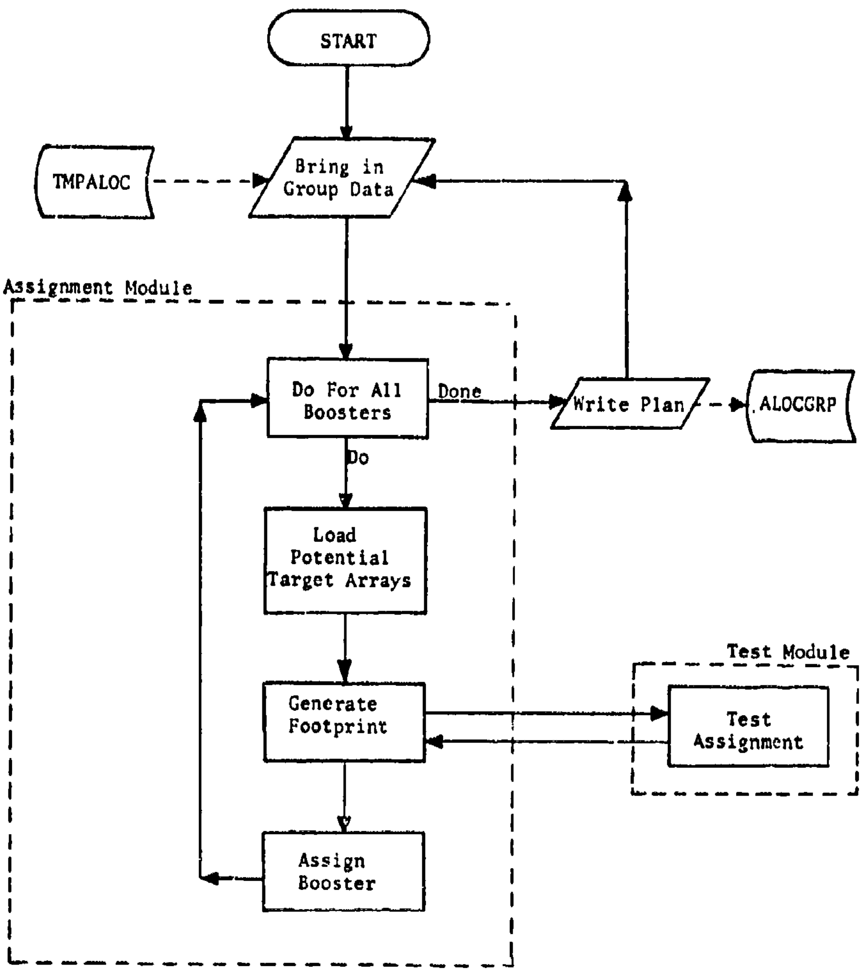


Fig. 81. Block Diagram of Program FOOTPRNT

task/subtask code (TASK), target country location code (CNTRYLOC), and target flag code (FLAG). Most of these data are not needed after the early calculations, so the data are written out onto a scratch file.

The geographic data are then converted to polar coordinates centered on the group centroid with axis passing through the North Pole. The range and launch azimuth from centroid to each target is computed and stored. The data are then reordered according to increasing value of launch azimuth. (The sequence array required for this ordering is also written on the scratch tape.) The reordered targets are then arbitrarily assigned to boosters in order of increasing azimuth. This initial assignment is made without consideration of footprint feasibility. It merely provides a starting point.

This processing has created the RAIDATA lists, contained in commons /RAIDATA/ and /2/. (In the remainder of this chapter, the RAIDATA index refers to the index of the target in the RAIDATA lists.) These lists comprise, for each target, the range (R) from the weapon group centroid, the launch azimuth (THETA), the relative damage value (RVAL), and the various pointers. The pointers for each target are defined as follows:

IFOR: A forward pointer. This is the RAIDATA index of the target which will follow in the footprint. If this target is not assigned to any footprint or if it is the last target in the footprint, IFOR = -1.

IBACK: A backward pointer. This is the RAIDATA index of the preceding target in the footprint. If this target is not assigned to any footprint, IBACK = -1. If this target is the first target in the footprint, then IBACK is set to the negative of the RAIDATA index of the target in the footprint with the greatest azimuth.

ISTATUS: A status indicator, defined as follows:

= -2 - Not assigned to any footprint and not in potential target list (see below)

= -1 - In the "lost" target list

= 0 - In potential target list

> 0 - Number of booster to which this target is assigned.

There are also two arrays in the RAIDATA lists which are indexed by booster. They are:

IBOOST: The RAIDATA index of the first target assigned to this booster

NTB: The number of targets currently assigned to this booster.

Although the RAIDATA list provides the basic data base for footprint construction, it would be inefficient to perform detailed calculations on all the targets in the list for every booster in the group. A subset of this list, named the potential target list, is created for each booster. The assignment to a booster can be formed with only the targets present in the potential target list for that booster. Detailed intertarget calculations are performed only on targets in this list. This list is contained in common blocks /POTENT/, /1/, and /3/. The index to targets in this list is called the POTENT index.

Within the POTENT list there is a further division. The hit list (IHIT) contains those targets which define the current footprint. The miss list (MISS) contains all those targets in the potential list which are not on the current hit list. The booster assignment is comprised of the last hit list constructed by the footprint construction subroutines.

The major arrays which comprise the POTENT list are:

IPOT: The RAIDATA index of the target.

INVERSE: A pointer to the hit and miss lists. If positive, it is the target's position on the hit list. If negative, it is the target's position on the miss list.

AGE: A factor related to the number of boosters processed while the target has remained in the POTENT list.

VALFIRST: The worth of using this target for the first re-entry vehicle delivered in a footprint.

JAFTER: The POTENT index of the target which would immediately precede this target if it were added to the footprint.

IHIT: The POTENT index of targets in the hit list.

TOFLY: The equivalent downrange distance between consecutive targets in the hit list.

COSTEFF: The worth of not deleting this target during the improvement phase.

MISS: The POTENT index of targets in the miss list.

VAL: The worth of adding the target to the current footprint.

NDEXVAL: A sequence array containing the order in which targets in the miss list will be tested for inclusion in the footprint.

IFREE: An array containing the indices of cells in the POTENT list which have no targets assigned to them. NFREE is the number of available cells. IFREE(NFREE) always contains the index of the next available cell.

LOST: The RAIDATA index of targets geographically close to the targets assigned to the current booster. These "lost" targets are awaiting entry into the miss list.

Common /3/ contains several arrays which store detailed data on the potential targets.

The details of the processing of elements in all three of these lists is contained in subsequent sections of this chapter. Figure 82 shows the hierarchy and function of the major subroutines of this program.

COMMON BLOCK DEFINITION

This program references external common blocks /MASTER/ and /TAPES/ from the BASFILE. In addition, certain information for common block /WPNGRPX/ is read from the BASFILE blocks /PAYLOAD/, /WPNGRP/, and /WPNTYPE/.

Tables 24 and 25 define the variables in each common block. Table 24 describes the external common blocks (those transferred on files to or from other QUICK programs), and table 25 describes the internal common blocks (those used internally to program FOOTPRNT). Table 26 lists these variables and their initial value. Those variables marked with an asterisk are given new values during execution.

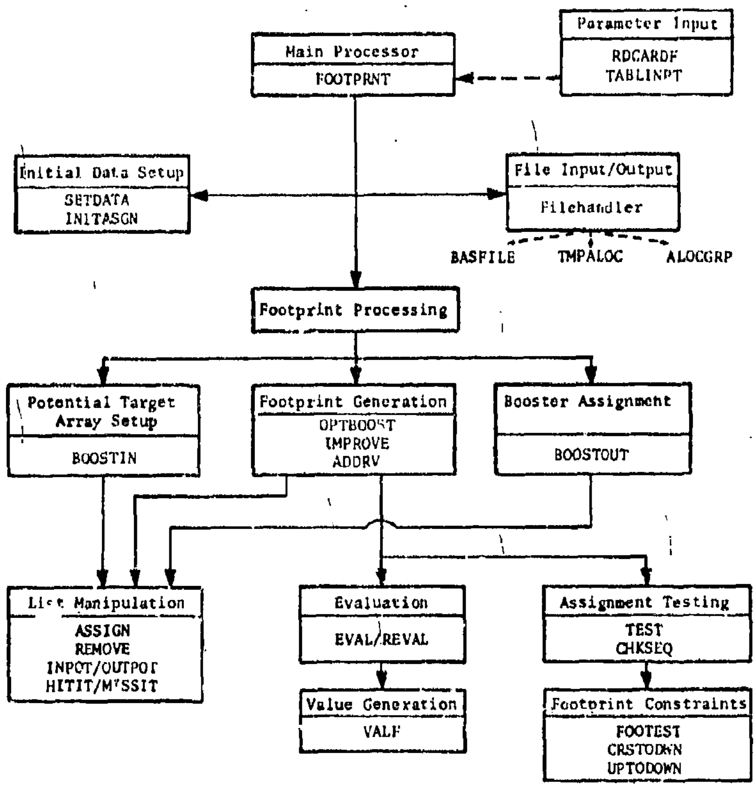


Fig. 82. Hierarchy of Major Subroutines of Program FOOTPRNT

Table 24. Program FOOTPRNT External Common Blocks
(Sheet 1 of 3)

INPUT FROM BASFILE

BLOCK	VARIABLE OR ARRAY*	DESCRIPTION
MASTER	IHDATE	Date of run initiation
	IDENTNO	Run identification number
	ISIDE	Attacking side
	NRTPT	Number of route points
	NCORR	Number of penetration corridors
	NDPEN	Number of depenetration corridors
	NRECOVER	Number of recovery bases
	NREF	Number of refuel areas
	NBNDRY	Number of boundary points
	NREG	Number of command and control regions
	NTYPE	Number of weapon types
	NGROUP	Number of weapon groups
	NTOTBASE	Total number of bases
	NPAYLOAD	Number of payload types
	NASMTYPE	Number of ASM types
	NWHDTYPE	Number of warhead types
	NTANKBAS	Number of tanker bases
	NCOMPLEX	Number of complex targets
	NCLASS	Number of weapon classes (two)
	NALERT	Number of alert conditions (two)
	NTGTS	Number of targets
	NCORTYPE	Number of penetration corridor types

*Parenthetical values indicate array dimensions. All other elements are single word variables.

Table 24. (cont.)
(Sheet 2 of 3)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
MASTER (cont.)	NCNTRY	Number of country codes on defending side
FILES	TGTFILE(2)*	Target data file
	BASFILE(2)	Data base information file
	MSLTIME(2)	Fixed missile timing file
	ALOCTAR(2)	Weapon allocation by targets file
	TMPALOC(2)	Temporary allocation file
	ALOCGRP(2)	Allocation by group file
	STRKFIL(2)	Strike file
	EVENTAPE**	Simulator events tape
	PLANTAPE**	Detailed plans tape
WPNGRPX***	NWPNS(200)	Number of weapons in group
	NVEHGRP(200)	Number of vehicles (boosters) in group
	WLAT(200)	Latitude of group centroid
	WLONG(200)	Longitude of group centroid
	ITYPE(200)	Weapon type
	IPAY(200)	Payload index
	ICLASS(80)	Class number
	ISIMTYPE(80)	Hollerith name of weapon system
	IMIRV(40)	MIRV system identification number

* First word is logical unit number; second word is maximum file length in words. These files are all on disk.

** Logical tape unit number. These files are on magnetic tape.

*** From blocks /WPNGRP/, /WPNTYPE/, and /PAYLOAD/ on BASFILE.

Table 24. (cont.)
(Sheet 3 of 3)

INPUT FROM TMPALOC AND OUTPUT ON ALOCGRP

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
STRKSUM	KGROUP	Group number
	NTSTRK	Total number of strikes assigned
	NCORR	Number of penetration corridors used
	NSTRK(30)	Number of strikes assigned to each penetration corridor
	LSTRKSUM	Length of STRKSUM record
RAIDATA*	NT	Total number of strikes
	JGROUP	Group number
	JCORR	Penetration corridor
	INDEX(1500)	Target index number
	TGTLAT(1500)	Target latitude
	TGTLONG(1500)	Target longitude
	RVAL(1500)	Relative value for target
	DLAT(1500)	DGZ offset latitude (degrees)
	DLONG(1500)	DGZ offset longitude (degrees)
	LRAID	Length of /RAIDATA/ block to this point
	NTMAX	Maximum number of target assignments for one group
	LLFIX(1500)	Fixed assignment indicator (logical type variable)
4	DESIG(1500)	Target designator code
	TASK(1500)	Target task/subtask code
	CNTRYLOC(1500)	Target country location code
	FLAG(1500)	Target flag code

* This block is redefined for internal use - see internal common block /RAIDATA/ in table 25.

Table 25. Program FOOTPRNT Internal Common Blocks
(Sheet 1 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
RAIDATA	NT	Total number of strikes
	JGROUP	Group number
	JCORR	Penetration corridor
	INDEX*(1500)	Target index number
	R(1500)	Distance from group centroid to DGZ (nautical miles)
	THETA(1500)	Launch azimuth of weapon from centroid to DGZ (radians)
	RVAL(1500)	Relative value for target
	IFOR(1500)	Forward pointer for booster assignments
	IBACK(1500)	Backward pointer for booster assignments
	LRAID	Length of /RAIDATA/ block to this point
	NTMAX	Maximum number of target assignments for one group
	LLFIX(1500)	Fixed assignment indicator (logical type variable)
CONTROL	NV	Number of boosters in group
	NARV	Average number of targets per booster in initial assignment
	NEXTRA	Number of boosters with initial assignments containing (NARV + 1) re-entry vehicles
	PEXTRA**	Fraction of total strikes that are excess strikes added by PREPALOC
	NPASS	Processing pass number

* Array IDUM, used for input/out temporary storage equivalenced to this array.

** From common block /EXCESS/ on BASFILE.

Table 25. (cont.)
(Sheet 2 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
CONTROL (cont.)	FRACLOOK	Fraction of next booster load for look-ahead
	MAXFOOT	Input parameter governing degree of effort expended in subroutine OPTBOOST
	DELAGE	Multiplier for AGE in potential target arrays
	PURGE	Fraction of targets in potential target arrays removed in BOOSTIN
	PN	Weighting factor for worth function
	EXTRAB*	Number of extra booster loads added in PREPALOC
	NOK	Actual number of correct strikes to be assigned
	IGSTART	First group to process
	IGEND	Last group to process
DSQUARE	CD2	Square of CROSSDWN } see common /RANGE/
	UD2	Square of UPDOWN } see common /RANGE/
	DEL2	Square of DELMIN } see common /RANGE/
	DZ2	Square of DZ } see common /VALPARM/
	VMIN	=VALF(DELMIN/DZ,TNZ) } see common /VALPARM/
EARTH	RADIUS	Radius of earth (Nautical miles)
	DEGTORAD	Conversion factor for degrees to radius
	PI	Pi
	PIDIV2	Pi/2
FOOTIO	MAXRV	Maximum number of re-entry vehicles allowed in one assignment
	ISYS	System identification number
	NTAR	Number of re-entry vehicles currently assigned

* From common block /EXCESS/ on BASFILE.

Table 25. (cont.)
(Sheet 3 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
FOOTIO (cont.)	RIN(20)	Range to target (nautical miles)
	THIN(20)	Azimuth to target (radius)
	IFEAS	Number of targets that can be reached within fuel constraints
	DELRSTRT	Maximum additional flying distance allowed if first target in footprint is to be changed (nautical miles)
	DELRAFT(20)	Maximum additional flying distance allowed if new target is to be added after this target in footprint (nautical miles)
	FUELEFT	Fuel left after completion of weapon deliveries
FOOTSAVE	IFOTSAVE(20)	Potential target index of targets in first footprint
	NHITOLD	Number of targets in first footprint
	VHITOLD	Sum of RVALs for targets in first footprint
	IF2SAVE(20)	Potential target index of targets in second footprint
	N2SAVE	Number of targets in second footprint
INDEX	JINR	RAIDATA index of target to be entered into potential target arrays
	JINP	Potential target index of target to be entered
	JOUTR	RAIDATA index of target to be removed from potential target arrays
	JOUTP	Potential target index of target to be removed
	JSAVE(20)	Potential target index of targets entered by look-ahead

Table 25. (cont.)
(Sheet 4 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
INDEX (cont.)	NJSAVE	Number of targets entered by look-ahead
	JSAVOPT	Look-ahead flag
LOADATA	LOADOPT	Booster loading option
	NRVADD(1500)	Number of extra re-entry vehicles added to this target
	NADDED	Total number of extra RVs added in a pass
	NTOADD	Number of RVs to be added to current footprint
	NONTAR(20)	Total number of RVs on each target in assignment
	NADDOLD	Number of extra RVs added in first pass
PARAMETR	MAXSYS	Maximum number of systems allowed in footprint parameter table
	IHNAME(40)	Hollerith name of MIRV system
	MINLOAD(40)	Minimum number of RVs per booster
	MAXLOAD(40)	Maximum number of RVs per booster
	DSPACE(40)	Minimum spacing (nautical miles) between consecutive DGZs in footprint
	THROWMAX(40)	Maximum distance between consecutive DGZs in footprint (nautical miles)
	MTYPE(40)	Footprint constraint functional form designator
	IDATA(40)	Index to footprint parameter data set
PERFORM	NASGN	Total number of targets assigned to boosters in current pass
	VALASGN	Sum of RVALs for all targets assigned in current pass
	TVAL	Sum of RVALs for all targets
	NOLD	Number of targets assigned in first pass

482-546 O - 72 - 3

Table 25. (cont.)
(Sheet 5 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
PERFORM (cont.)	VALOLD	Sum of RVALs for targets assigned in first pass
POTENT	MAXPOT	Maximum number of potential targets
	MAXHIT	Maximum number of targets in hit list
	IPOT(50)	RAIDATA index of potential targets
	NHIT	Number of targets in hit list
	IHIT(20)	Hit list - potential target index
	TOFLY(20)	Distance (nautical miles) between successive targets in hit list
	NMISS	Number of targets in miss list
	MISS(50)	Miss list - potential target index
	NFREE	Number of available spaces in potential arrays
	IFREE(50)	Potential target index of available spaces
	NLOST	Number of "lost" targets
	LOST(50)	RAIDATA index of "lost" targets
	INVERSE(50)	Index to position in hit or miss list
	AGE(50)	Factor related to number of boosters processed while target remains in potential target arrays
RANGE	CROSSDWN	Ratio of downrange to crossrange distance
	UPDOWN	Ratio of downrange to uprange distance
	DELMIN	Minimum spacing between consecutive DGZs in a footprint
	DEFAULT	Minimum spacing allowed for computation
TSCRATCH	ISCR	Logical unit number for assignment data scratch file

Table 25. (cont.)
(Sheet 6 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
TSCRATCH (cont.)	ITABL	Logical unit number for footprint parameter data scratch file
VALPARM	DZ	Maximum distance between consecutive DGZs in footprint
	TNZ	Intercept for value line (determined by PN in /CONTROL/)
	SLZ	Slope of value line
WPNTGT	IPOTGT	Potential target index of target to be added or deleted from hit list
	JAFT	Potential target index of target after which new target is to be added in hit list
	JTGTD	RAIDATA index of target to be removed from a booster assignment
	NUMBOOST	Booster number currently being processed
1	VAL(50)	Worth of target if added to footprint
	JAFTER(50)	Potential target index of target preceding new target in footprint
	VALFIRST(50)	Worth of making target first target in footprint
	COSTEFF(20)	Inverse of additional fuel needed to reach this target
	D(50,50)	Distance computation matrix
2	MAXBOOST	Maximum number of boosters allowed in one group
	IBOOST(500)	RAIDATA index of first target assigned to booster
	NTB(500)	Number of targets assigned to booster
	ISTATUS(1500)	Target processing status
	NDEX(1500)	Temporary index storage

Table 25. (cont.)
(Sheet 7 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
3	RP(50)	Range of target
	TP(50)	Azimuth to target
	RVALP(50)	Target relative value
	SINES(50)	Sine of azimuth
	COSINES(50)	Cosine of azimuth
	AVRP	Average range of all potential targets
	AVTHP	Average azimuth of all potential targets
	RHIT	Range to first target in footprint
	THIT	Azimuth to first target in footprint
	SINAV	Sine of AVTHP
	COSAV	Cosine of AVTHP
	SINHIT	Sine of THIT
	COSINHIT	Cosine of THIT
	THOLD	Azimuth used to compute entries in distance matrix
DEBUG	IOTA	Index to last entry in ICAMFROM array
	ICAMFROM(20)	Hollerith names of subroutine calling sequence
PRINT	ICALL	Print request number
	IMUST	Error condition indicator
Filehandler*		See section in this manual on filehandler
FLAG	NFLAG	Maximum number of print options
	IFLAG(100).	Active print indicator
	NC	Number of print requests

* ITP, TWORD, MYIDENT, NOPRINT, IFTPRNT, FILABEL, MYLABEL

Table 25. (cont.)
Sheet 8 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
FLAG (cont.)	IPRNT(60)	Print option number
	IFG(60)	First group to be printed
	IFP(60)	First pass to be printed
	IFB(60)	First booster to be printed
	ILG(60)	Last group to be printed
	ILP(60)	Last pass to be printed
	ILB(60)	Last booster to be printed
	MYPRT(60)	Mode by which print was requested (DEFAULT, INPUT, or REMOVED)
	IDUMP	Print number to abort run with memory dump

The following blocks contain the parameters which define the footprint constraints. The descriptions of subroutines TABLINPT and SETDATA contain more detailed information.

FOOTDATA (long-range system)	GAS(2)	Fuel available for footprinting
	RX(2,2)	Basic range extension coefficient
	RAXX(2,2)	Added range extension coefficient
	TOSSC1(2,2,2), TOSSC2(2,2,2)	Fuel consumption parameters
	TEONE(2,2), TETWO(2,2)	Fuel consumption exponents
	TDENOM(2)	Distance scaling factor
	RBASIC(2,2)	Basic maximum booster range
	RADD(2,2)	Added maximum booster range
	EONE(2), ETWO(2)	Downrange-crossrange ratio exponents
	DENOM	Distance scaling factor

Table 25. (co)
(Shee)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
FOOTDATA (cont.)	CONE(2,2) CTWO(2,2)	Downrange-crossrange ratio coefficients
	LLNGDAT	Length of this block
SHRTDAT (short range system)	ALPHAZ(16) ALPHA1(16) ALPHA2(16)	Fuel consumption parameters
	BETAZ(16) BETA1(16) BETA2(16)	Fuel load parameters
	MAXRBOST(16)	Maximum booster range
	GTWO GONE GZERO	Downrange-crossrange ratio parameters
	DONE DZERO	Downrange-uprange ratio parameters
	LSHTDAT	Length of this block
PENADD (additions for penetration aids)	TOTFUEL	Total fuel available for spacing, release, and footprinting
	SRFC1(2) SRFC2(2)	Spacing and release fuel coefficients
	SRFEXP1(2) SRFEXP2(2)	Spacing and release fuel exponents
	SRFDEN	Distance scaling factor
	LPENDAT	Length of this block

Tablo 26. List of Initial Settings of Variables**
(Sheet 1 of 2)

An asterisk flags variables whose values are changed during processing.

VARIABLE	BLOCK	INITIAL VALUE	REMARKS
AGE*	POTENT	0.0	Length of time target has remained in potential target list
AZDIFF		.01	Used by subroutine REVAL to determine necessity of recomputing distance matrix
AZOLD*		10^{-9}	Used by subroutine FOOTEST to determine necessity of recomputing fuel consumption parameters
DEFAULT	RANGE	1.0	Minimum spacing of DGZs required for computation
DEGTORAD	EARTH	.0174532	Conversion factor-degrees to radians
EPSILON*		10^{-9}	Same use as AZOLD
IERR*		0	Error counter in subroutine FOOTEST
ILASTL*		0	Used by subroutine SETDATA to determine if new footprint data are required
ILASTP*		0	
ILASTS*		0	
IMUST*	PRINT	0	Error condition indicator
ISCR	TSCRATCH	5	Scratch file logical unit (assignment data)
ITABL	TSCRATCH	6	Footprint data file logical unit
LLNGDAT	FOOTDATA	57	Number of words in long-range system data set
LPENDAT	PENADD	67	Number of words in penetration aids system data set
LRAID	RAIDATA	9003	Length of /RAIDATA/ block

**This list does not include the default values of the user-input parameters which are described in subroutine RDCARDF.

Table 26. (cont.)
(Sheet 2 of 2)

VARIABLE	BLOCK	INITIAL VALUE	REMARKS
LSHTDAT	SHRTDAT	117	Length of short-range system data set
LSTRKSUM	STRKSUM	33	Length of /STRKSUM/ block
MAXBOOST	2	500	Maximum number of boosters per group
MAXHIT	POTENT	20	Maximum number of RVs in one footprint
MAXPOT	POTENT	50	Maximum number of entries in potential target list
MAXRV	FOOTIO	20	Maximum number of RVs in footprint that can be tested
MAXSYS	PARAMETR	40	Maximum number of systems that can be considered in one run
NFLAG	FLAG	100	Maximum number of print options
NTMAX	RAIDATA	1500	Maximum number of targets per group
PDIFF		.001	Used by subroutine FOOTEST to determine necessity of fuel parameter recomputation
PI	EARTH	3.1415927	
PIDIV2	EARTH	1.5707963	
RADIUS	EARTH	3437.746	Nautical miles
THOLD*	3	10^{+300}	Azimuth used for distance matrix
XOLD		10^{-9}	Same as AZOLD

PROGRAM FOOTPRNT

PURPOSE: This is the main program. It acts as a control driver for the rest of the subroutines. It is the interface subroutine between this program and the remainder of the QUICK system.

ENTRY POINTS: FOOTPRNT

FORMAL PARAMETERS: None

COMMON BLOCKS: MASTER, FILES, TSCRTCH, WPNGRPX, STRKSUM, RAIDATA, 4, CONTROL, DSQUARE, EARTH, FOOTIO, FOOTSAVE, INDEX, LOADATA, PARAMETR, PERFORM, POTENT, RANGE, VALPARM, WPNTGT, 1, 2, 3, DEBUG, PRINT, Filehandler (ITP, MYIDENT, NOPRINT, IFTPRNT, TWORD, FILABEL, MYLABEL)

SUBROUTINES CALLED: STORAGE, Filehandler (INITAPE, SETREAD, RDARRAY, RDWORD, SETWRITE, WRARRAY, TERMTAPE), RDCARDF, PRINTSET, SKIP, INITRANS, GOPRINT, TRANSFER, VALF, SETDATA, NEWCOOR, INITASGN, BOOSTIN, OPTBOOST, BOOSTOUT, ORDER, REMOVE, REORDER, LREORDER

CALLED BY: Operating System; this is a main program

Method

The functioning of program FOOTPRNT can be divided into five parts; the flowchart and the following description are similarly divided. The parts are: the initialization of the program control variables, reading the strike data and determining the groups with the MIRV capability, setting the control data for each individual MIRV group, generating the footprints for each booster in the group, and finally selecting, formatting, and writing the final plan. The majority of the file reading and writing is accomplished in this program and the specific cases are discussed in later paragraphs.

Part I - The Initialization of Control Variables

The functioning of this part of the program is quite straightforward logically. The program begins by calling subroutine INITAPE to initialize the filehandler. Subroutine RDCARDF is then called to read and interpret the user-input parameters. These parameters include the print requests, program control variables, and footprint parameter data tables. The use of these parameters is described under subroutine RDCARDF. Subroutine PRINTSET is then called to initialize the print function flags. Then the majority of the basic weapon group information is read from the BASFILE. Commons /MASTER/ and /FILES/ are filled from the blocks of the same name on the BASFILE. Common /WPNGRPX/ is filled from BASFILE blocks /PAYLOAD/, /WPNGRP/, and /WPNTYPE/. Finally, the variables EXTRAB and PEXTRA (in common /CONTROL/) are read from block /EXCESS/ on the BASFILE. This part finishes by initializing the TMPALOC and ALOCGRP files and requesting the preliminary prints.

Part II - Reading of the Strike Data and Determination of Groups with a MIRV Capability

This section merely determines the data to be read from the TMPALOC file and places the data in core for processing by the remainder of the program. (For groups which do not have a MIRV capability, the data are copied from the TMPALOC file onto the ALOCGRP file for use by program POSTALOC.) This section begins by reading the data for common /STRKSUM/ from the TMPALOC file. This record contains the group number, the number of corridors for which strikes are planned, and the total number of strikes in each corridor. If the value of the group number is equal to an end-of-file marker (3HEOT) or if this value is greater than the user-input parameter IGEND read by subroutine RDCARDF, then the program goes to a termination block which finishes processing. (The termination block merely sets an end-of-file marker on the ALOCGRP file, terminates all the files, prints a termination message, and returns control to the system monitor.) If the end of processing has not been reached, the program determines the length of the fixed assignment indicator record. This record is a logical array which has been constructed by program ALOCOUT to show which weapons in the group were set by the fixed assignment capability of program ALOC. This indicator is used by later processors so that if various constraints require deletion of certain weapons, those weapons whose assignments were fixed by the user in program ALOC will not be among those deleted. Since the CDC 3800 computer system packs logical arrays with more than one element per computer word, the program must determine the length in words of this logical array which must be read for further processing. The program then determines if the current group has a MIRV capability. The first test is on the number of corridors in the strike data. If the number of corridors is greater than one, then the group must have bomber weapons since a weapon group with only missile weapons will send all its strikes to the same corridor, labeled 0. If the current group is a bomber group with more than one corridor for its strikes, the data are just copied out onto the ALOCGRP file and control returns to the beginning of this part to read the block of data for the next group. If only one corridor is assigned for the strikes of the group, the program reads the first three words of the /RAIDATA/ block. These words contain information on the number of the corridor to which these strikes are to be assigned. If the corridor number is not a 0, then again it is a bomber group and the data are merely copied out onto the ALOCGRP file. If the corridor number is a 0, it is a missile group and the program determines if the group number is greater than the user-input parameter IGSTART read by subroutine RDCARDF. If the group number is too low (that is, less than IGSTART), the data for the missile group are copied out onto the ALOCGRP file. The next test checks the payload table to see if the attribute IMIRV has a value greater than zero for this group. If so, this missile group does have a MIRV capability and is a candidate for further processing. The final test that is made before going on to the next part is to check for sufficient room in the arrays for the strikes assigned to this group. If there is not enough room, an error message is printed, the data are copied out onto the ALOCGRP file without further processing, and control is returned to the beginning of this part. If there is sufficient room to enable the later subroutines to process the data, control is transferred to Part III.

Part III - Setting Control Data for the Individual Group

Certain data must be preset before processing can begin on the MIRV group. Processing in this part is relatively straightforward. First, the Hollerith name of the weapon type is tested to see if there is agreement between the name input with the footprint parameter constraints and the name used by the QUICK system. If the names do not match, a warning message is printed and processing continues as usual. The program then sets up the minimum intertarget distance; that is, the minimum distance in nautical miles between consecutive desired ground zeros for targets assigned to the same booster. The program then sets up the value line for function VALF.

A determination is now made of the actual number of re-entry vehicles that are to be assigned to all the boosters in this group. Program PREPALOC added some extra re-entry vehicles so that the allocator would assign an excess of targets to the group. These extra re-entry vehicles were added to simplify the processing in program FOOTPRNT. Program ALOC does not consider footprint constraints when allocating weapons to targets. It is possible, therefore, that some of the targets assigned by the allocator cannot be put into a feasible footprint. Therefore, the excess of weapons allows for these targets to be ignored when generating the assignments for each booster without producing a total assignment which underutilizes the weapons. Thus this part of the program must determine the actual number of re-entry vehicles to be assigned. These data are stored in the variable named NOK.

Since the majority of data in the /RAIDATA/ block will be reordered and modified by later processing, it must be saved so that the correct data may be written out on the ALOCGRP. Therefore this part writes onto a scratch file, ISCR, the data that were read from the TMPALOC file into common /RAIDATA/. Table 27 displays the format of this scratch file. This part then calls three subroutines, SETDATA, NEWCOOR, and INITASGN. These subroutines, respectively, set up the footprint testing algorithms with the correct footprint parameters, convert the geographic information from latitude and longitude to range and launch azimuth from the group centroid, and perform the initial assignment of targets to the boosters. The function of these subroutines is discussed later. Finally, this part initializes all the arrays which make up potential target lists.

Part IV - Construction of Footprints

This part is divided into two passes. In the first pass the boosters are considered individually in order of increasing launch azimuth. In the second pass they are considered in order of decreasing azimuth. This method is used so that any potential target will be investigated by boosters whose initial assignment falls on either side of the launch azimuth of the potential target. Because the majority of the processing of footprints is done by subroutines BOOSTIN, OPTBOOST, and BOOSTOUT, the sole function of this part of program FOOTPRNT is to call these subroutines in their proper order. The only logically complicated section of this part involves the deletion of excess weapons from the assignment. As was mentioned earlier, it is possible that the program may be able to form feasible footprints for a number of re-entry vehicles which is greater than the actual number that the group has. This is caused by the excess weapons which are added in program PREPALOC. Therefore, if the number which has been assigned is greater than the number which the group really has (i.e., NOK), then certain targets must be omitted from the assignments. (This function is performed by the 3000 series statements in the program.) The targets are ordered by increasing marginal damage, RVAL, as determined by program ALOCOUT. Thus the first targets to be omitted from the assignment are those with the least marginal damage (i.e., lowest RVAL). The program considers the targets in value order until it reaches a target which is assigned to the current set of footprints, has not been allocated by the fixed assignment capability of program ALOC, and is assigned to a booster with at least the minimum load (if the free-booster loading option is not in effect). If a target meets all these conditions, then it can be removed from the assignment. This process continues until a sufficient number have been omitted so that the total number assigned to all the boosters in the group does not exceed the actual number of re-entry vehicles which are available to the group. Control then passes to the fifth and final part of this program.

Table 27. Format for Assignment Data Scratch File

BLOCK	LENGTH	VARIABLE	DESCRIPTION
1	NT	INDEX	Target index number
2	NT	TGTLAT	Target latitude
3	NT	TGTLONG	Target longitude
4	NT	RVAL	Relative damage value
5	NT	DLAT	Offset latitude
6	NT	DLONG	Offset longitude
7	NT	NDEX	Sequence array for reordered data
8	NT	IFOR	Forward pointers for first pass
9	NV	IBOOST	First target for booster in first pass
10	NV	NTB	Number of targets assigned to booster in first pass
11	NT	NRVADD	Number of RVs added to each target in first pass

NT = Number of targets input from TMPALOC

NV = Number of boosters in group

Part V - The Selection, Formatting, and Output of the Final Plan for the Group

This part begins by retrieval of the group data which were put on the assignment data scratch file while the footprints were being processed. Then the program determines the better plan, the plan constructed in the first pass or the plan constructed in the second pass. The method used to select the better plan is to construct a weighted sum of the actual number of targets assigned to the boosters and the number which were added to meet the minimum load constraints in those cases where the free booster loading option was not in effect. This weighted sum is equal to twice the number of the actual targets assigned to the boosters plus the number assigned to meet the minimum load constraints. The plan whose weighted sum is greater is selected as the plan to be output for later programs. If the weighted sums for the two passes are equal, then the plan with the greater value assigned is selected. The value of a plan is the sum of the marginal damage values, RVAL, of all the targets assigned to the plan. If the first plan is selected then the program retrieves the pointers to the lists which were output onto the scratch tape. This destroys the pointers for the second plan.

The later programs in the Plan Generation subsystem expect the plan for missile groups to be in order of decreasing value of the booster assignments. That is, if we define the booster value to be the sum of the RVAL values of all the targets assigned to the booster, then the later processors expect these booster values to be decreasing as the lists are output onto the tape. This part of program FOOTPRNT, therefore, computes the value of the strikes assigned to each booster. It then reorders the total target list so that the strikes are in order not only by booster value but by order of delivery by the MIRV equipment. This function is done by assigning to every strike a "sequence" index which is defined as follows:

SI = (1000*NBBV)+ND

where SI = sequencing index

ND = order of delivery from booster (1 = first, 2 = second, etc.)

NBBV = order of booster value (1 = most valuable booster, 2 = second most valuable, etc.)

Thus once every strike has been assigned a sequencing index, a simple operation to order the strikes by the sequencing index will put them into correct sequence. The program then negates the index numbers for those strikes which are the first strike to be delivered from each booster and prepares the plan for output onto the ALOCGRP file for program POSTALOC.

The most complicated section of this part involves the addition of the extra re-entry vehicles which were placed on the booster in order to meet the minimum load constraints. The method to make these additions is as follows. When a target has been reached which has an extra number of RVs allocated to it, then the data (such as the target latitude, longitude, and relative value) for this target are saved. The program then looks through the RAIDATA list to find a target which has not been assigned to any booster. This position in the list is then used for a re-entry vehicle which has been added. The data which had been in this list previously for the strikes are removed and the saved data for the target to which the re-entry vehicles have been added are placed in this position in the list. The sequencing index is also placed in an array so that the added re-entry vehicles will follow the first re-entry vehicle allocated to the target.

Finally, the data are written onto the ALOCGRP file and control returns to the beginning of Part II where the strike data for the next group are read.

Program FOOTPRNT is illustrated in figures 83 and 84.

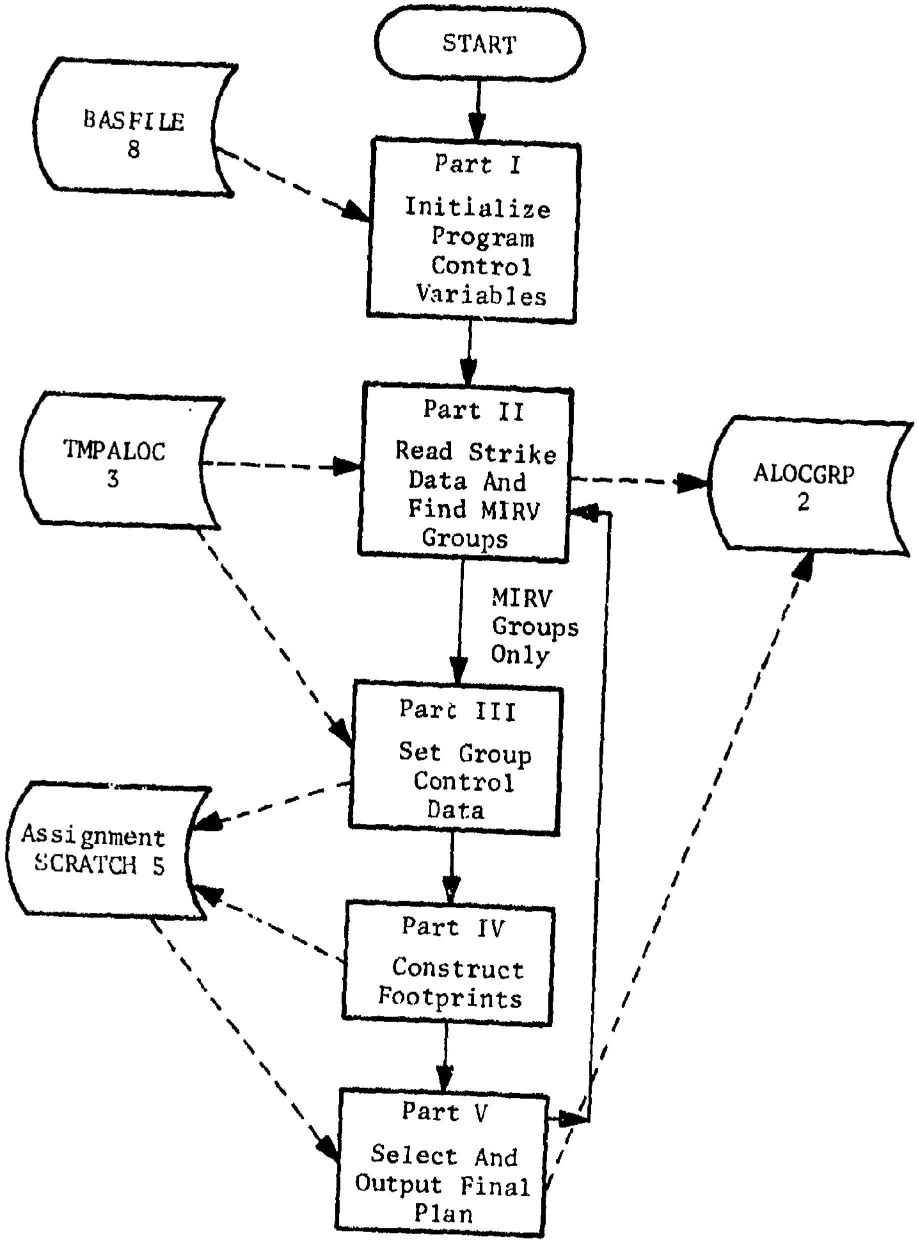


Fig. 83. Program FOOTPRNT (General Flow)

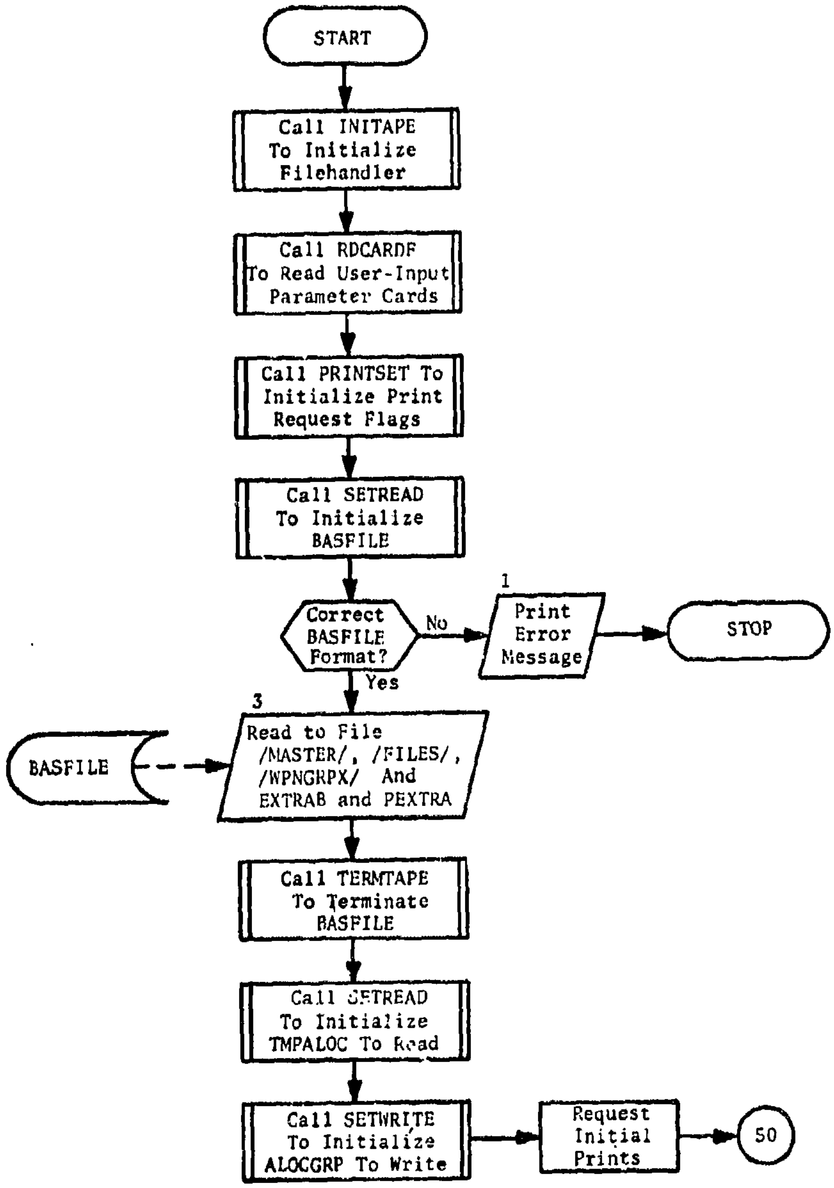


Fig. 84. Program FOOTPRNT (Detailed Flow)
Part I: Control Variable Initiation

462-546 O - 72 - 4

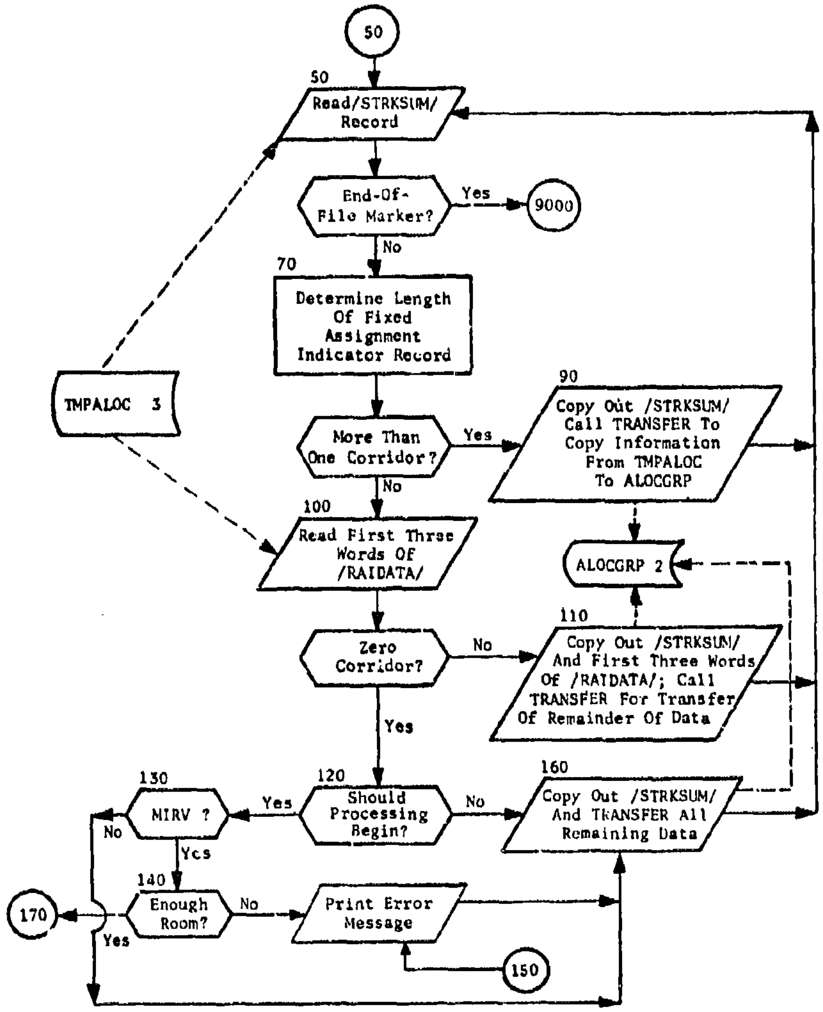


Fig. 84. (cont.)
Part II: Read Strike Data and Find MIRV Groups

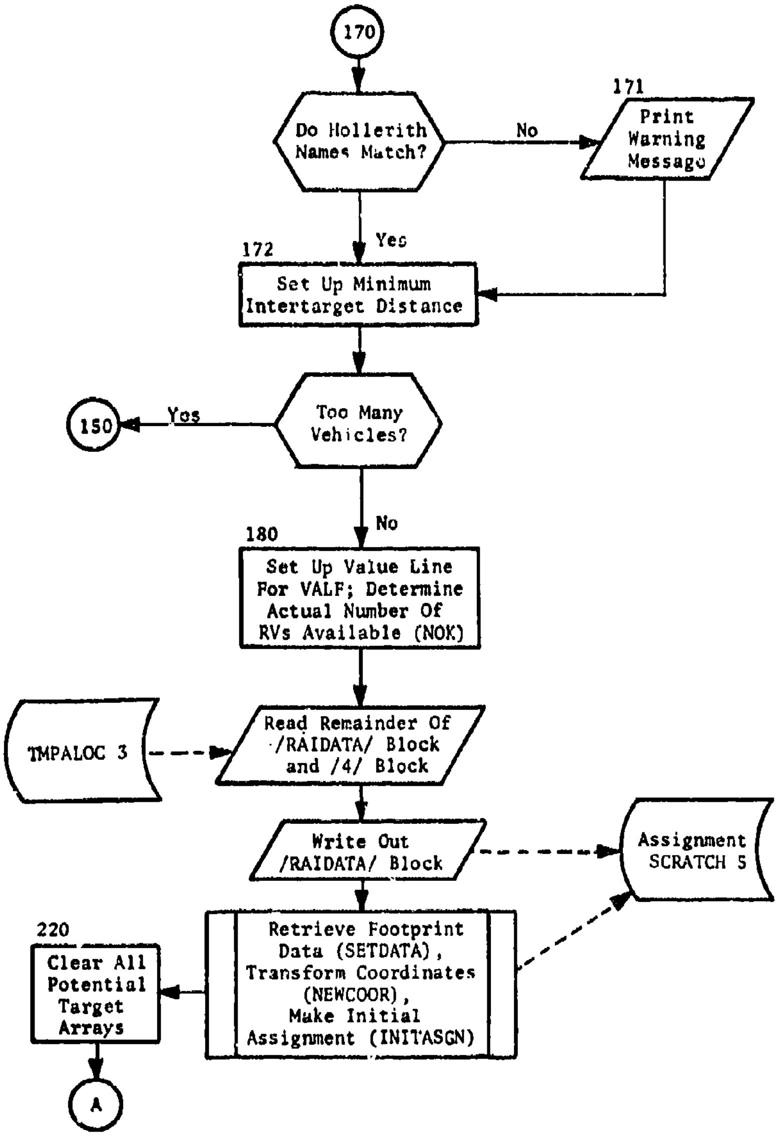


Fig. 84. (cont.)
Part III: Set Group Control Data

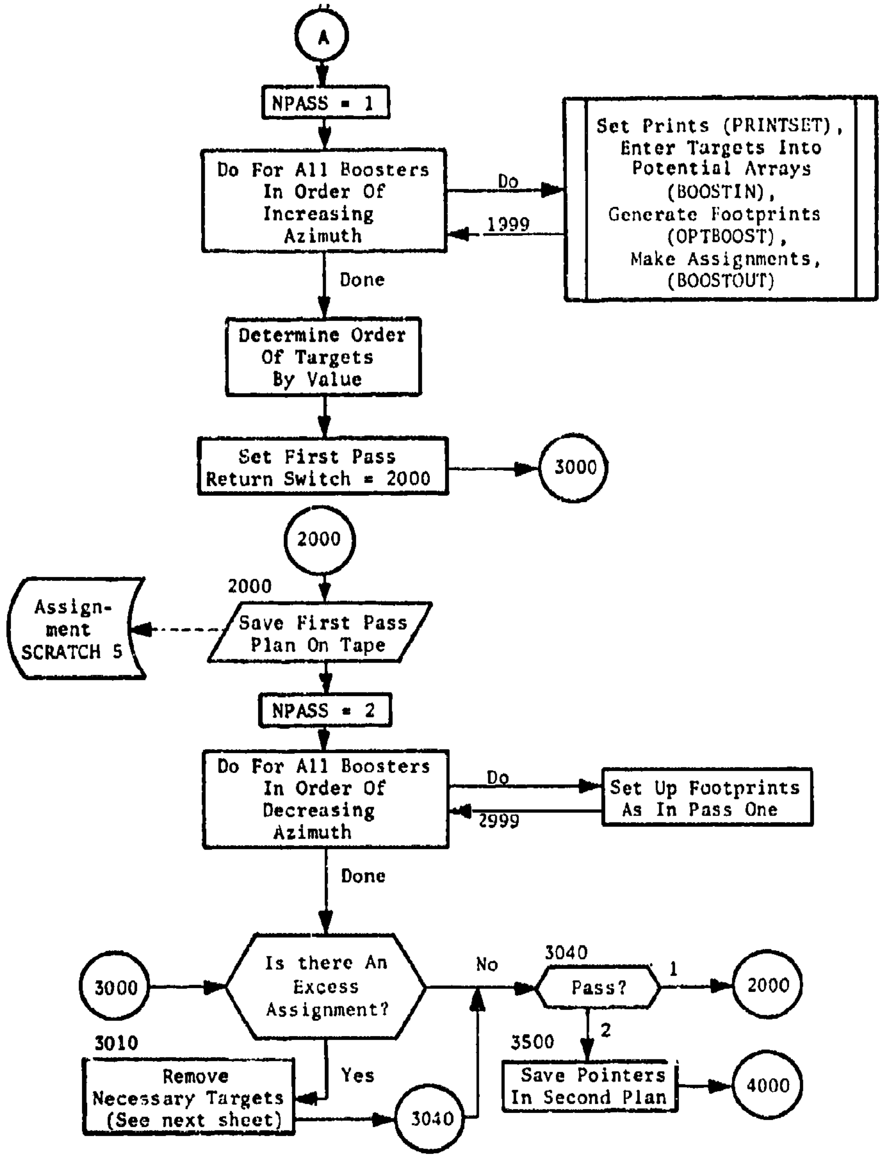


Fig. 84. (cont.)
Part IV: Construct Footprints

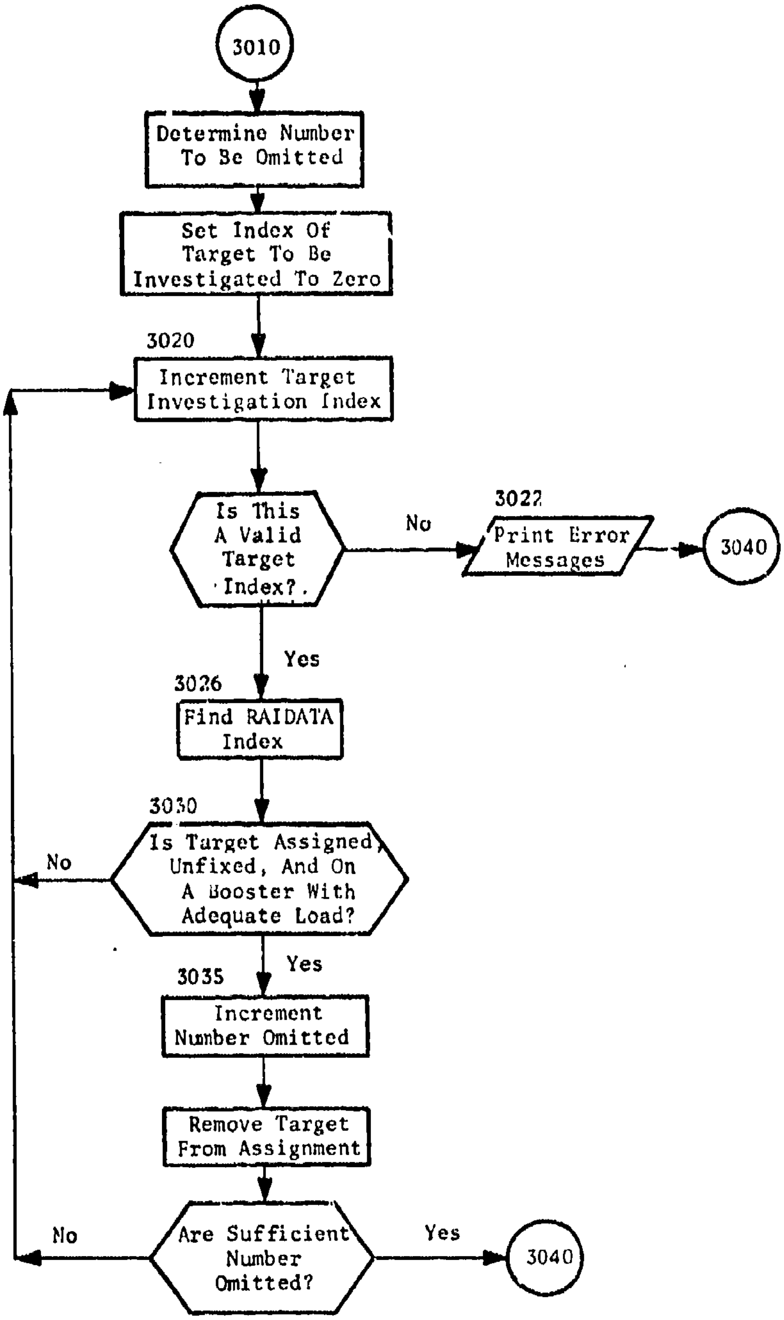


Fig. 84. (cont.)
Part V: Excess Target Removal (Detail)

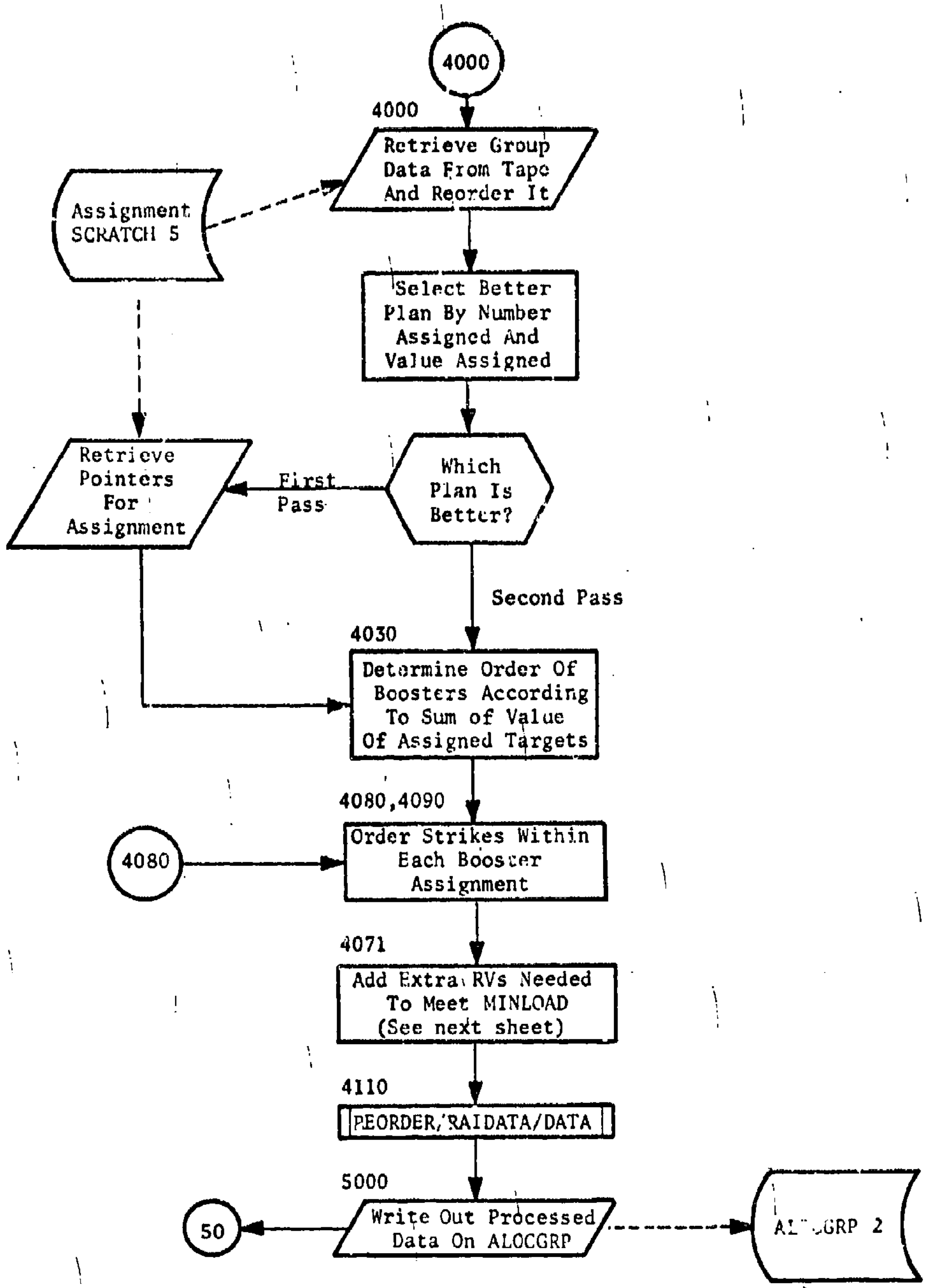


Fig. 84. (cont.)
Part VI: Select and Output Final Plan

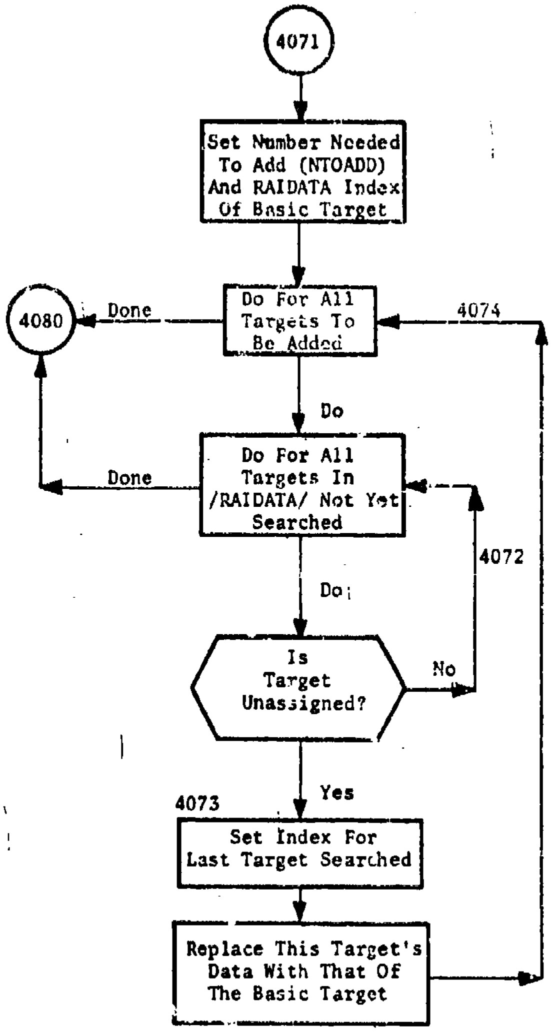


Fig. 84. (cont.)
Part VII: Detail for Adding RVs Required to Meet MINLOAD Constraint

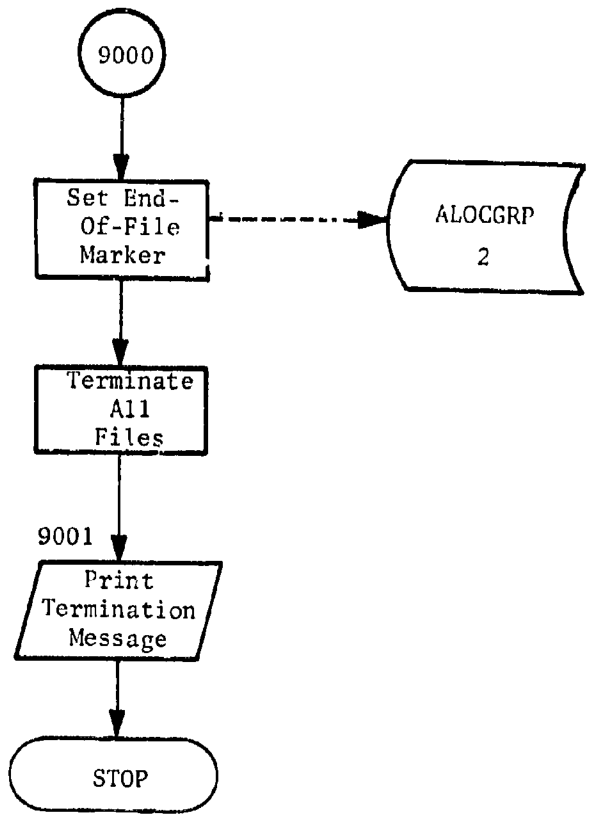


Fig. 84. (cont.)
Part VIII: Termination Block

SUBROUTINE ADDRV

PURPOSE: This routine determines the placement of re-entry vehicles added to a booster assignment solely to meet a minimum load constraint.

ENTRY POINTS: ADDRV

FORMAL PARAMETERS: None

COMMON BLOCKS: 3, DEBUG, FOOTIO, LOADATA, PARAMETR, POTENT, PRINT

SUBROUTINES CALLED: FOOTEST, GOPRINT

CALLED BY: OPTBOOST

Method

If the free booster loading option is not exercised (i.e., LOADOPT$\neq$0), then the program may require that some target points in a footprint receive more then one re-entry vehicle (RV). If so, this subroutine is called to assign the extra RVs to the targets already assigned to the booster.

This subroutine computes the number of RVs to be added, NTOADD. It then sets up an array, NONTAR, which specifies the number of re-entry vehicles assigned to each target in the current hit list. The testing arrays in common /FOOTIO/ are filled by referencing both the hit list and the NONTAR array. In this fashion, the feasibility of adding the extra RVs is tested.

The subroutine begins by adding NTOADD re-entry vehicles to the first target in the footprint. If this allocation is not feasible, ADDRV decrements the number of RVs added to the first target until it reaches a feasible allocation. There is no further processing for this allocation since, if a re-entry vehicle cannot be added to the first target of a footprint, it cannot be added to any later target.

If the total number of added re-entry vehicles could be added to the first target, the subroutine searches for an alternative allocation with less variance in the number of RVs allocated to each target point. (The optimal allocation would have the same number of vehicles assigned to each target in the footprint.) The alternative allocations are constructed by examining the number of vehicles on each target. The targets are examined in order of delivery of their RVs by the final stage of the booster. At the first target where this number decreases (a decreasing step), a vehicle is removed and placed at the last target which has a number allocated less than the preceding target. Figure 85 demonstrates the construction of a series of alternative allocations. If at any time an alternative allocation is infeasible, the subroutine reduces the number of targets to be investigated for addition of RVs and continues processing.

Subroutine ADDRV is illustrated in figure 86.

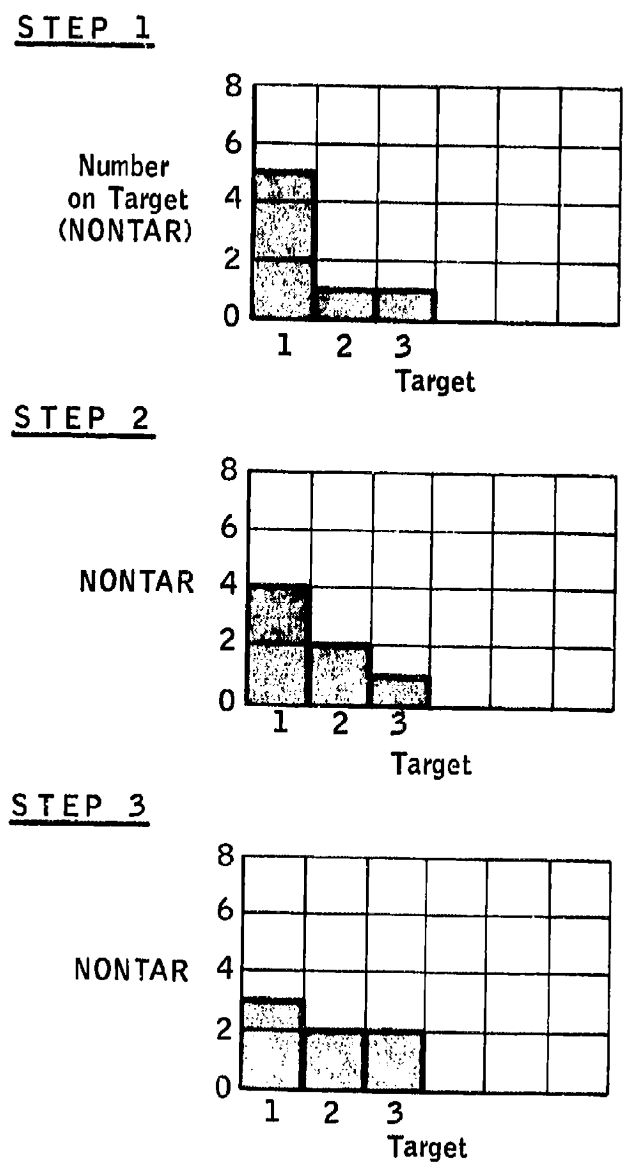


Fig. 85. Extra Re-entry Vehicle Allocation Example

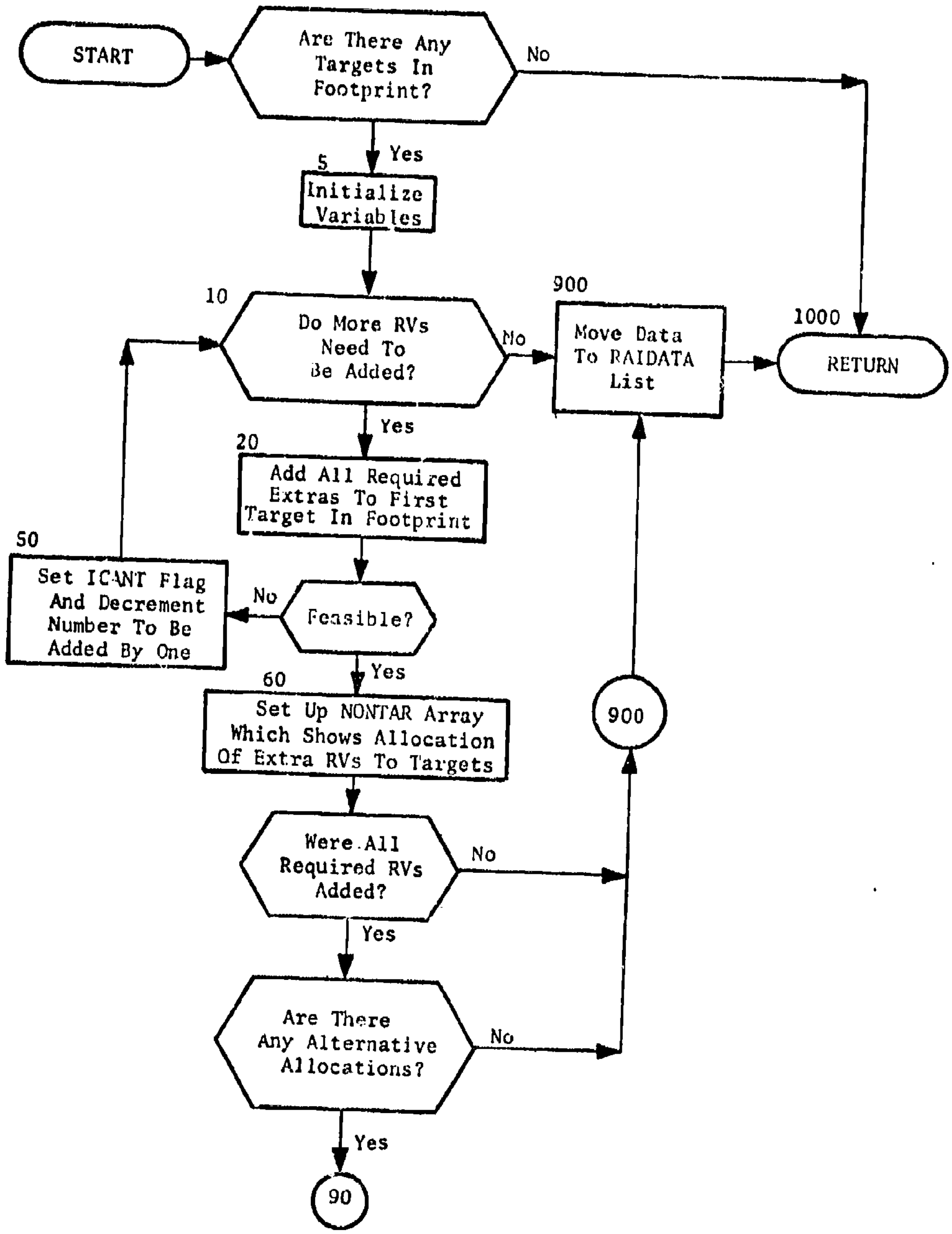


Fig. 86. Subroutine ADDRV
(Sheet 1 of 2)

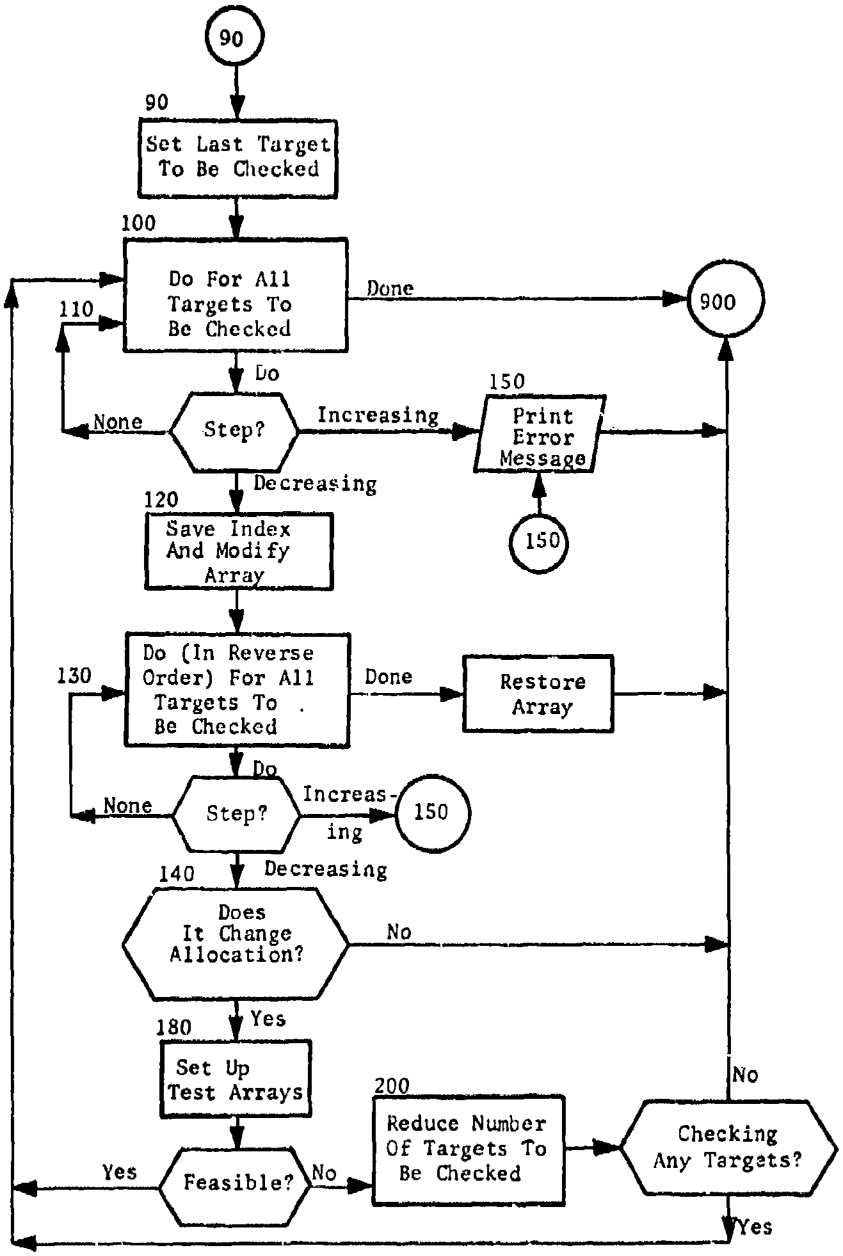


Fig. 86. (cont.)
(Sheet 2 of 2)

SUBROUTINE ASSIGN

PURPOSE:	This routine assigns the entire hit list to the current booster, NUMBOOST.
ENTRY POINTS:	ASSIGN
FORMAL PARAMETERS:	None
COMMON BLOCKS:	RAIDATA, 4, PERFORM, POTENT, WPNTGT, 2, DEBUG, PRINT
SUBROUTINES CALLED:	GOPRINT
CALLED BY:	BOOSTOUT

Method

This routine retrieves the RAIDATA index of each target in the hit list and modifies the pointer in the IFOR, IBACK, IBOOST, and NTB arrays in the RAIDATA lists. It also increments the total number of targets assigned, NASGN, and the total value assigned, VALASGN.

Subroutine ASSIGN is illustrated in figure 87.

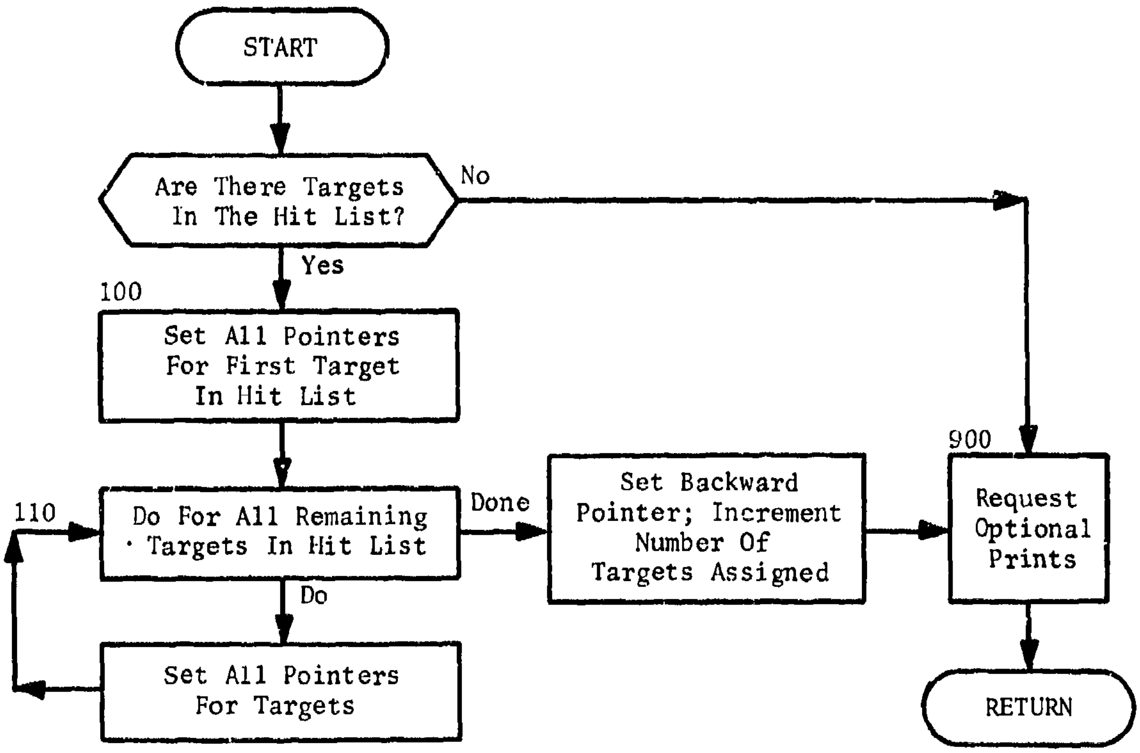


Fig. 87. Subroutine ASSIGN

SUBROUTINE BOOSTIN

PURPOSE: This routine determines the set of potential targets for each booster and computes detailed intertarget parameters for all targets in the potential target list.

ENTRY POINTS: BOOSTIN

FORMAL PARAMETERS: None

COMMON BLOCKS: RAIDATA, 4, CONTROL, DSQUARE, FOOTIO, INDEX, PARAMETR, POTENT, RANGE, VALPARM, WPNTGT, 1, 2, 3, DEBUG, PRINT

SUBROUTINES CALLED: GOPRINT, ORDER, OUTPOT, INPOT, CRSTODWN, UPTODOWN, VALF

CALLED BY: FOOTPRNT

Method

This routine is called once each pass for each booster. Its purpose is to set up the potential target arrays for the booster. Its functions are:

1. Remove targets from the potential target arrays
2. Search for unassigned targets in the neighboring geographic area and place them in potential target arrays
3. Enter targets currently assigned to the booster into the potential target arrays
4. Compute intertarget distance matrix
5. Determine worth of maintaining each target in the array for the next booster processed
6. Compute the worth of starting the footprint with each target.

Processing begins with the search for "lost" targets. These are targets which are currently unassigned to a booster and not in the potential target arrays. This search is done only on the second pass since the initial assignment generated by subroutine INITASGN contains every target. The geographic area to be searched is determined by targets currently assigned to the booster and also the targets assigned to the next booster to be processed. The backward pointer (IBACK) of the first target in each footprint is set to the RAIDATA index of the target with the largest launch azimuth in the footprint. Thus BOOSTIN uses this value of IBACK for each of the two boosters to determine the area of the RAIDATA list to investigate. Any targets in this area which are neither assigned nor in the potential target arrays (i.e., ISTATUS = -2) are placed in the lost target list (LOST) and ISTATUS is set to -1.

The routine now determines which targets in the potential target arrays should be removed to make room for the targets to be entered. The worth of maintaining a target in the potential list is always stored in the diagonal elements of the distance matrix D. The worth of maintaining the target whose POTENT index is J is saved in D(J,J). The number of targets to be dropped is determined by the input parameter PURGE. First the routine computes the number of targets in the POTENT list which were not entered in the list by the look-ahead feature of subroutine OPTBOOST. If this number is less than the average number of targets per booster, no targets are removed. Otherwise the routine omits the fraction, PURGE, of these targets. The targets are omitted in order of increasing worth. If this fraction removed does not leave sufficient room for the current assignment, targets are removed singly until there is sufficient room. The routine then enters the current booster assignment into the list. Finally, as many of the lost targets as possible are entered.

Two sets of intertarget parameters are now computed. First, the intertarget distance matrix is computed. The entries in this array, D, are defined as follows, for targets whose POTENT indices are i and j:

$$D(i,j) = \begin{cases} \text{Square of actual downrange distance from target i to target j (this quantity is set negative if j is uprange of i)} & i < j \\ \text{Worth of maintaining target in potential list} & i = j \\ \text{Square of actual crossrange distance from target i to target j} & i > j \end{cases}$$

The off diagonal terms are computed first by simple geometry. For this calculation, the downrange axis is defined to have the average launch azimuth of all the targets in the list.

The diagonal terms, the worth of maintaining the target in the list, are computed next. In order to keep targets in the arrays for at least two boosters, a target that has just been entered is given an artificially

462-546 O - 72 - 5

high value. For the other targets, this worth is computed according to the formula given in the Analytical Manual.*

The second set of intertarget parameters is the worth of making each target the first target in the footprint. This worth will be used by subroutine OPTBOOST to initiate footprint generation. When computing the equivalent downrange distances, the downrange axis is defined by the launch azimuth to the target currently being considered for selection as the first target. The booster load is assumed to be the average number of targets per booster. Multiple calls on function VALF are made for each target. The Analytical Manual shows the formula which defines this worth.*

Subroutine BOOSTIN is illustrated in figure 88.

---

*Volume II, Plan Generation Subsystem, Chapter 2, Analytical Concepts and Techniques, Basic Sortie Generation, MIRV Missile Plans, Value of Assigning a Target to a Booster.

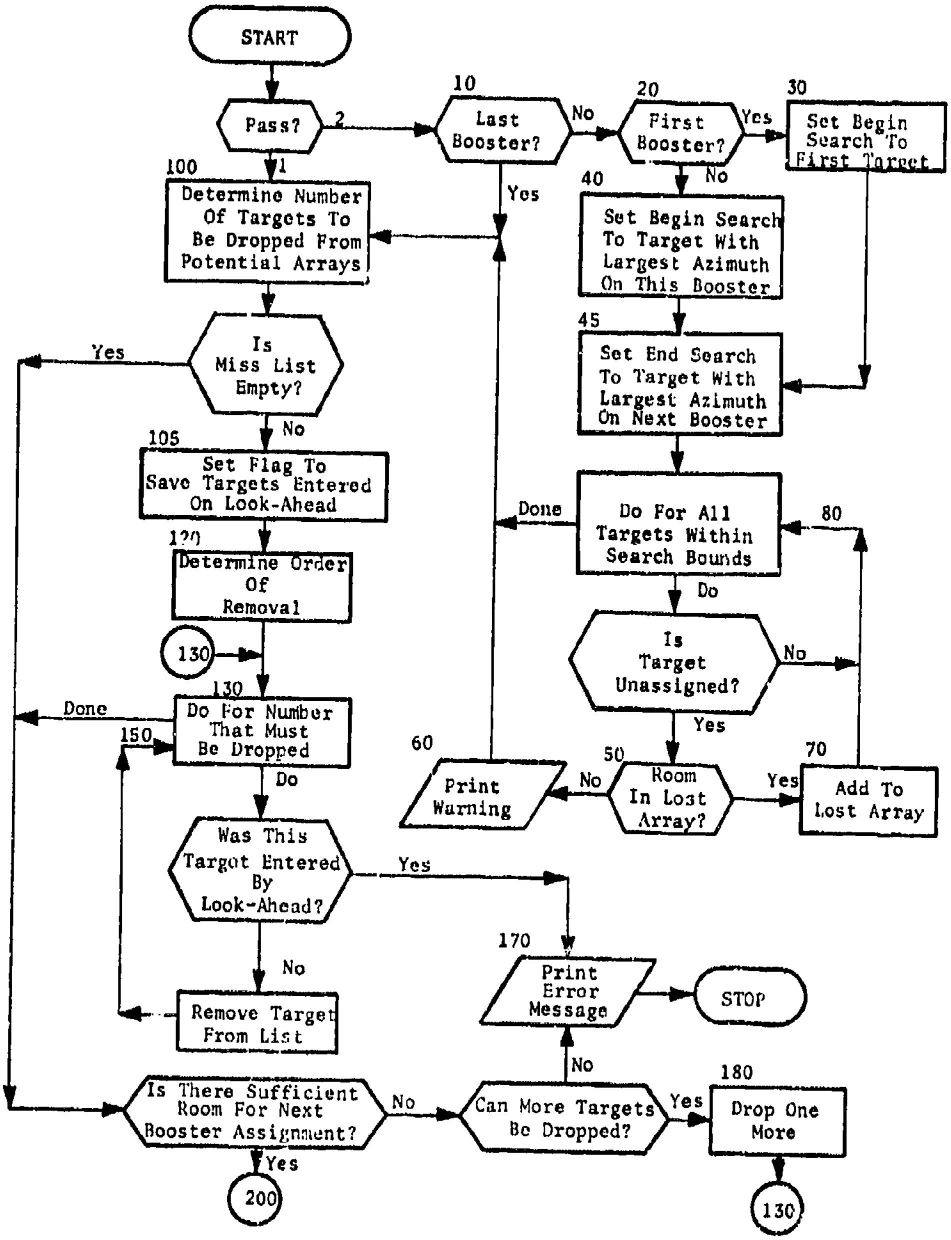


Fig. 88. Subroutine BOOSTIN
(Sheet 1 of 4)

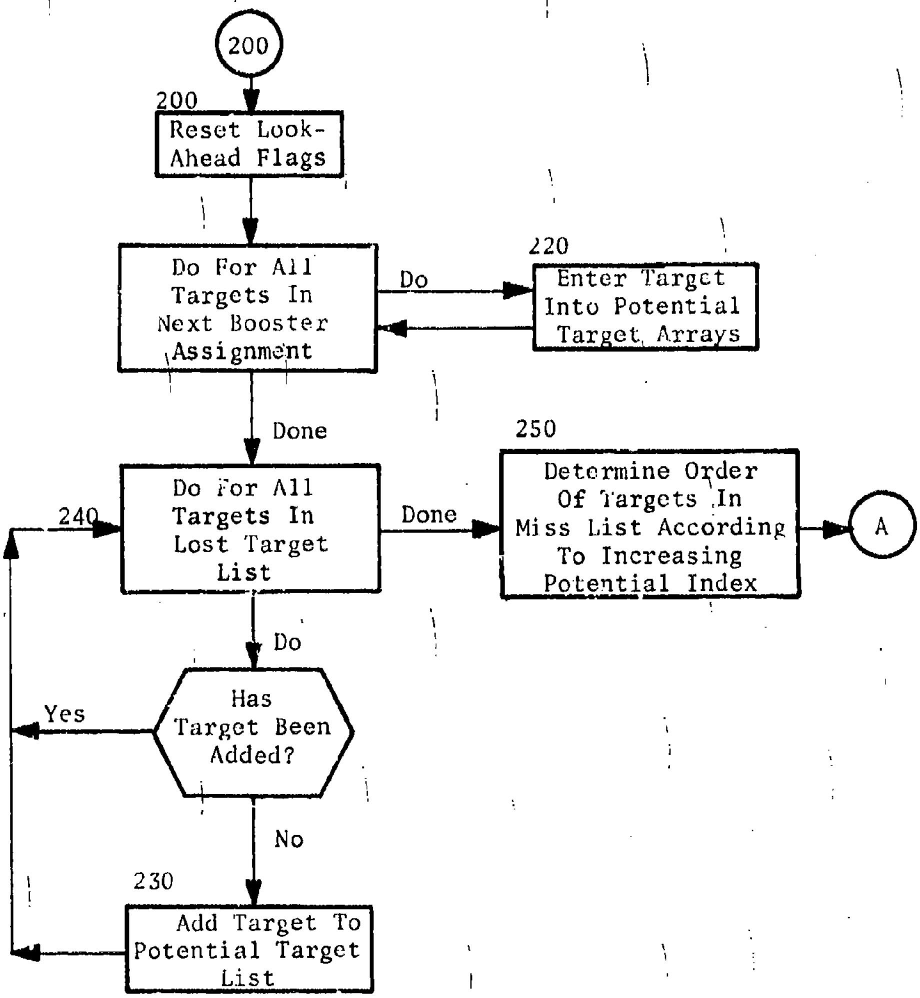


Fig. 88. (cont.)
(Sheet 2 of 4)

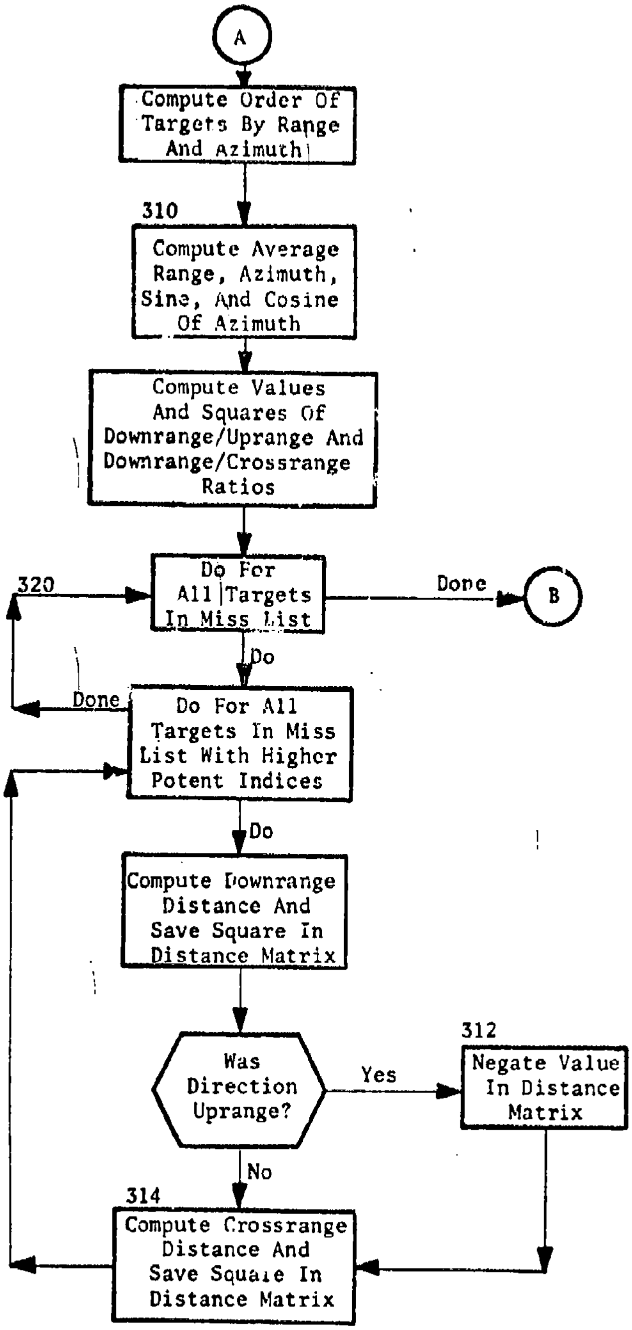


Fig. 88. (cont.)
(Sheet 3 of 4)

Preceding page blank

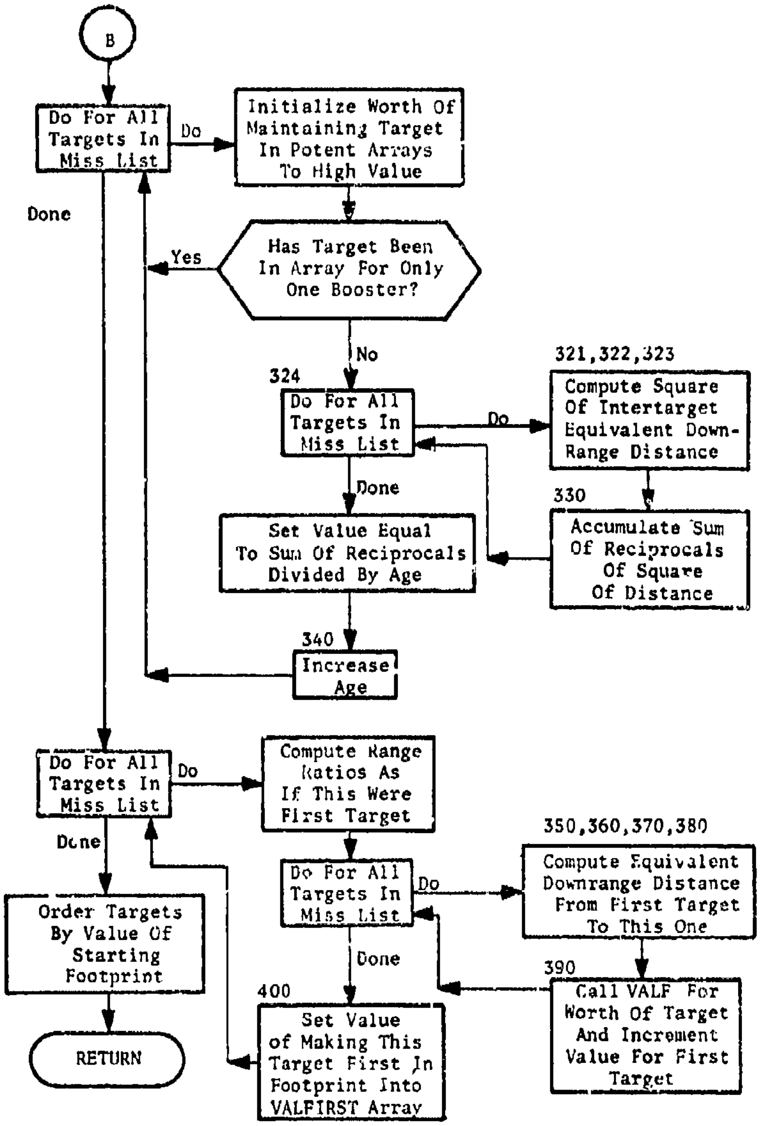


Fig. 88. (cont.)
(Sheet 4 of 4)

SUBROUTINE BOOSTOUT

PURPOSE:	This routine processes the hit list to assign targets to boosters and returns potent indices for use.
ENTRY POINTS:	BOOSTOUT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	POTENT, 1, 3, DEBUG, PRINT
SUBROUTINES CALLED:	ASSIGN, GOPRINT
CALLED BY:	FOOTPRNT

Method

Subroutine ASSIGN is called to assign the entire hit list to the booster. Subroutine BOOSTOUT then resets all the indices and pointers in the potential arrays for each target in the hit list. For each of these targets, the number of available positions in the POTENT list (NFREE) is incremented by one and the POTENT index of the target is placed at the end of the available list, IFREE.

Subroutine BOOSTOUT is illustrated in figure 89.

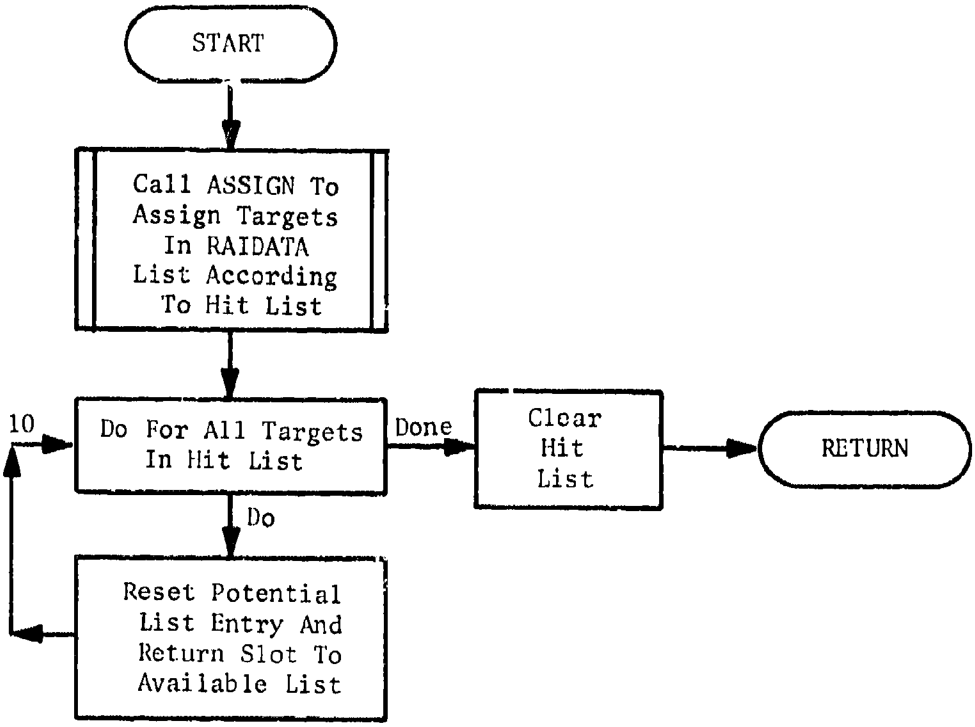


Fig. 89. Subroutine BOOSTOUT

SUBROUTINE CHKSEQ

PURPOSE: This routine determines if a reordering of the hit list will reduce the total fuel used and, if so, reorders the list.

ENTRY POINTS: CHKSEQ

FORMAL PARAMETERS: None

COMMON BLOCKS: FOOTIO, POTENT, WPNTGT, 1, DEBUG, PRINT

SUBROUTINES CALLED: GOPRINT, FLYDIST, MISSIT, HITIT, TEST

CALLED BY: EVAL

Method

This routine tests the sequence of targets in the hit list. It takes each consecutive pair of targets in the list and determines if the total equivalent downrange distance for the assignment would decrease if the order of delivery to the pair were inverted. If so, then it tests the fuel use of the inverted target ordering. If the new order uses less fuel than the old order, the new order is retained. Otherwise, the old order is restored. After checking the sequence of all consecutive pairs, the routine checks the number of inversions performed in the last test of all consecutive pairs. If any inversions were made, the process is repeated. This continues until no further improvement is possible.

Subroutine CHKSEQ is illustrated in figure 90.

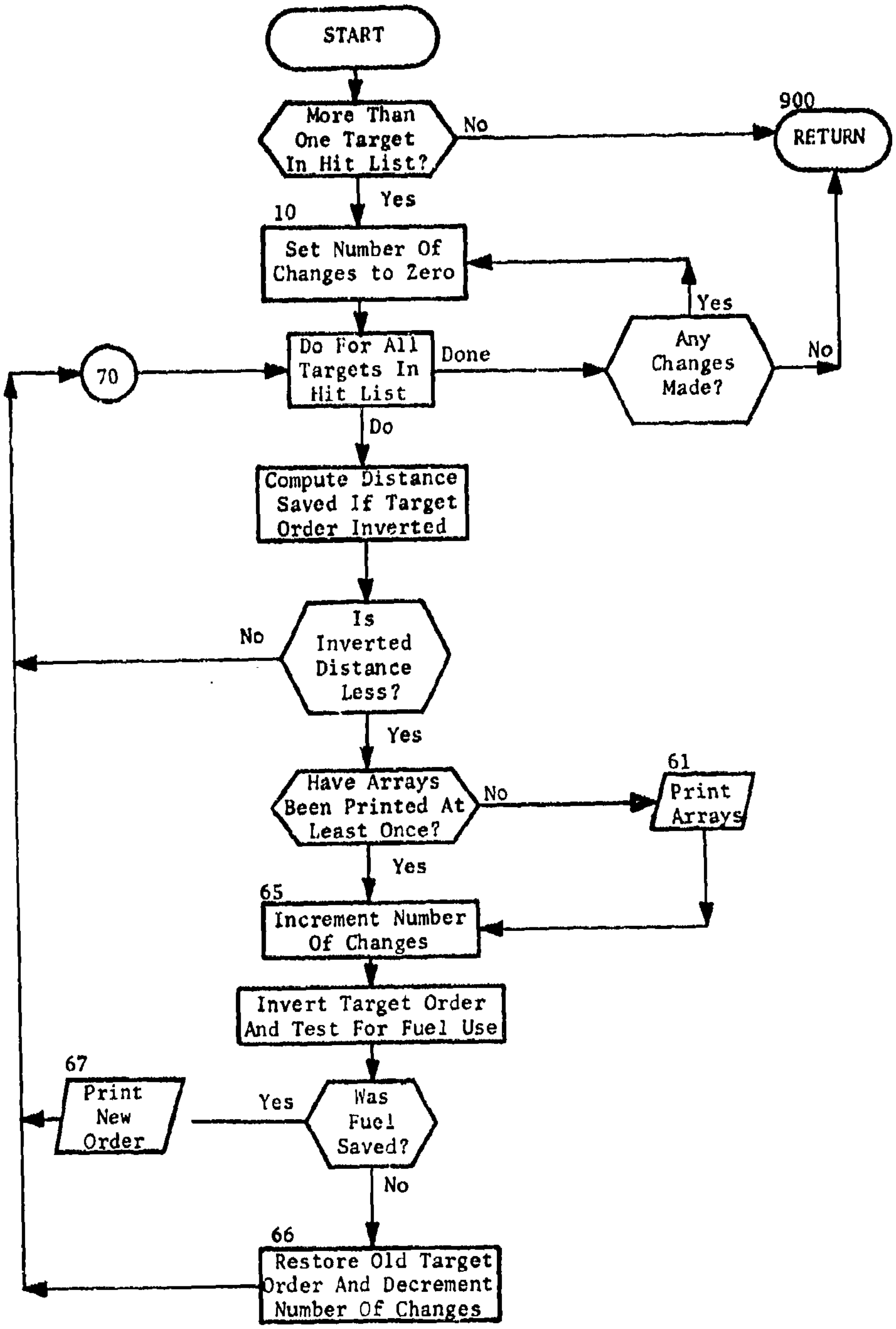


Fig. 90. Subroutine CHKSEQ

FUNCTION CRSTODWN

PURPOSE:	This routine computes crossrange-downrange ratios.
ENTRY POINTS:	CRSTODWN
FORMAL PARAMETERS:	I = System type index (MTYPE)
	R = Range to first target (nautical miles)
	AZ = Launch azimuth to first target
	N = Number of re-entry vehicles carried
COMMON BLOCKS:	FOOTDATA, SHRTDAT, PENADD
SUBROUTINES CALLED:	None
CALLED BY:	BOOSTIN, EVAL, FOOTEST

Method

This function simply applies the crossrange-downrange ratio equations whose parameters were input in subroutine TABLINPT. The formal parameters are the system type number (MTYPE), the range and azimuth to the first target, and the number of re-entry vehicles currently on board.

Function CRSTODWN is illustrated in figure 91.

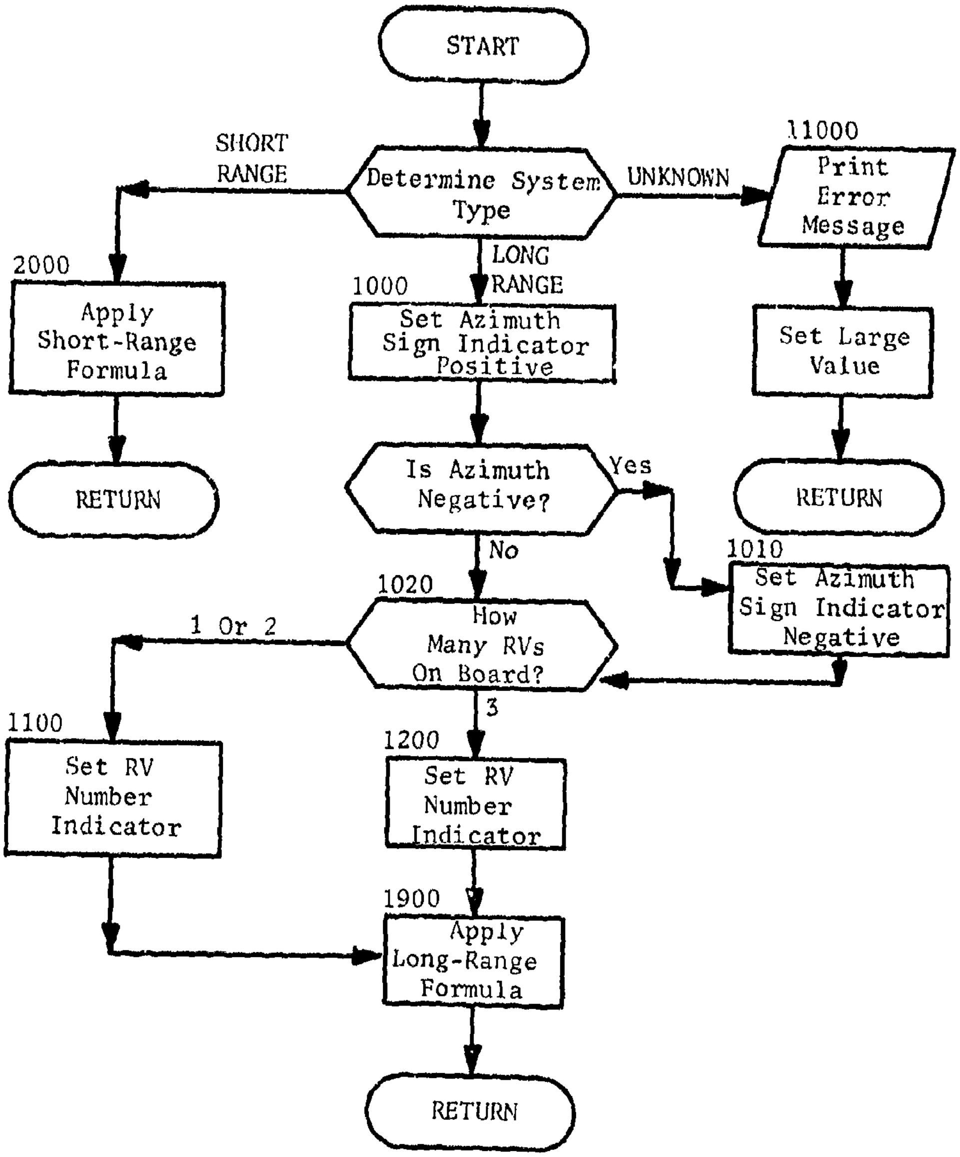


Fig. 91. Function CRSTODWN

SUBROUTINE EVAL

PURPOSE:	This routine recomputes the value arrays for each change in the potential target list. Entry REVAL is used to recompute the intertarget distance matrix.
ENTRY POINTS:	EVAL, REVAL
FORMAL PARAMETERS:	None
COMMON BLOCKS:	RAIDATA, 4, CONTROL, DSQUARE, FOOTIO, PARAMETR, POTENT, RANGE, VALPARM, 1, 2, 3, DEBUG, PRINT
SUBROUTINES CALLED:	GOPRINT, CRSTODWN, UPTODOWN, CHKSEQ, FLYDIST, VALF, ORDER, REORDER
CALLED BY:	OPTBOOST, IMPROVE

Method

This routine has three functions:

1. To determine placement in the hit list of each unassigned target if it were placed in the current footprint
2. To calculate the worth of adding each target (individually) to the current footprint
3. To recompute the intertarget distance matrix if necessary.

The determination of the correct placement in the current footprint of an additional target uses an approximation to a minimal increased fuel consumption criterion. A target is placed in sequence so as to require the minimum increase in fuel use. The procedure for this calculation is exercised once for each target in the miss list. Every possible footprint is tested to determine the placement of the target in the footprint with the maximum ratio of remaining available distance to maximum allowable distance in the list.

Assume a test on the placement of the new target K between targets J and L of the current footprint. Previous operations in subroutine FOOTEST have defined the following variables.

DELRAFT(J) - The maximum increased equivalent downrange distance that could be traveled after target J (and before target L) that would still allow completion of the footprint within fuel constraints.

TOFLY(J) - The equivalent downrange distance from target J to target L.

Function FLYDIST is defined as follows:

FLYDIST(a,b) = Equivalent downrange distance between targets a and b.

For example, in this case TOFLY(J) = FLYDIST(J,L).

Then, the remaining available distance for insertion of target K between targets J and L is defined as follows:

DLEFT = DELRAFT(J)-[FLYDIST(J,K)+FLYDIST(K,L)-TOFLY(J)]

The ratio which EVAL tests is

DLEFT/DELRAFT(J).

The placement with the maximum value of this ratio is selected as the best. Note that these equations are suitably modified for the cases of placement as the first or last target in the footprint.

After selection of the proper sequence for a target, this routine computes the worth of adding it to the footprint. This procedure for each target requires a call on function VALF for each target in the miss list. As explained in the Analytical Manual,* this function returns a value related to a ratio of distances and a distance weighting function. The ratios input by EVAL when evaluating target K are

FLYDIST(K,M)/DMAX

where M is the index of a target in the miss list and DMAX is the maximum available remaining distance (DLEFT) determined previously for target K. This ratio is computed for each target on the miss list, and the results of the VALF calls multiplied by the relative target values RVAL are summed to provide the total worth of adding the target to the footprint. Before returning to the calling program, subroutine EVAL computes the order of targets in the miss list according to decreasing total worth. This ordering is stored in the NDEXVAL array.

The REVAL entry point is used whenever the first target in a footprint has been changed. If the launch azimuth for this new first target is significantly different (>.01 radians) from the launch azimuth of the previous first target, then the downrange axis is redefined and this part of the subroutine recomputes the intertarget distance matrix.

Subroutine EVAL is illustrated in figure 92.

---

*Volume II, Plan Generation Subsystem, Chapter 2, Analytical Concepts and Techniques, Basic Sortie Generation, MIRV Missile Plans, Value of Assigning a Target to a Booster.

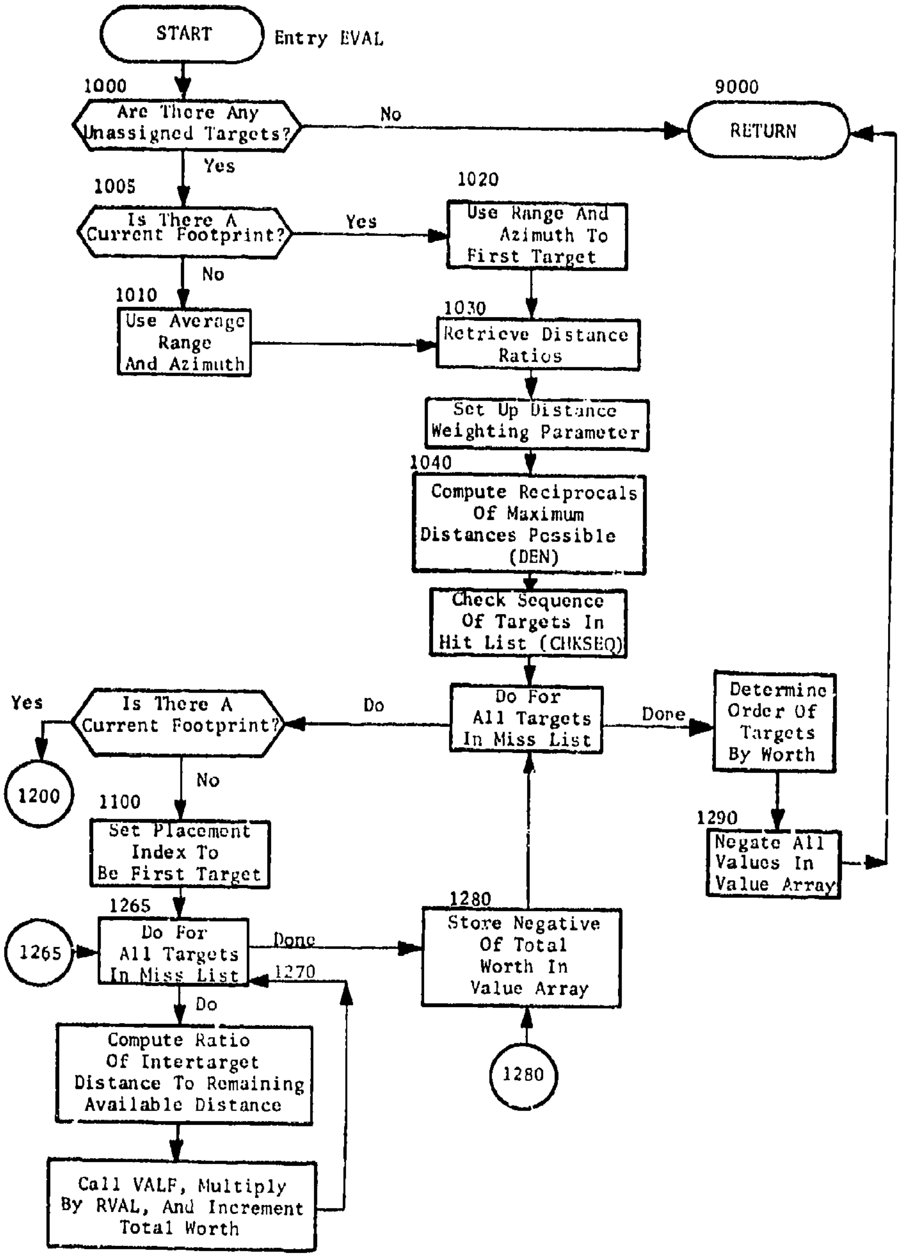


Fig. 92. Subroutine EVAL
Part I: Entry EVAL

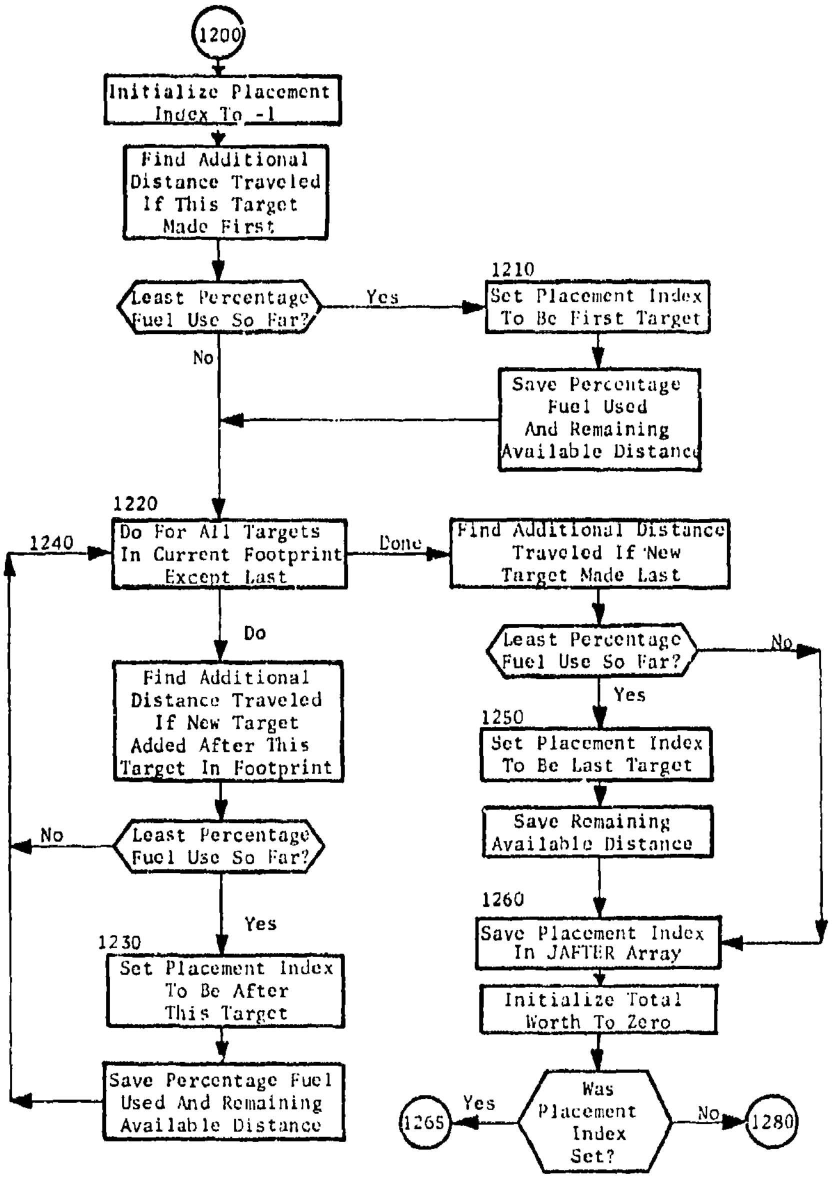


Fig. 92. (cont.)
Part II: Determination of Target Sequence

462-546 O - 72 - 6

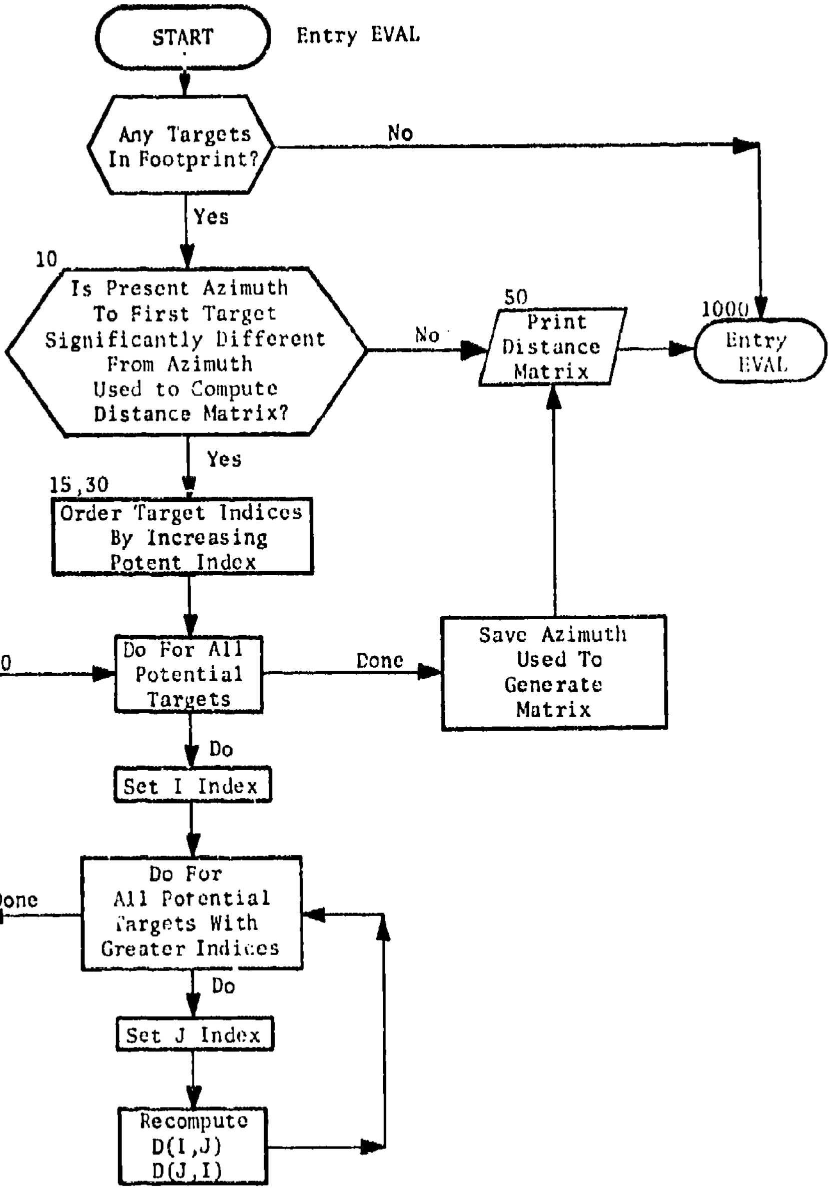


Fig. 92. (cont.)
Part III: Entry REVAL

FUNCTION FLYDIST

PURPOSE: This routine computes equivalent downrange distance.

ENTRY POINTS: FLYDIST

FORMAL PARAMETERS: N - POTENT index of first target

M - POTENT index of second target

COMMON BLOCKS: DSQUARE, RANGE, 1

SUBROUTINES CALLED: None

CALLED BY: CHKSEQ, EVAL

Method

This function uses the intertarget distance matrix D to compute the equivalent downrange distance from the target with POTENT index N to the target with POTENT index M. It assumes that functions UPTODOWN and CRSTODWN have already been called to load the correct range ratio parameters into common /DSQUARE/. This function merely manipulates the data in the distance matrix to provide the calculations for equivalent downrange distance as described in the Analytical Manual, Volume II, Plan Generation Subsystem, Chapter 2, Analytical Concepts and Techniques, Basic Sortie Generation, MIRV Missile Plans, Equivalent Downrange Distance.

This function is called an extremely large number of times in any run of program FOOTPRNT. Up to 15% of the total execution time may be spent in this function and in the SQRT function called by this function. It is therefore very important that the execution time efficiency of this function be maintained.

Function FLYDIST is illustrated in figure 93.

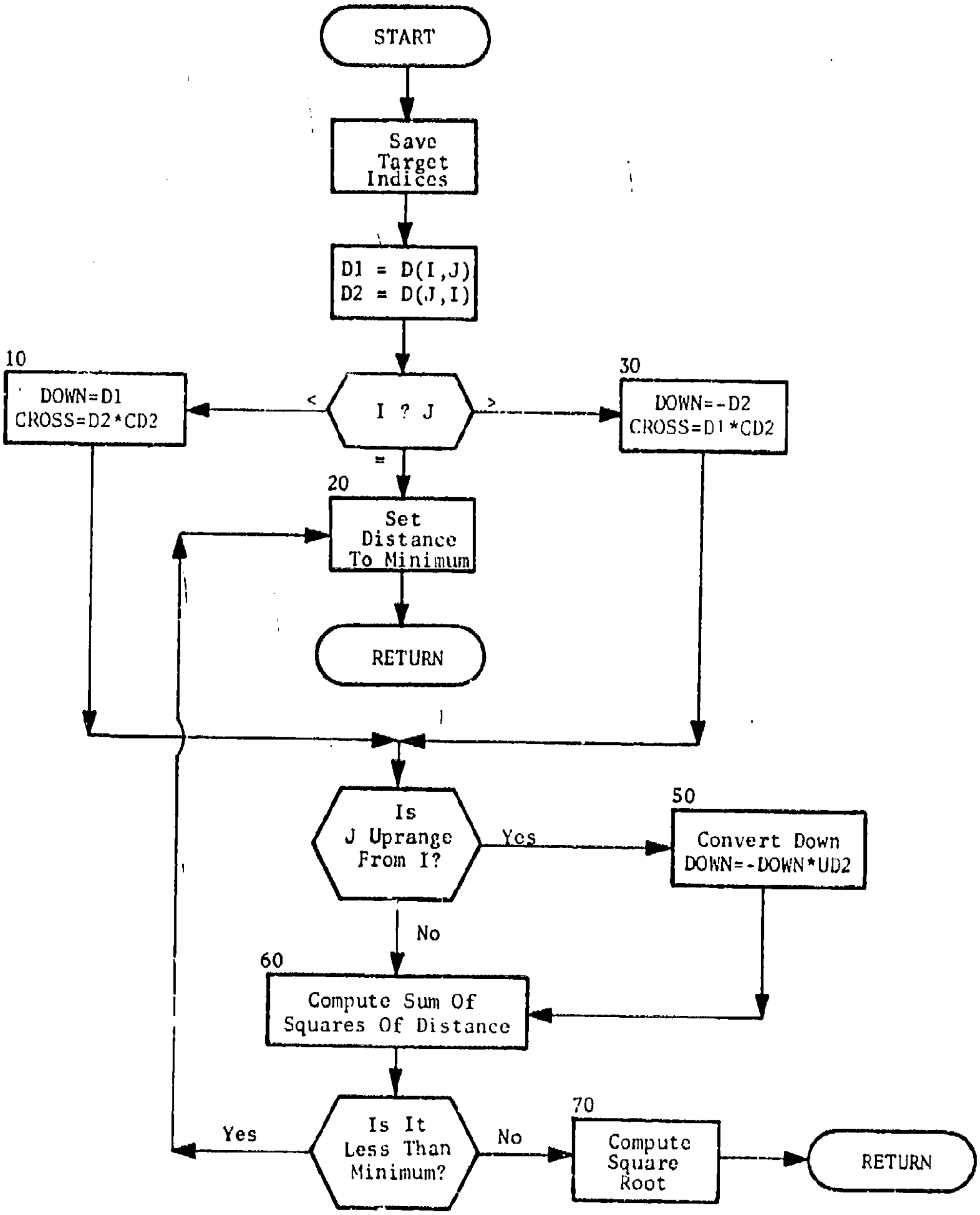


Fig. 93. Function FLYDIST

SUBROUTINE FOOTEST

PURPOSE:	This routine tests footprint feasibility.
ENTRY POINTS:	FOOTEST
FORMAL PARAMETERS:	None
COMMON BLOCKS:	FOOTIO, PARAMETR, POTENT, RANGE, 1, FOOTDATA, SHRTDAT, PENADD
SUBROUTINES CALLED:	CRSTODWN, UPTODOWN, ABORT
CALLED BY:	ADDRV, TEST

Method

Most of the input/output for this subroutine is contained in common /FOOTIO/. Data on the target set to be tested are contained in the arrays RIN and THIN (for range and launch azimuth, respectively). Subroutine FOOTEST computes the equivalent downrange distance between each successive target in the footprint. It then determines the number of RVs that can be delivered to the target set without violating the fuel consumption constraints. If an RV can be delivered to each target in the set, then this subroutine computes the effect of using the total remaining fuel load to deliver one more RV. It outputs the maximum equivalent downrange distance that the remaining fuel will allow from each point in the footprint.

Figure 94 displays the processing flow for this routine. Since the methods used to test footprints are essentially the same for both long-range and short-range systems, only the long-range method will be described. The short-range system method differs only in the details of processing.

Part I: Distance Computation

Before testing the footprint for feasibility, the routine calls functions CRSTODWN and UPTODOWN to retrieve the correct downrange-crossrange and downrange-uprange ratios. Then it computes the equivalent downrange distance between successive targets in the footprint. These distances are placed in array DT, which is equivalenced to array TOFLY in common /POTENT/. This subroutine does not use function FLYDIST for the distance computation, but rather computes the equivalent distance from the basic range and azimuth data. There are two reasons for this independent calculation. First, this subroutine can test any data contained in common /FOOTIO/, without considering the data in the potential target arrays. Second, the footprint testing subprograms, FOOTEST, CRSTODWN, UPTODOWN, SETDATA, and TABLINPT, which comprise the testing module, were designed to be as separate as possible from the other subprograms. This modular design allows for modification of either the assignment module subprograms or the test module subprograms without excessive manipulation of the interface between the modules.

Part II, Sheet 1: Long-Range System

In the following discussion, a "leg" of a footprint refers to the line between two successive targets in the footprint. The Jth leg will be the line between target J and target J+1. The length of these legs (in terms of equivalent downrange distance) determines the feasibility of the footprint.

The testing algorithm (for the long-range system) begins with a determination of the number of re-entry vehicles in the footprint. A number indicator JRV (NRV for short-range) is set to specify the correct set of footprint parameter constraint equations to be used. These equations will vary according to the number of RVs on board the bus. In order to save processing time, FOOTEST precomputes all the necessary fuel consumption and booster range parameters and stores them in a set of temporary arrays (e.g., RSAVE, REXSAVE, and CSAVE). These parameters will change only if there is a new first target in the footprint with a significantly different range (or azimuth for the long-range system). Thus, on each call to FOOTEST, the range and azimuth of the first target are tested against the previous values for these factors. If either factor differs from the saved value by an amount greater than or equal to PDIFF (a preset test variable), then the range and fuel parameters are recomputed. The larger the preset value of PDIFF, the fewer times these parameters will be recomputed.

If the long range system has a full load of penetration aids (i.e., MTYPE = 3), then the fuel load at booster separation is computed by a special set of equations (statement 3000).

The main testing algorithm begins at statement 1308 (2008 for short-range) with a calculation of the total fuel load available for footprinting and the maximum booster range. If the range to the first target exceeds the maximum booster range, some of the fuel is used for range extension. This range extension fuel is subtracted from the total load available for footprinting. If this subtraction results in a negative fuel load then the subroutine returns with the feasibility indicator, IFEAS, set to 0.

This variable contains the number of targets in the footprint that can be reached within fuel constraints.

Part II, Sheet 2: Long-Range System (continued)

If there is some fuel left for use on the legs, the feasibility indicator is set to 1 and the number of re-entry vehicles to be delivered NTOGO is set to the original number minus one (since one RV has been delivered to the first target point). FOOTEST then computes the fuel use on each leg. It retrieves the correct fuel consumption rate for the current load and the equivalent downrange distance for the leg. A division gives the amount of fuel used on the leg. If there is not sufficient fuel left for that leg, the fuel remaining indicator FUELEFT is set to 0 (statement 1365), and the routine returns control to the calling program. If there is sufficient fuel for the leg, the fuel remaining is decremented by the fuel used on the leg, the feasibility indicator is incremented, and the number of RVs on board is decremented. Then the next leg is tested.

When all the legs have been tested the fuel left after footprinting FUELEFT is saved (statements 1390 to 2060). If the booster is currently carrying the maximum allowed load, control returns to the calling program. If more re-entry vehicles can be added, the best use of the extra fuel is calculated (starting at statements 1396 or 2070).

Part II, Sheet 3: Long-Range System (continued)

This section begins by resetting the initial load indicator to show the potential addition of another re-entry vehicle to the original payload. (If necessary, the number indicator, JLV or NRV, is reset.) The same computations as were done previously to test the footprint are repeated with the increased load. This time, however, the difference in fuel use between the original load and the increased load is saved in array EXTRA. The value of the element EXTRA(J) is the amount of extra fuel that would be needed on leg J (from target J to target J+1) to carry one more RV. These computations are performed in the "do loop" ending on statement 1420 (or 2100 for short-range).

Then, this extra required fuel is subtracted, cumulatively, from the fuel left after completion of the footprint ("do loop" ending on statements 1430 or 2110). The elements of the array EXTRA are changed to contain the successive results of these subtractions. The contents of EXTRA(J) now contain the fuel that would be available for further footprinting if a new target were added to the footprint between target J and target J+1. The algorithm assumes the extra re-entry vehicle is carried on the bus for the deliveries through target J. Then the extra re-entry vehicle is delivered to another target and the bus proceeds as before. The amount of fuel that could be used for the extra flying distance created by insertion of a new target is now contained in the EXTRA array.

This extra fuel available for further footprinting is now converted into a maximum allowable distance. The testing algorithm assumes that all the extra fuel would be used by the bus to deliver the added re-entry vehicle. (Note that the fuel needed to complete the footprint after that delivery is reserved and cannot be used for the addition.) Thus, the extra fuel for each leg is multiplied by the saved consumption rate for each leg, CR, to calculate the maximum extra flying distance allowed on the leg. This distance is stored in array DELRAFT in common /FOOTIO/. Subroutine EVAL uses this distance to determine the worth of adding a new target to the footprint. Since other subprograms divide by these distances, the values placed in the array are increased to a minimum nonzero value (EPSILON, preset to .00000001) to allow that division. This completes footprint testing and control returns to the calling program.

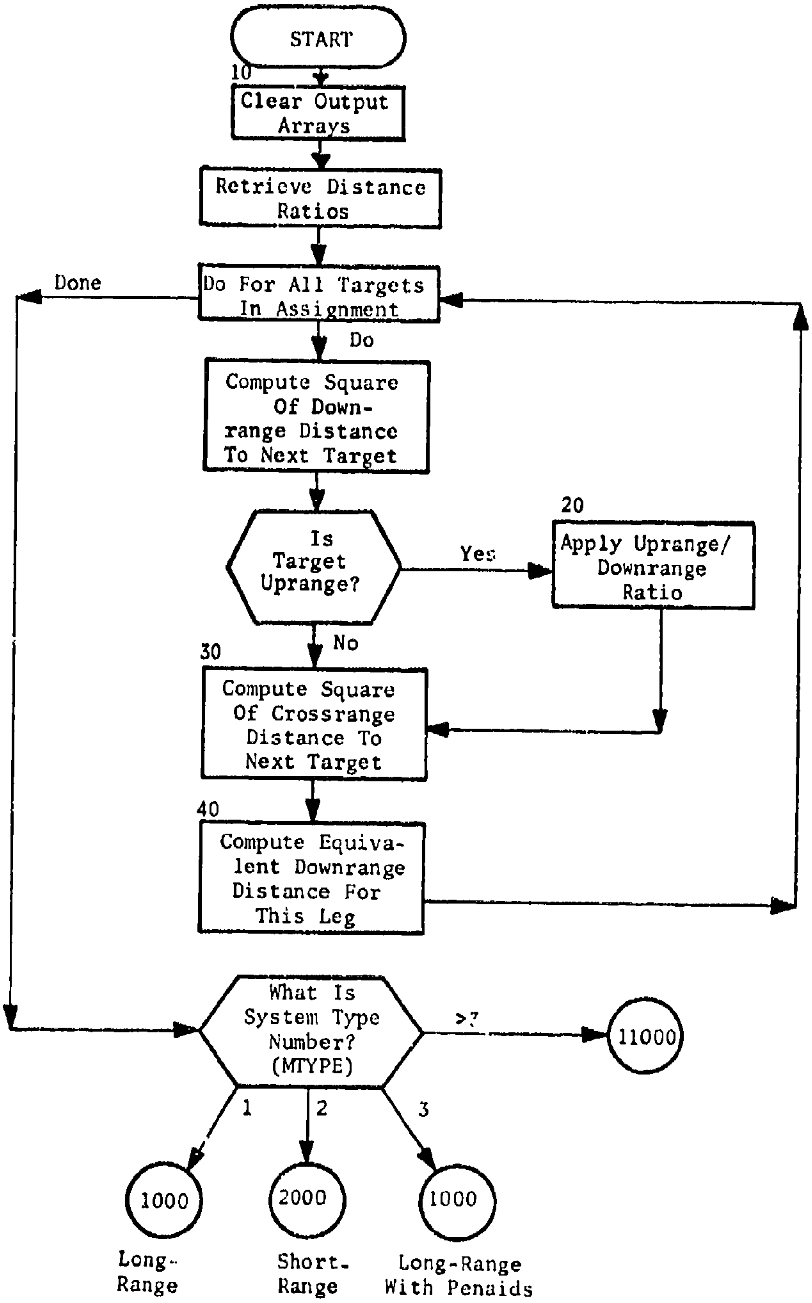


Fig. 94. Subroutine FOOTEST
Part I: Distance Computation

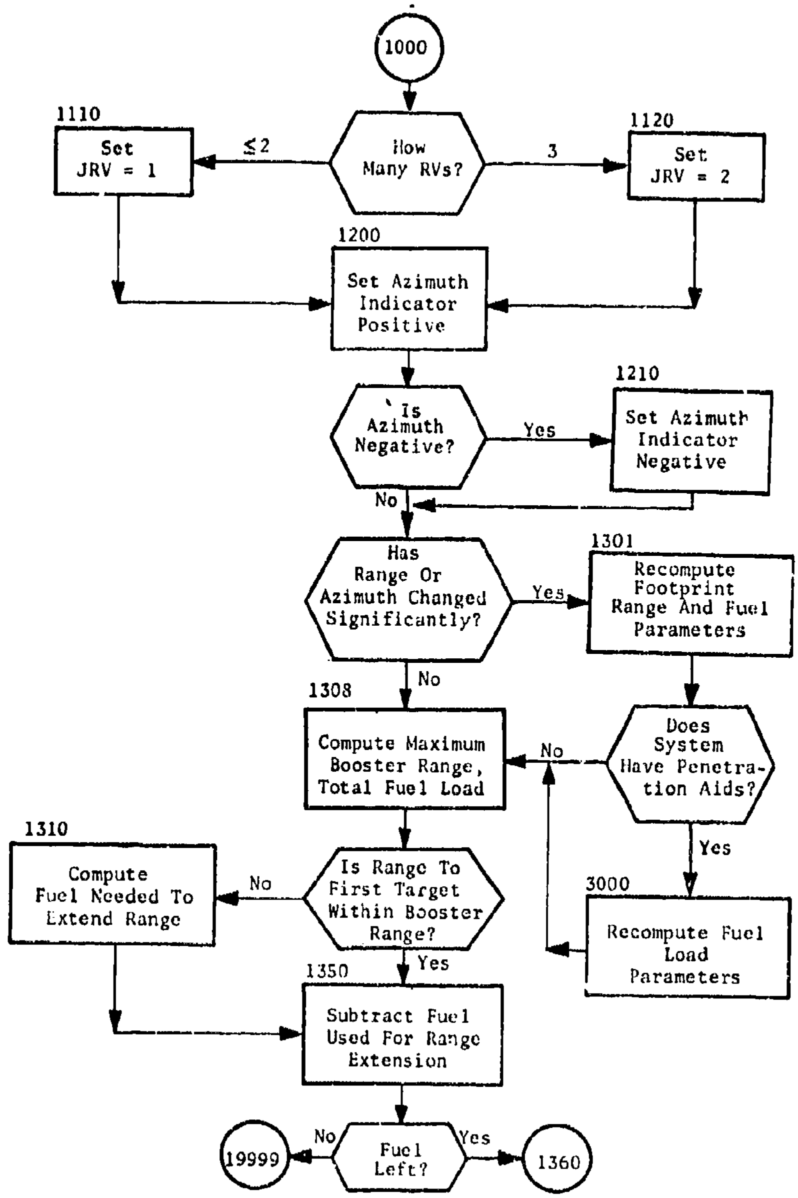


Fig. 94. (cont.)
Part II: Long-Range System
(Sheet 1 of 3)

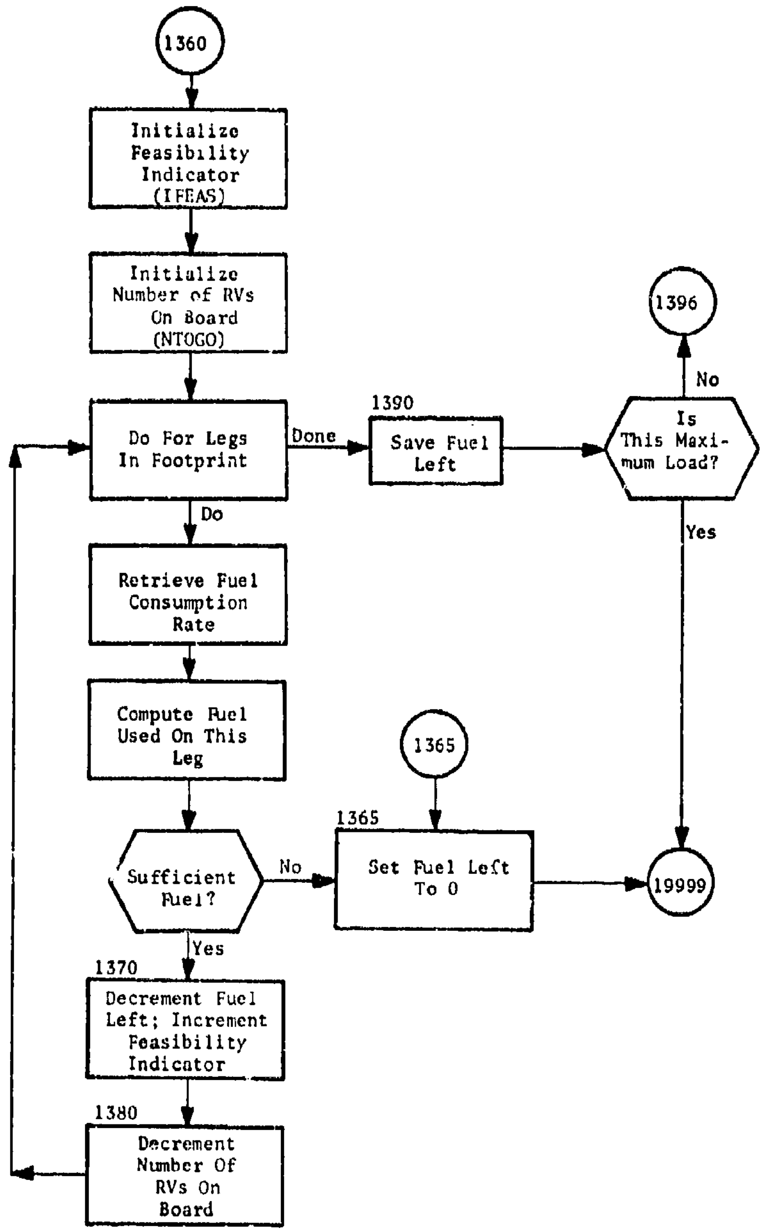


Fig. 94. (cont.)
Part II: (cont.)
(Sheet 2 of 3)

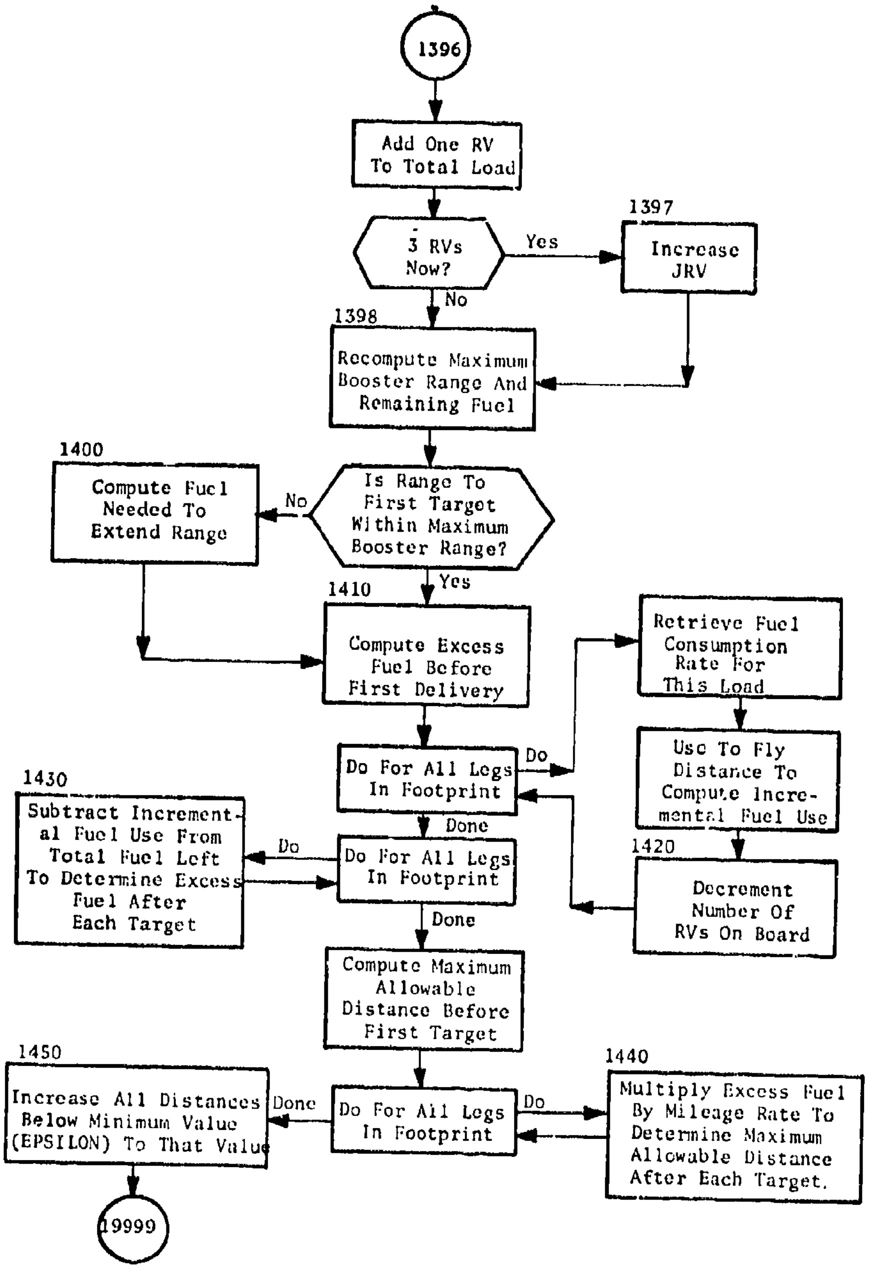


Fig. 94. (cont.)
Part II: (cont.)
(Sheet 3 of 3)

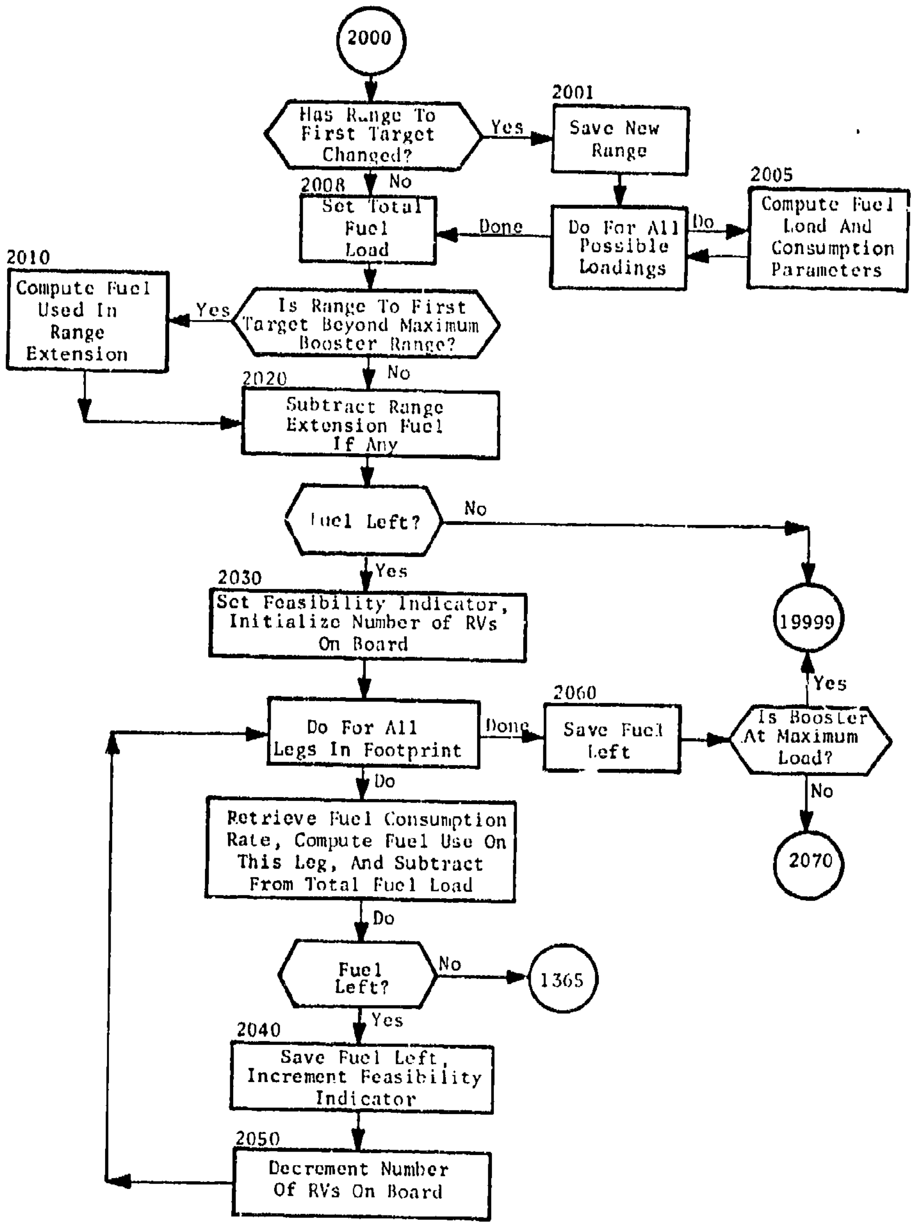


Fig. 94. (cont.)
Part III: Short-Range System
(Sheet 1 of 2)

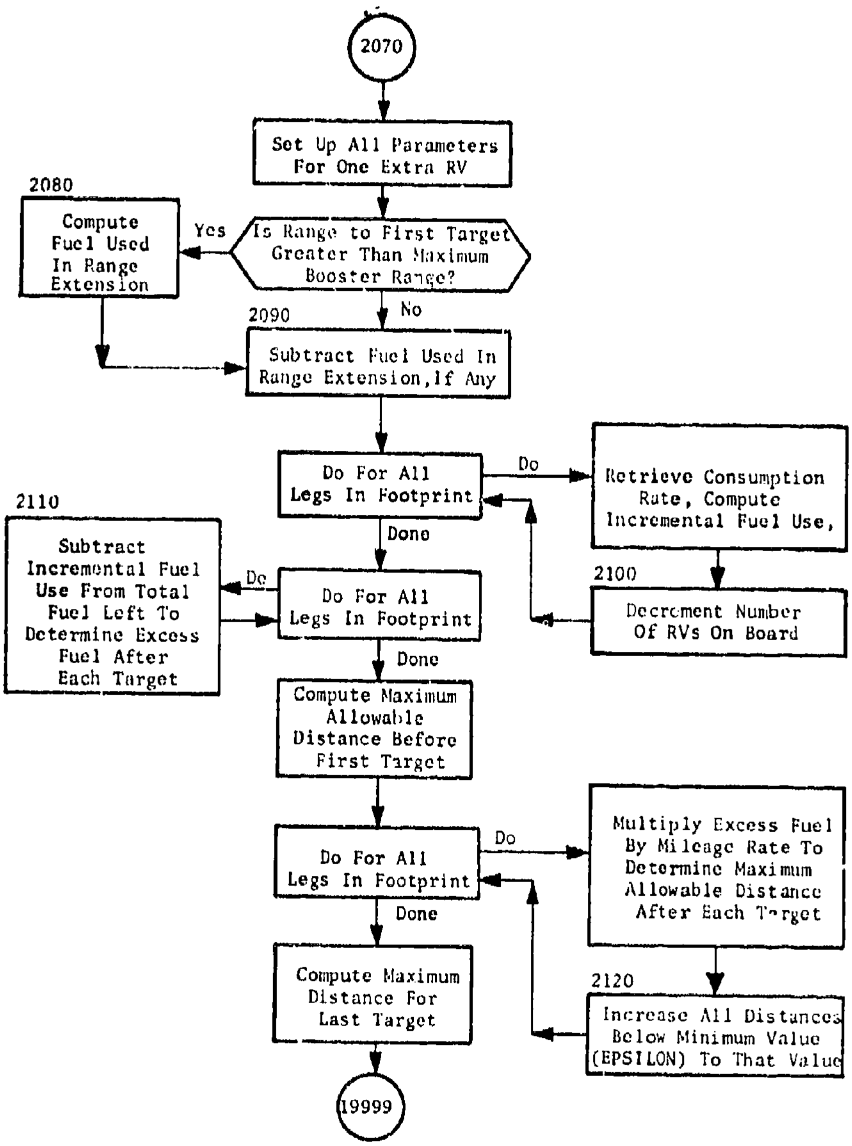


Fig. 94. (cont.)
Part III: (cont.)
(Sheet 2 of 2)

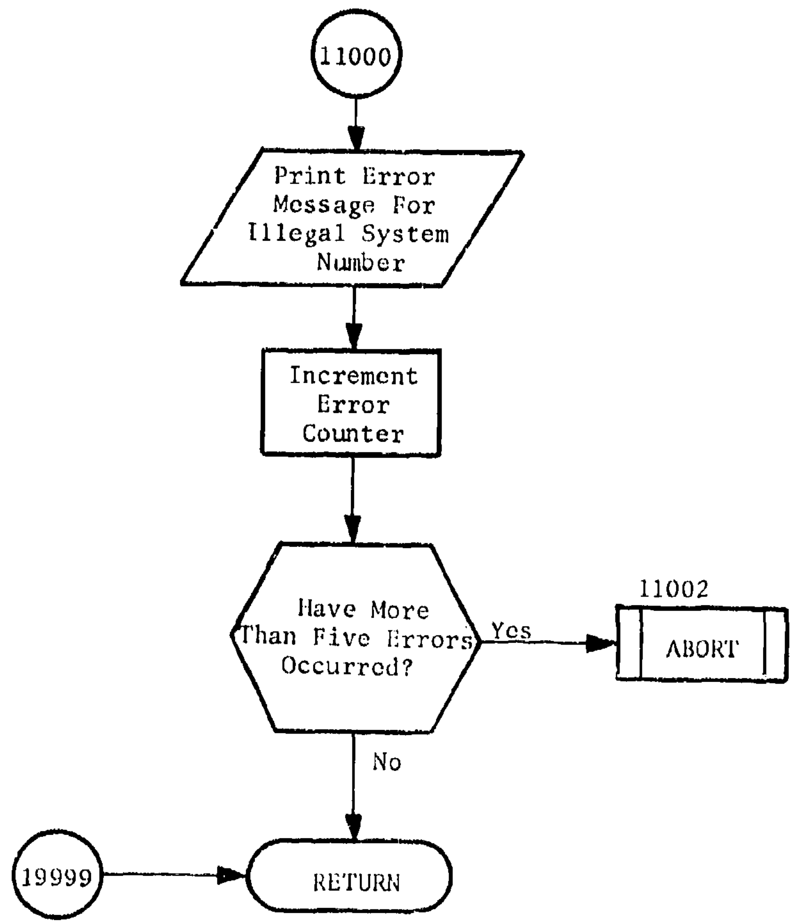


Fig. 94. (cont.)
Part IV: Error and Termination Blocks

SUBROUTINE FUELSAVE

PURPOSE:	This routine computes the fuel saved by the omission of each target in the footprint.
ENTRY POINTS:	FUELSAVE
FORMAL PARAMETERS:	None
COMMON BLOCKS:	FOOTIO, POTENT, WPNTGT, 1, DEBUG, PRINT
SUBROUTINES CALLED:	GOPRINT, TEST, MISSIT, HITIT
CALLED BY:	IMPROVE

Method

This subroutine takes the current hit list and calls subroutine TEST to compute the fuel remaining after completion of the weapon deliveries. Then, FUELSAVE modifies the hit list by the removal of the first target. TEST is called again and FUELSAVE calculates the difference in fuel used. The omitted target is returned to the hit list in the same position and the next target is omitted. This process of omission and testing is continued until the deletion of each target is tested.

The reciprocal of the fuel saved by the deletion of each target is stored in array COSTEFF in common /1/. The values in this array are used by IMPROVE to determine the order in which targets will be deleted during the improvement phase.

Subroutine FUELSAVE is illustrated in figure 95.

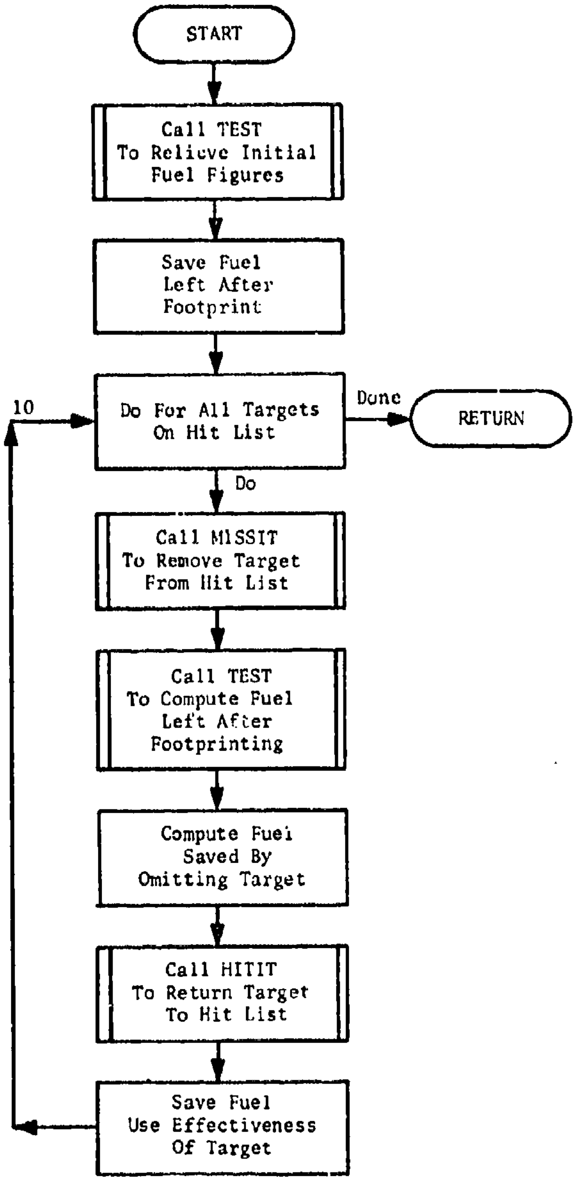


Fig. 95. Subroutine FUELSAVE

SUBROUTINE GOPRINT

PURPOSE:	This routine prints data at various stages of processing.
ENTRY POINTS:	GOPRINT
FORMAL PARAMETERS:	IX - Data block number
COMMON BLOCKS:	MASTER, WPNGRPX, STRKSUM, RAIDATA, 4, CONTROL, FOOTIO, FOOTSAVE, INDEX, LOADATA, PARAMETR, PERFORM, POTENT, RANGE, VALPARM, WPNTGT, 1, 2, 3, FLAG, DEBUG, PRINT
SUBROUTINES CALLED:	PRNTABLE, PRNTLOAD, ABORT
CALLED BY:	FOOTPRNT, ADDRV, ASSIGN, BOOSTIN, BOOSTOUT, CHKSEQ, EVAL, REVAL, FUELSAVE, HITIT, MISSIT, IMPROVE, INITASGN, INPOT, OUTPOT, NEWCOOR, OPTBOOST, REMOVE, TEST, TRANSFER

Method

This is the subroutine which does most of the printing in a run of the program. Each subroutine that desires a print sets the print request number, ICALL in common /PRINT/, and calls GOPRINT with a data block number as a formal parameter. These data blocks are sets of data of similar size and function that are printed to display processing results. Table 28 shows the data block definitions.

Upon entering GOPRINT, the error flag IMUST in common /PRINT/ is tested. If it is greater than zero, a print is produced regardless of the print flag setting. If there is no error (IMUST = 0) then GOPRINT tests the flag for the print request number* ICALL. If this flag was set true by subroutine PRINTSET, the print is produced. Otherwise control returns to the calling program with no further action by GOPRINT. If the flag is set true, then a computed GO TO statement directs processing to the print of the data block requested in the formal parameter.

Two special subroutines, PRNTABLE and PRNTLOAD, are used to print the footprint parameter tables and the booster loading data, respectively.

At the end of printing, GOPRINT determines if the print request number is the same as the dump number, IDUMP in common /FLAG/, set by

*Also called "print option number."

subroutine RDCARDF. If so, subroutine ABORT is called to produce a memory dump. Otherwise control is returned to the calling program.

Subroutine GOPRINT is illustrated in figure 96.

Table 28. Data Blocks Used in Print Requests
(Sheet 1 of 2)

NUMBER	COMMON	DESCRIPTION
1	WPNGRPX	Group data read from BASFILE
2	PARAMETR	MIRV system general parameters
3	------	Detailed footprint parameter tables
4	------	Detailed booster loading tables (not used)
5	STRKSUM	Gross strike data block
6	RAIDATA 4	Detailed strike data block
7	------	Range and launch azimuth of target set (includes index according to azimuth)
8	2	Status array, pointer arrays, booster loadings and pointers
9	RANGE	Uprange/downrange, crossrange/downrange ratios
10	DEBUG	List of chain of subroutine calls
11	POTENT 1	Potential target arrays; includes hit, miss, lost and free lists as well as age and value arrays
12	CONTROL	Control parameters for program
13	FOOTIO	Input/output data for footprint tester
14	PERFORM	Gross performance parameters
15	------	Same as number 11, /POTENT/ and /1/
16	WPNTGT	Indices for moving targets between hit and miss lists (includes target to booster assignment indices)
17	3 RANGE	Temporary storage of various parameters for all targets in potential target arrays. (includes common /RANGE/, number 9)

Table 28. (cont.)
(Sheet 2 of 2)

NUMBER	COMMON	DESCRIPTION
18	INDEX	Indices for adding and removing targets from potential target arrays; includes indices of targets added on "look ahead"
19	FOOTSAVE	Indices of targets in footprint saved during processing in OPTBOOST
20	------	Intertarget distance matrix for potential targets
21	------	Final plan
22	------	Distribution of re-entry vehicles to boosters
23	VALPARM	Program control parameters
24	LOADATA	Booster loading control information

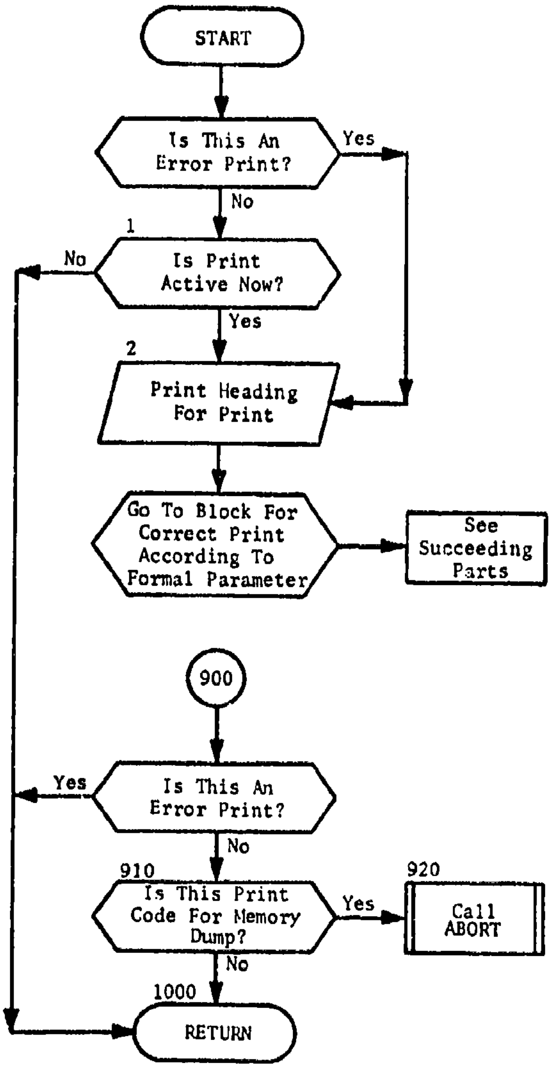


Fig. 96. Subroutine GOPRINT
Part I: Main Processing

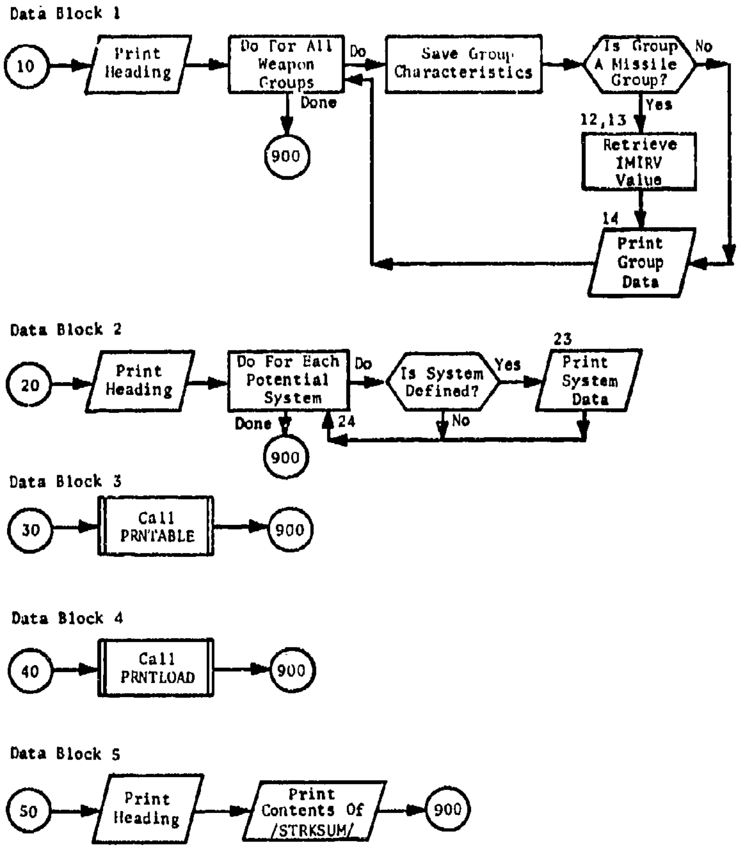


Fig. 96. (cont.)
Part II: Data Blocks 1,2,3,4,5

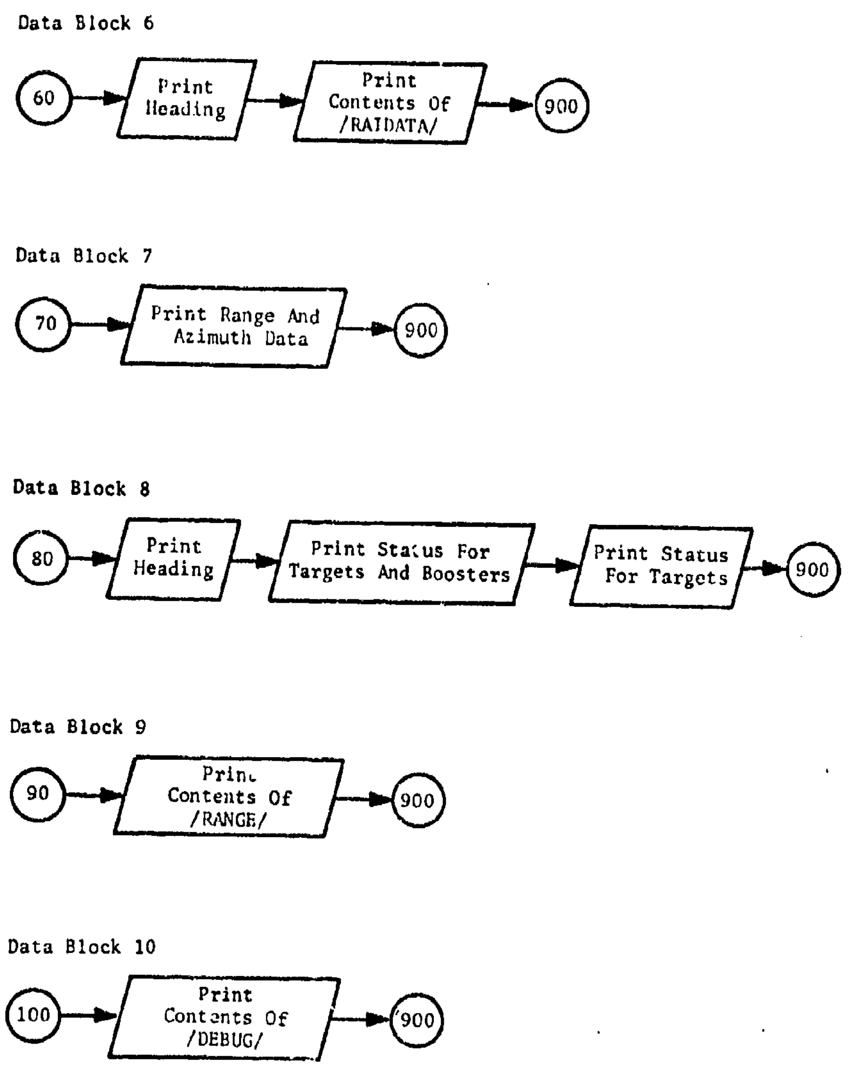


Fig. 96. (cont.)
Part III: Data Blocks 6,7,8,9,10

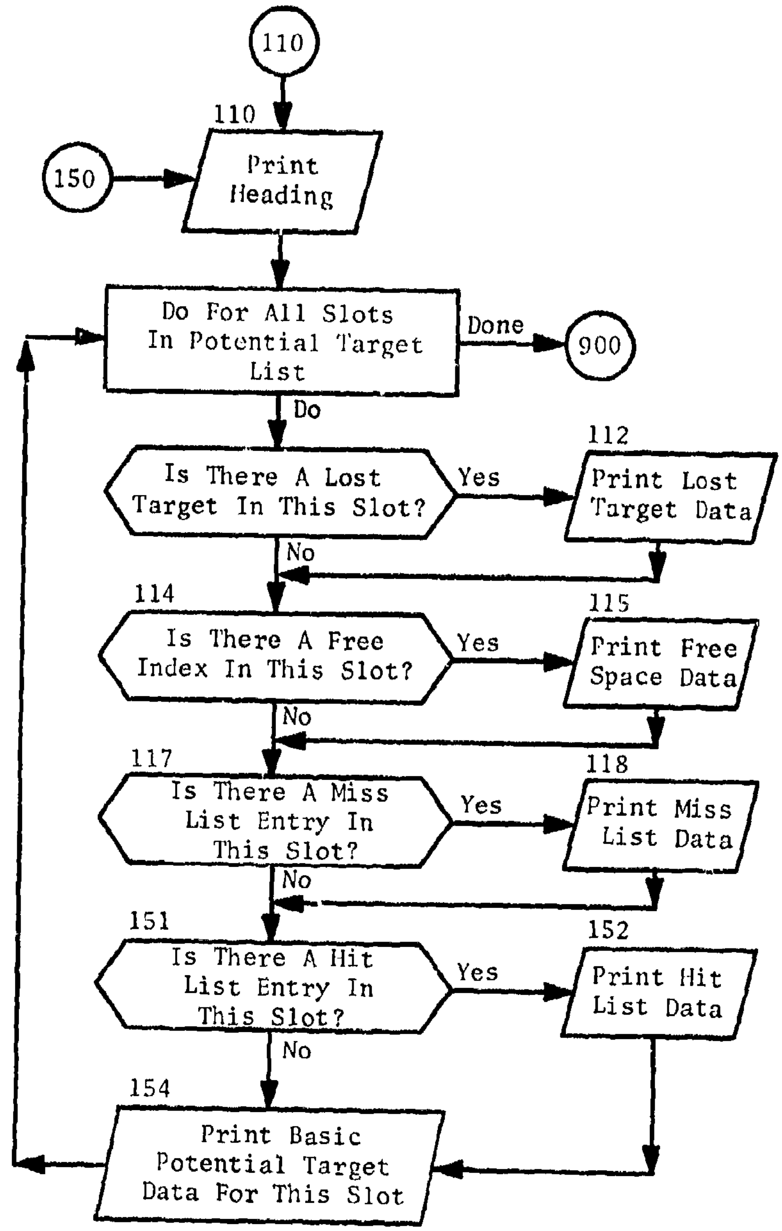


Fig. 96. (cont.)
Part IV: Data Blocks 11,15

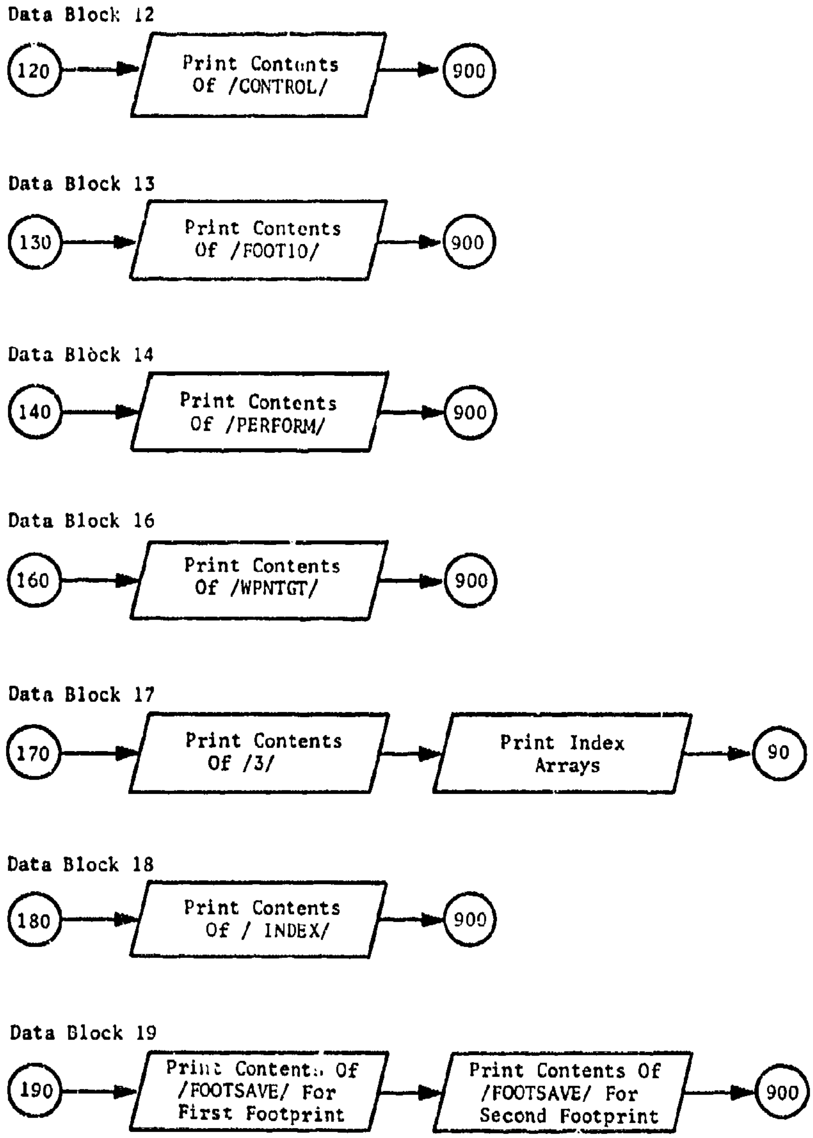


Fig. 96. (cont.)
Part V: Data Blocks 12,13,14,16,17,18,19

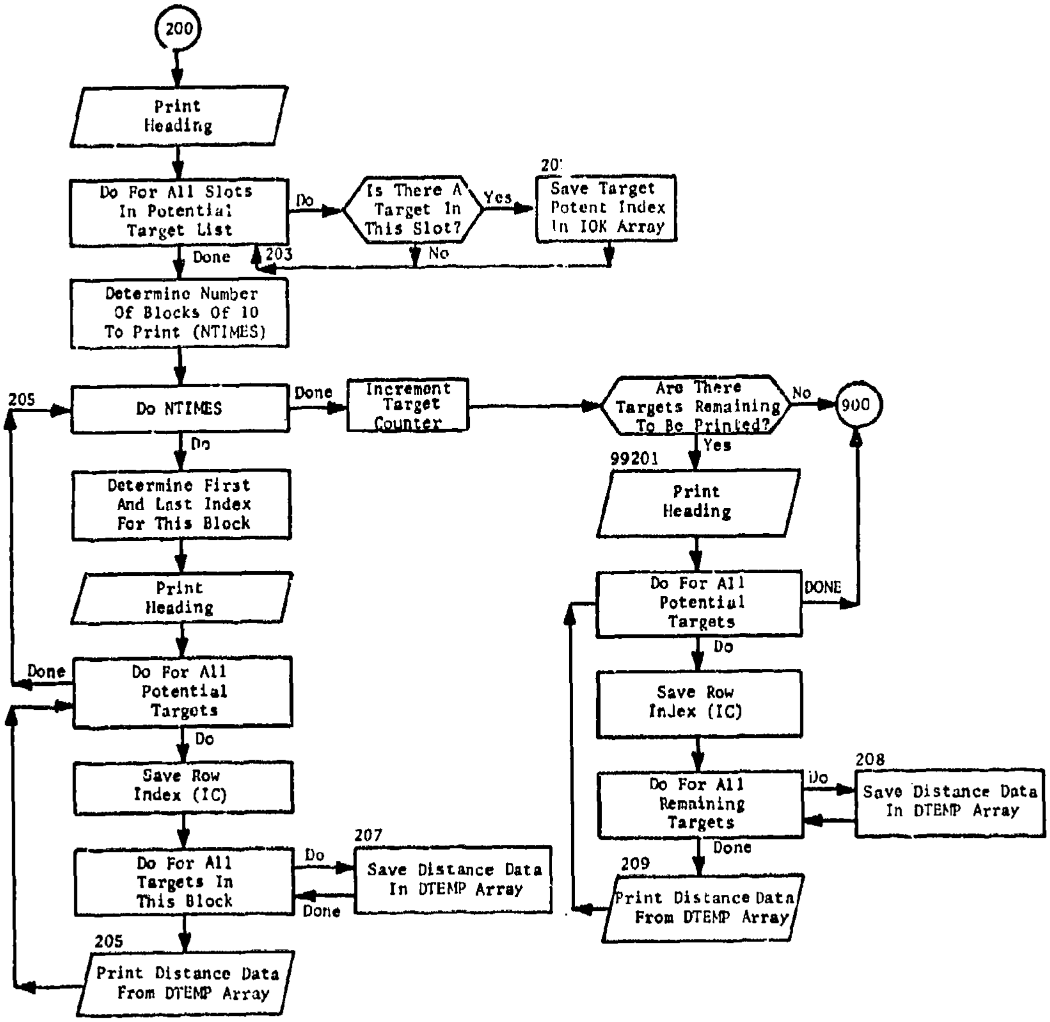


Fig. 96. (cont.)
Part VI: Data Block 20

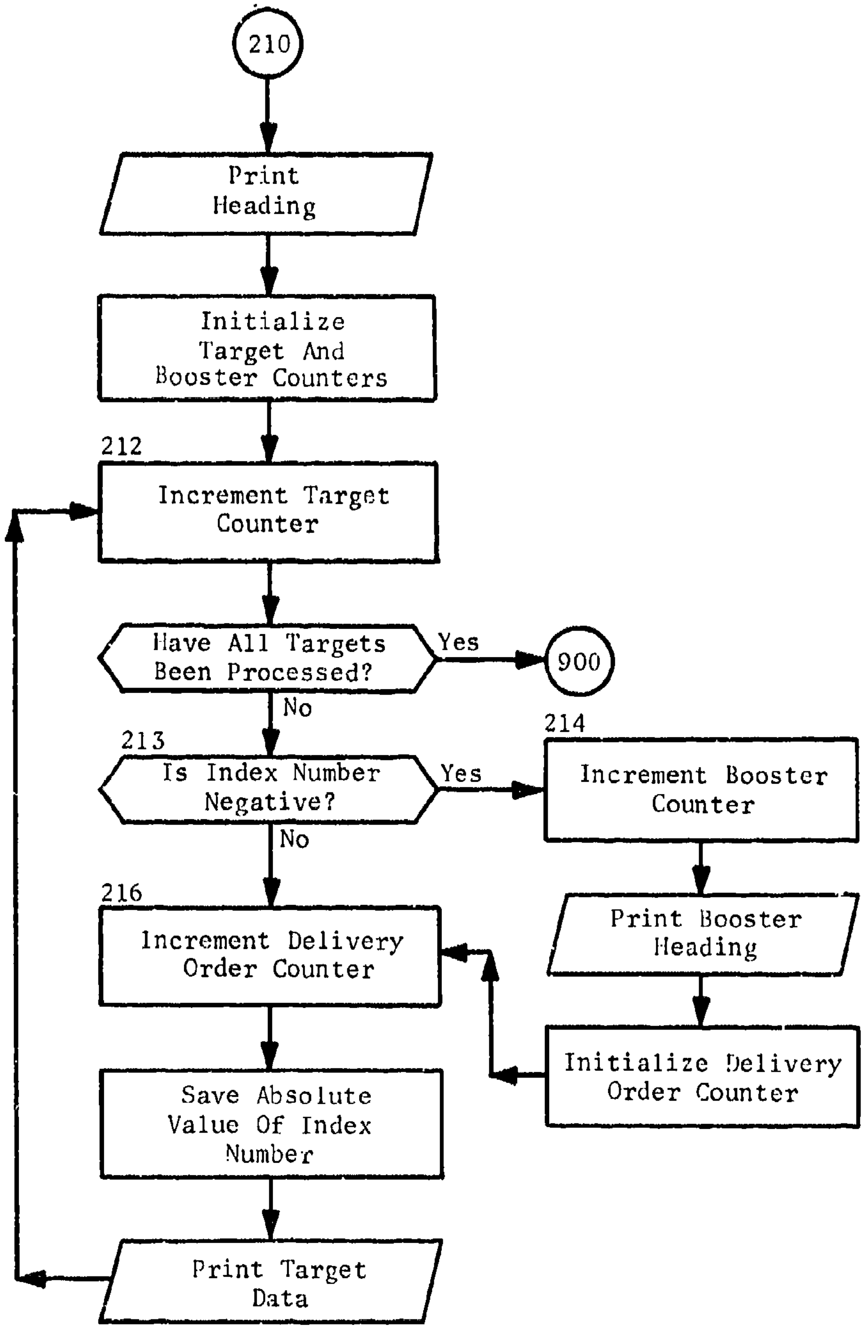


Fig. 96. (cont.)
Part VII: Data Block 21

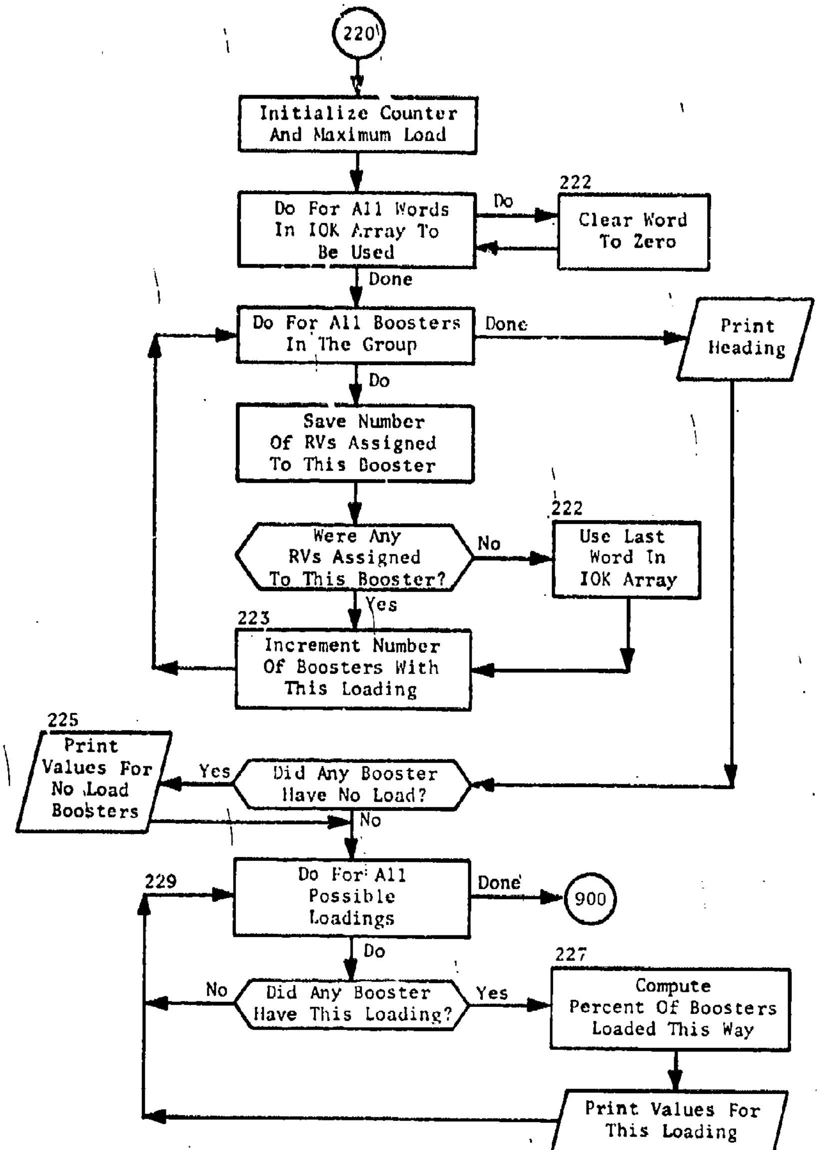
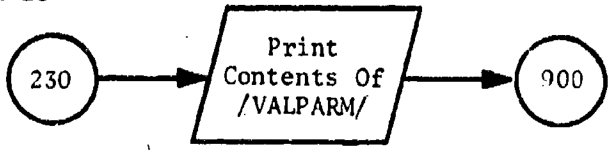


Fig. 96. (cont.)
Part VIII: Data Block 22

Data Block 23



Data Block 24

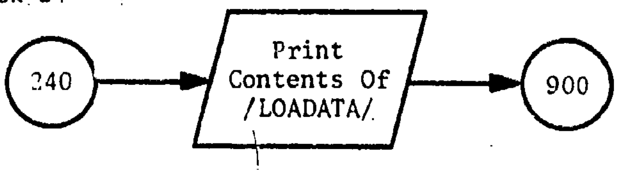


Fig. 96. (cont.)
Part IX: Data Blocks 23,24

SUBROUTINE HITIT

PURPOSE:	This routine enters and removes targets from hit and miss lists.
ENTRY POINTS:	HITIT, MISSIT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	POTENT, WPNTGT, 1, DEBUG, PRINT
SUBROUTINES CALLED:	GOPRINT
CALLED BY:	CHKSEQ, FUELSAVE, IMPROVE, OPTBOOST

Method

This routine performs the list manipulation operations required to move a target between the hit and miss lists.

The first entry, HITIT, is used to move a target from the miss list to the hit list. The required data for the move are contained in common /WPNTGT/ as follows:

IPOTGT - Potential target index of target to be moved

JAFT - The position in the hit list after which the target is to be added.

If the added target is to be the first target in the hit list, JAFT is set to 0. This subroutine resets all the indices and pointers to move the target to the hit list.

The second entry, MISSIT, is used to move a target from the hit list to the miss list. The target whose potential target index is IPOTGT is removed from the hit list and placed at the end of the miss list. The entries which followed the removed target on the hit list are moved up on that list.

Both entries use the INVERSE array to determine the position of the target on the respective lists. If INVERSE (IPOTGT) is greater than zero, the value is the position of target with potential index IPOTGT in the hit list. If the value is negative, it is the position in the miss list.

Subroutine HITIT is illustrated in figure 97.

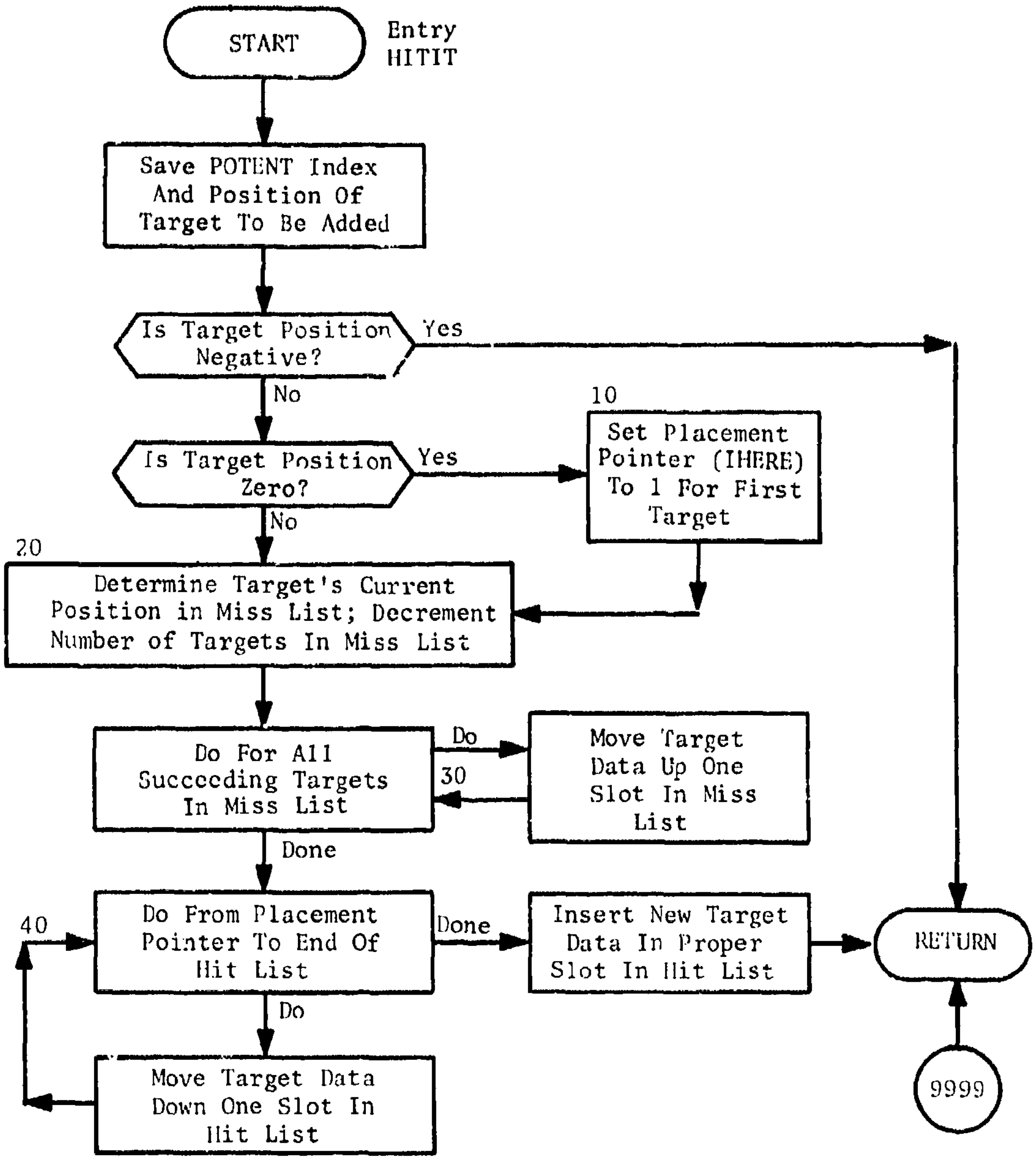


Fig. 97. Subroutine HITIT
Part I: Entry HITIT

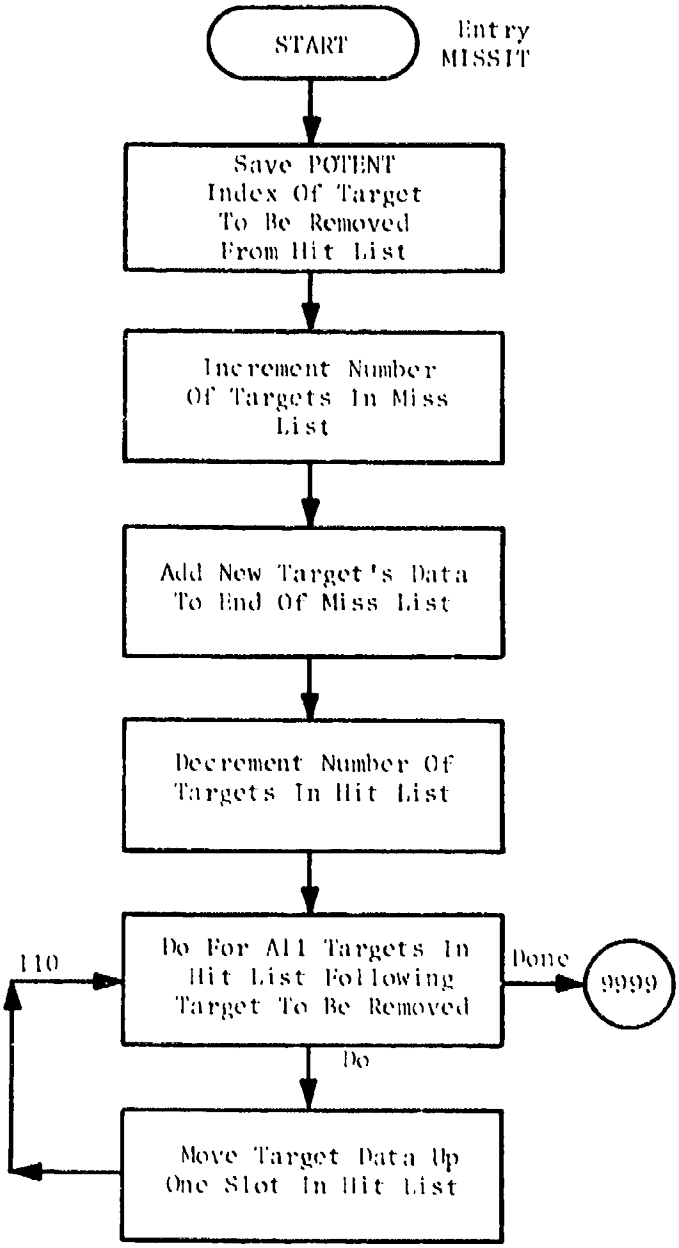


Fig. 97. (cont.)
Part II: Entry MISSIT

462 546 O - 72 - 8

SUBROUTINE IMPROVE

PURPOSE:	This routine improves the footprint by removing the target that uses the most additional fuel and adds other targets if possible.
ENTRY POINTS:	IMPROVE
FORMAL PARAMETERS:	None
COMMON BLOCKS:	CONTROL, FOOTIO, PARAMETR, POTENT, WPNTGT, 1, 2, 3, DEBUG, PRINT
SUBROUTINES CALLED:	GOPRINT, HITIT, MISSIT, TEST, FUELSAVE, EVAL, REVAL,
CALLED BY:	OPTBOOST

Method

This subroutine attempts to improve the best footprint provided by subroutine OPTBOOST. It investigates minor modifications to the footprint to determine if more targets (or targets of greater value) can be added to the footprint. If there is only one target in the footprint, IMPROVE assigns to the footprint the most valuable target that is feasible. (See part II of figure 98.)

If the footprint input from OPTBOOST contains more than one target, IMPROVE determines the best target to remove temporarily from the footprint. (See part I of figure 98.) Subroutine FUELSAVE is called to determine the marginal fuel use of each target in the footprint. The target with the greatest fuel use is removed from the hit list.

The processing shown in part III of figure 98 determines the best target (or targets) to add to replace the temporarily omitted target. Subroutine EVAL (or REVAL) is called to determine the worth of adding each target to the footprint. IMPROVE attempts to add each unassigned target (excluding the temporarily removed target) to the footprint in order of decreasing worth. This process continues until either all unassigned targets have been investigated or until the booster has been assigned its maximum load. The routine then determines if more than one target has replaced the omitted one. If so, the cycle repeats with a new determination of the best target to remove temporarily. (Note that if the target selected for removal was the last target added in the improvement phase, the subroutine will return control to the calling program without removing the target.)

The improvement phase ends when IMPROVE cannot add more than one target after removing the target with maximal fuel use (see part IV of figure 98). If no target could be added, IMPROVE returns the omitted target to the footprint. If only one target was added, IMPROVE determines which target, the removed or the added, is more valuable. The more valuable target is then used in the footprint to the exclusion of the other.

If IMPROVE has returned the omitted target to the footprint, its feasibility is checked once again. If the footprint is not feasible the targets have been shuffled during processing. The subroutine will print an error message to this effect and return control in the normal fashion.

Note that IMPROVE never attempts to return the omitted target during the improvement phase shown in part III of figure 98. This procedure assumes that OPTBOOST has previously investigated all the possible footprints containing that target and it is more efficient for IMPROVE to ignore those possibilities.

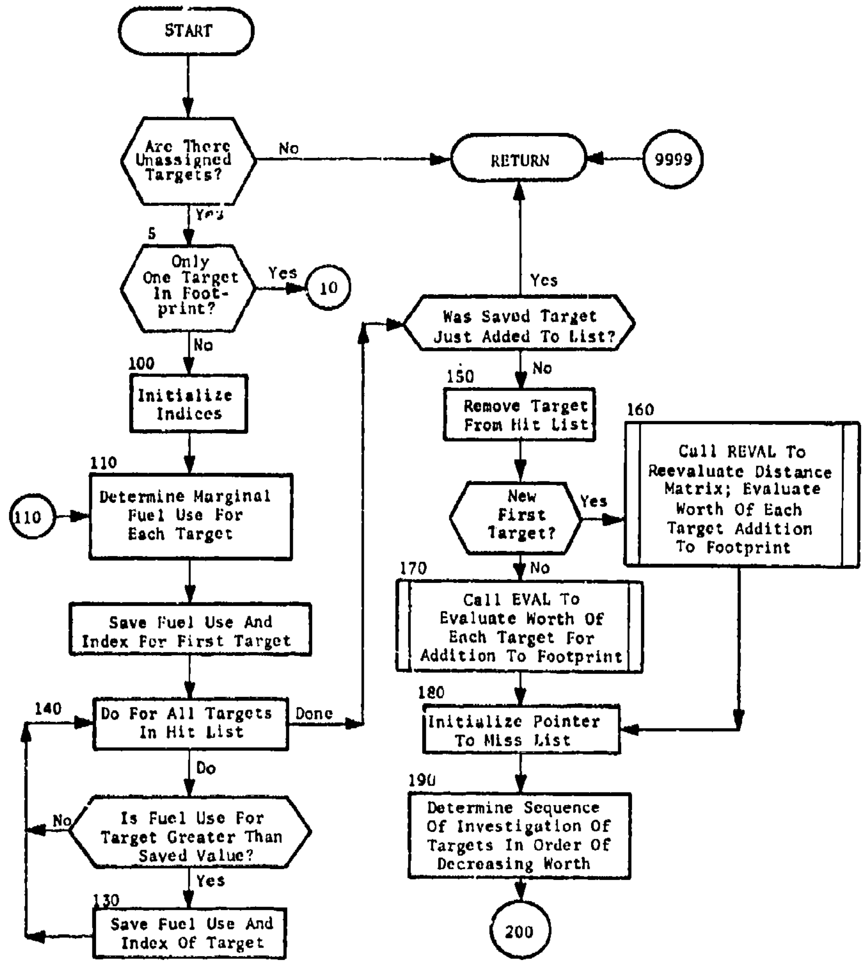


Fig. 98. Subroutine IMPROVE
Part I: Removal of Target with
Maximum Fuel Use

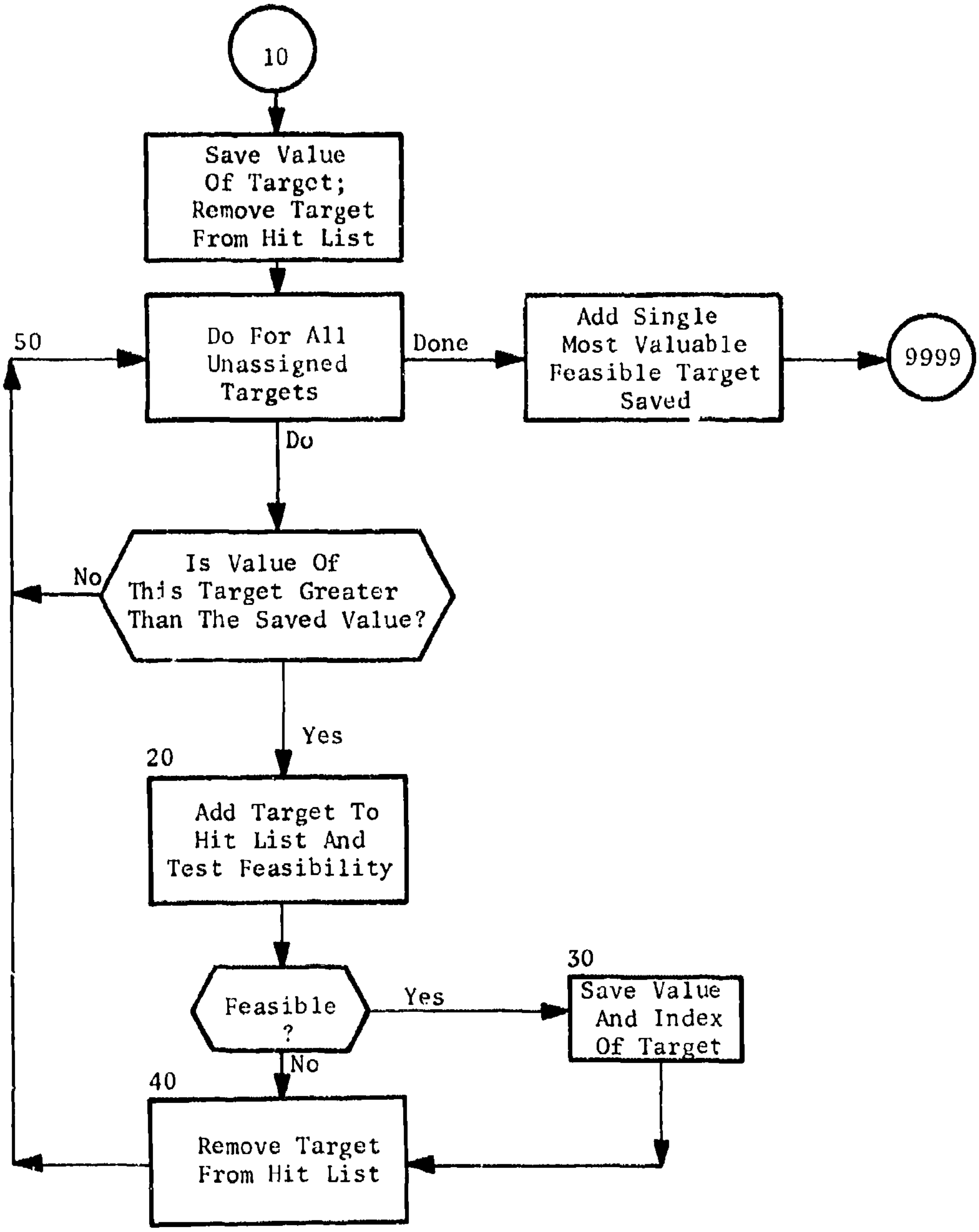


Fig. 98. (cont.)
Part II: Single Target Improvement

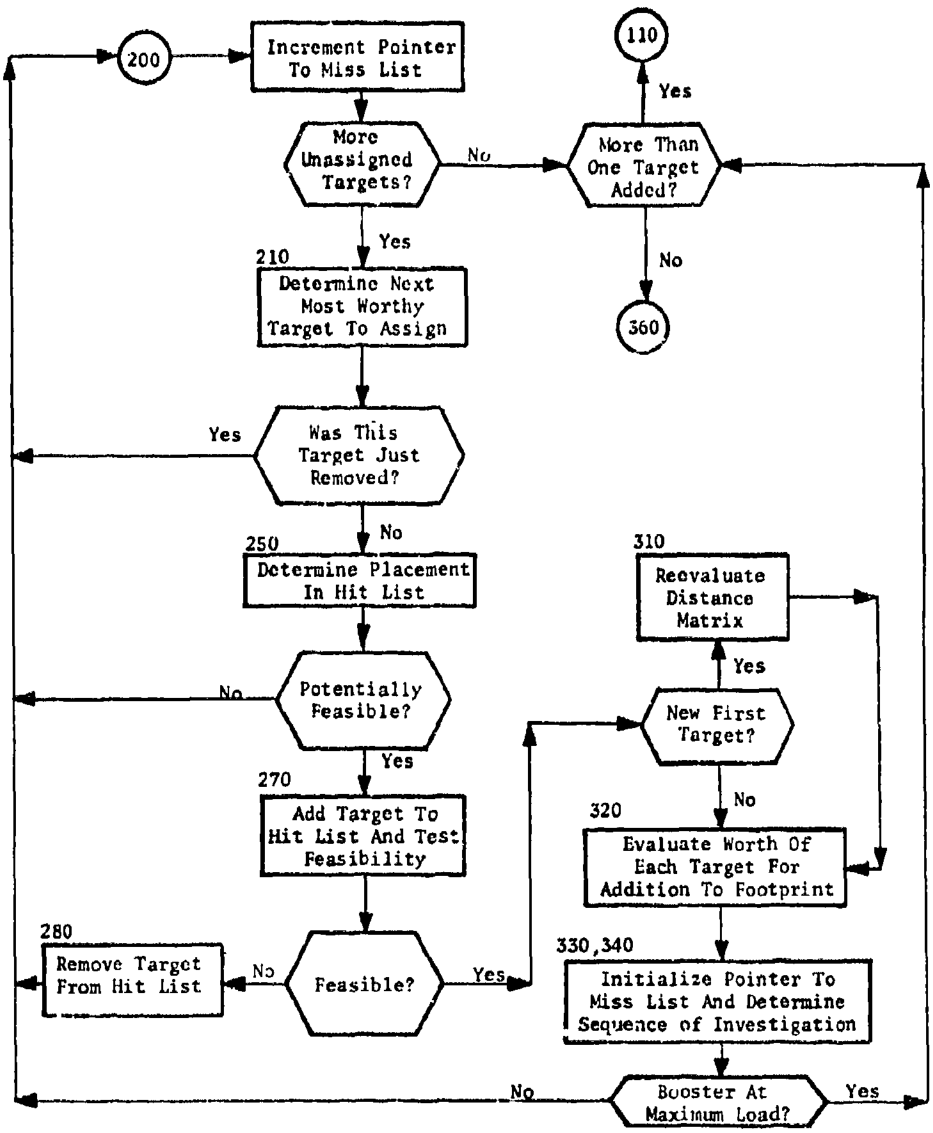


Fig. 98. (cont.)
Part III: Determination of Best Replacement Targets

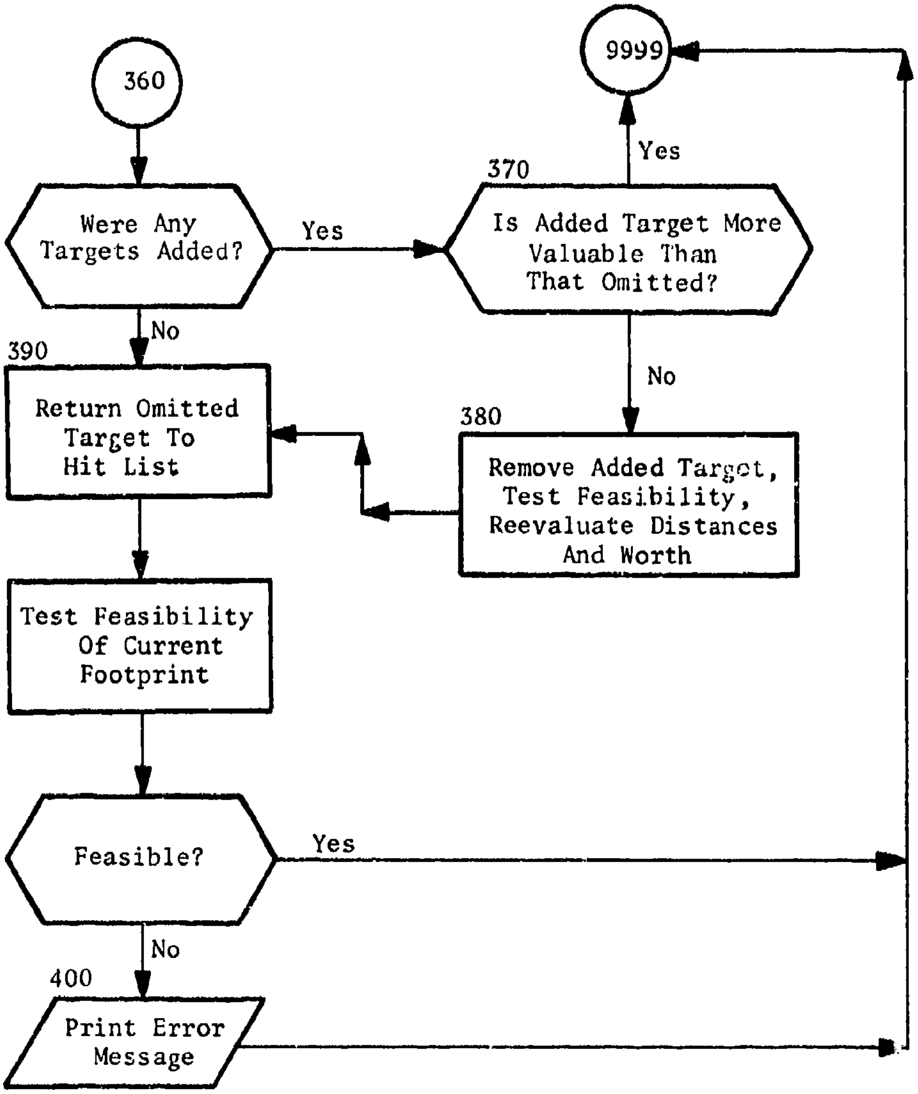


Fig. 98. (cont.)
Part IV: Check Improvement

SUBROUTINE INITASGN

PURPOSE: This routine performs the initial assignment of targets to boosters.

ENTRY POINTS: INITASGN

FORMAL PARAMETERS: None

COMMON BLOCKS: RAIDATA, 4, CONTROL, PERFORM, POTENT, 2, DEBUG, PRINT

SUBROUTINES CALLED: GOPRINT

CALLED BY: FOOTPRNT

Method

This subroutine performs the list manipulation required to assign all the targets to the boosters in an initial assignment. This assignment, whose feasibility is never tested, serves as a starting point for later processing. At the time INITASGN is called the targets are ordered by increasing values of launch azimuth. The data used by INITASGN to assign the targets are contained in common /CONTROL/ and are as follows:

NV - Number of boosters

NARV - Ratio of number of targets input (NT in /RAIDATA/) to the number of boosters (truncated to largest integer less than or equal to value.)

NEXTRA- Number of boosters which must carry one more RV than the average number (NEXTRA = NT - (NARV*NV)).

After INITASGN assigns the targets, NEXTRA boosters will be assigned NARV + 1 targets and the remaining boosters (NV-NEXTRA) will be assigned NARV targets. (The first NEXTRA boosters in order of increasing azimuth will each be assigned the extra target.)

The assignment method is straightforward manipulation of the RAIDATA list pointers, IFOR, IBACK, IBOOST, and NTB. In addition the number of targets currently assigned, NASGN, and the sum of the relative values (RVAL) of the targets currently assigned, VALASGN, are incremented as each target is added to the initial assignment. The targets are assigned in serial order to each booster.

Subroutine INITASGN is illustrated in figure 99.

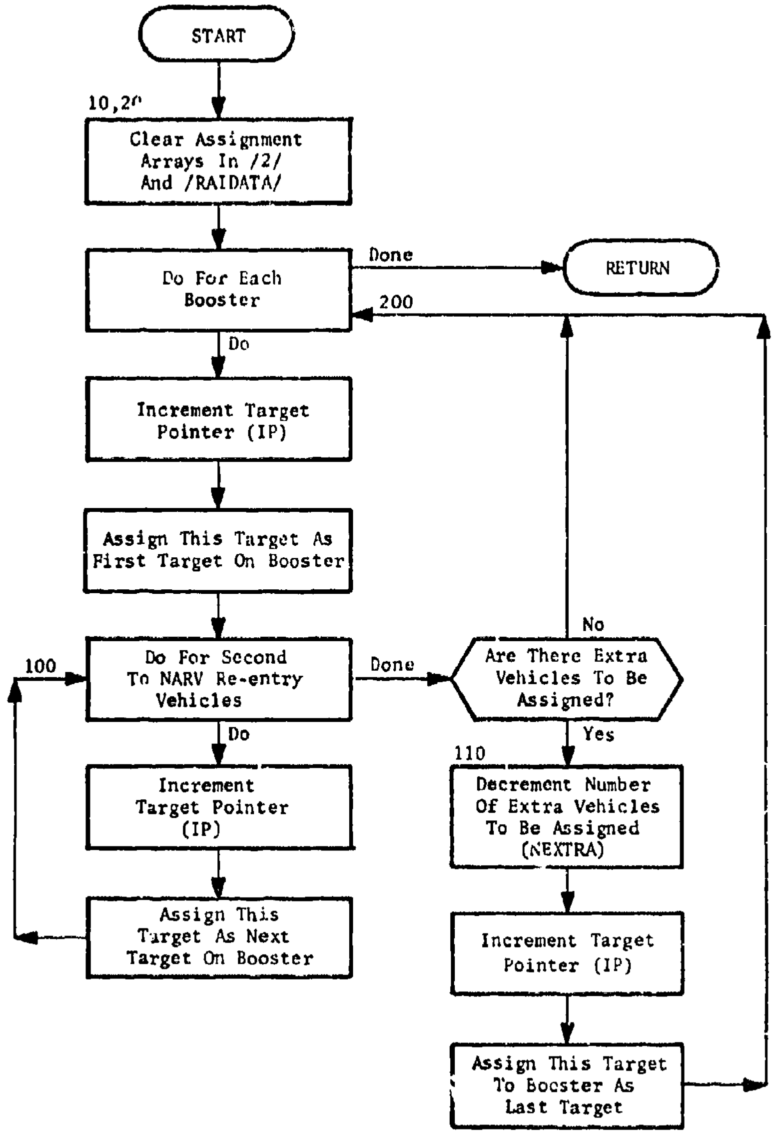


Fig. 99. Subroutine INITASGN

SUBROUTINE INPOT

PURPOSE: This routine adds and deletes targets from the potential target arrays.

ENTRY POINTS: INPOT, OUTPOT

FORMAL PARAMETERS: None

COMMON BLOCKS: RAIDATA, 4, CONTROL, INDEX, POTENT, WPNTGT, 1, 2, 3, DEBUG, PRINT

SUBROUTINES CALLED: GOPRINT, REMOVE

CALLED BY: BOOSTIN, OPTBOOST

Method

Entry INPOT removes a target from the RAIDATA lists and enters it in the potential target arrays. If the target is currently assigned to a booster, subroutine REMOVE is called to remove the assignment. Entry OUTPOT removes a target from the potential target arrays and returns it to the RAIDATA list in an unassigned state. (Subroutine BOOSTOUT is used to remove targets that are assigned to a booster.) The data which controls this subroutine are contained in common /INDEX/ as follows:

JINR - RAIDATA index of target to be entered into potential target arrays

JOUTP - Potential target index of target to be removed from potential target arrays.

During processing, the following indices in common /INDEX/ are also defined:

JINP - Potential target index for target entering potential arrays (=IFREE (NFREE))

JOUTR - RAIDATA index of target to be removed (=IPOT (JOUTP)).

To save time in processing by reducing the number of references to variables in common storage, the following substitutions are made for these indices:

JR = JINR (in INPOT)

JR = JOUTR (in OUTPOT)

JP = JINP (in INPOT)

JP = JOUTP (in OUTPOT)

The processing of this subroutine is very straightforward, as displayed in figure 100.

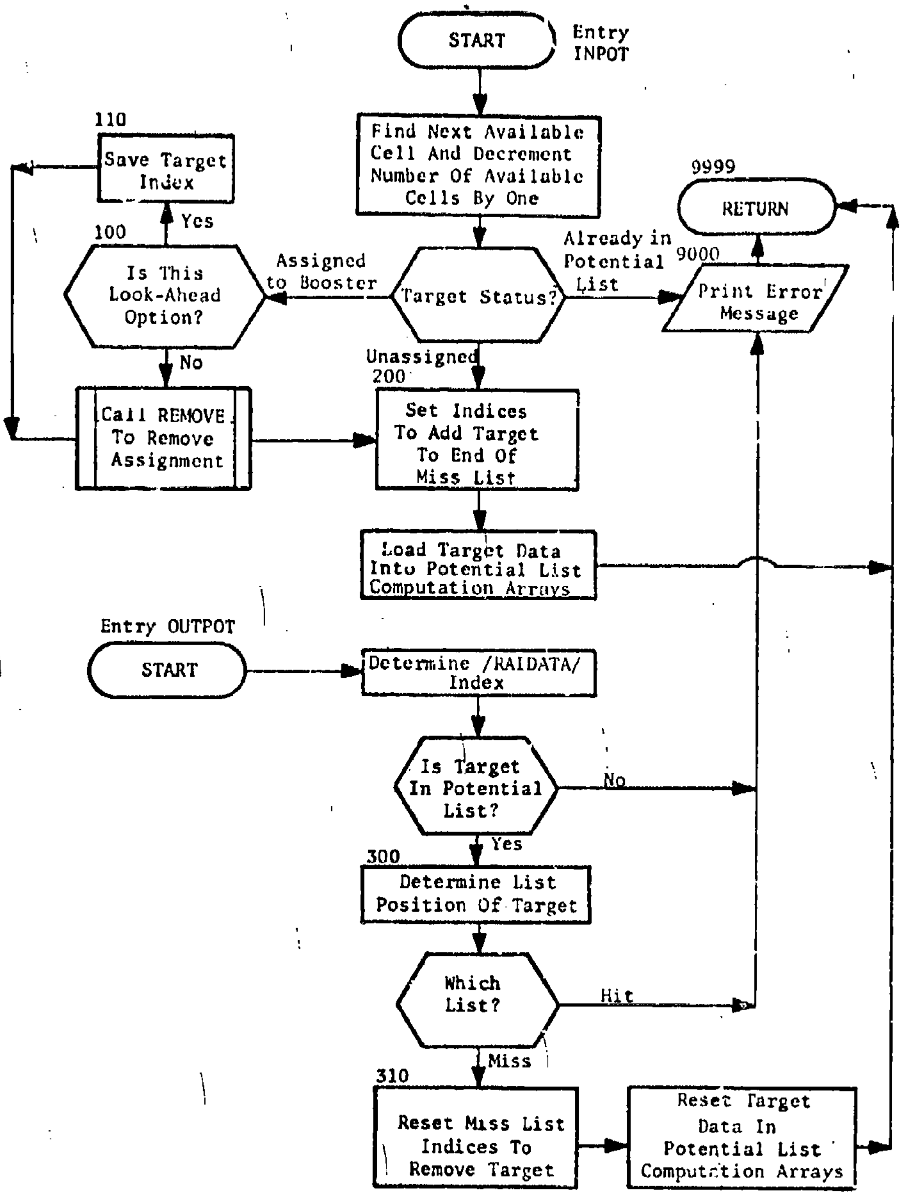


Fig. 100. Subroutine INPOT

SUBROUTINE LOADREAD

PURPOSE: This routine reads and prints booster loading data.

ENTRY POINTS: LOADREAD, PRNTLOAD

FORMAL PARAMETERS: None

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: GOPRINT (PRNTLOAD), RDCARDF (LOADREAD)

Method

This subroutine is currently a dummy routine. Its purpose will be to read data on variable booster loadings within a group and also to print that data. The dummy routine merely reserves the entry points for later expansion of the program to include a variable booster loading option. (LOADOPT = *VARY* = option 2)

SUBROUTINE LREORDER

PURPOSE:	This routine reorders the elements of a packed logical array.
ENTRY POINTS:	LREORDER
FORMAL PARAMETERS:	ISEQ - A sequence array to control reordering N - Number of elements to be reordered LOGAR - A logical array to be reordered
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	FOOTPRNT, NEWCOOR

Method

This subroutine uses the same method as the utility subroutine REORDER. This extra routine is required for logical arrays on the CDC 3800 computer system. Logical arrays are packed 32 elements to one computer word on this system and the word manipulation code of subroutine REORDER would not correctly reorder a packed array.

The ISEQ array is a sequence key array of the type produced by subroutine ORDER. It contains the indices of the array LOGAR in the order in which they are placed. That is, ISEQ(1) contains the index of the element in LOGAR that is to be placed first, ISEQ(2) contains the index of the element that is to be placed second, and so on. The parameter N determines the number of elements to be reordered. At the end of the subroutine, the elements of LOGAR have been reordered.

LREORDER stores one element from LOGAR in a temporary location. It then reads from ISEQ the element which should go in that position (which may now be considered empty) and moves it, filling the position and creating a new empty cell. Each new empty cell is filled with its proper contents as soon as the original contents have been removed. When the element in temporary storage is required, LREORDER finds another element which is not already in proper sequence, puts it into temporary storage, and continues as before. This process continues until no elements are out of sequence. The contents of ISEQ are returned to the calling program unchanged so that the sequence key can be used again.

Subroutine LREORDER is illustrated in figure 101.

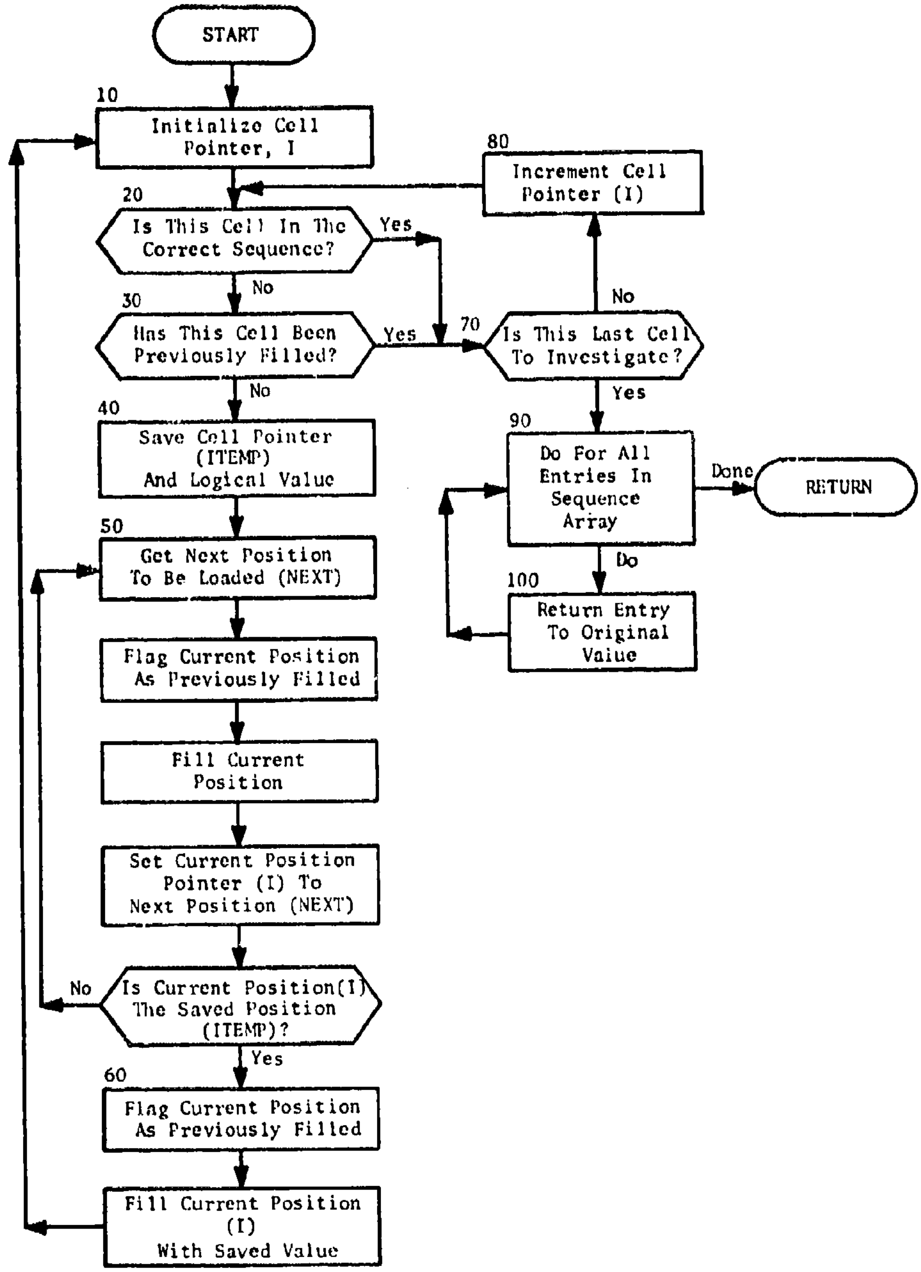


Fig. 101. Subroutine LREORDER

SUBROUTINE NEWCOOR

PURPOSE:	This routine converts target coordinates from latitude and longitude to range and azimuth from weapon group centroid.
ENTRY POINTS:	NEWCOOR
FORMAL PARAMETERS:	IG - Group Number
COMMON BLOCKS:	FILES, WPNGRPX, RAIDATA, 4, EARTH, 2, DEBUG, PRINT, TSCRATCH, Filehandler Blocks (ITP, MYIDENT, TWORD, NOPRINT, FILABEL)
SUBROUTINES CALLED:	GOPRINT, ORDER, REORDER, LREORDER, WRARRAY, DISTF
CALLED BY:	FOOTPRNT

Method

This routine converts the target coordinates for use by the footprint generation subroutines. It is called once for each group.

For each target, NEWCOOR adds the target point offsets (DLAT, DLONG) to the target coordinate (TGTLAT, TGTLONG). The range from the weapon group centroid to the target point (RANGE) is then computed by a call on the distance function, DISTF. The position of the group centroid is given by the variables WLAT and WLONG in common /WPNGRPX/. The formal parameter IG is used to retrieve the correct position.

The calculation of the launch azimuth uses spherical trigonometry. First all the latitudes are converted to radians by the factor DEGTORAD in common /EARTH/. The range is normalized by dividing by the radius of the earth (RADIUS in common /EARTH/).

The computation of the launch range is performed as follows. Define a spherical triangle with vertices at the group centroid, North Pole and target. (See figure 102.) Let angle A (the launch azimuth) be the angle between the line connecting the centroid and the North Pole and the line connecting the centroid and the target. Measure the distances between the points in terms of the number of radians subtended by the connecting lines. If distance a is the distance between target and North Pole, b is the distance between centroid and target, and c is the distance between centroid and North Pole, then:

a = π/2 - (TGTLAT*DEGTORAD)

b = RANGE/RADIUS

c = π/2 - (WLAT*DEGTORAD)

Using the law of cosines for spherical triangles, then:

$$\cos A = \frac{\cos a - (\cos b \cdot \cos c)}{\sin b \cdot \sin c}$$

The difference between the target longitude and the centroid longitude is then used to determine the sign of the launch azimuth.

After all launch azimuths have been computed, the target data are reordered according to increasing value of launch azimuth. The sequence array used for this reordering is written on the assignment data scratch file, ISCR, for later use.

Subroutine NEWCOOR is illustrated in figure 103.

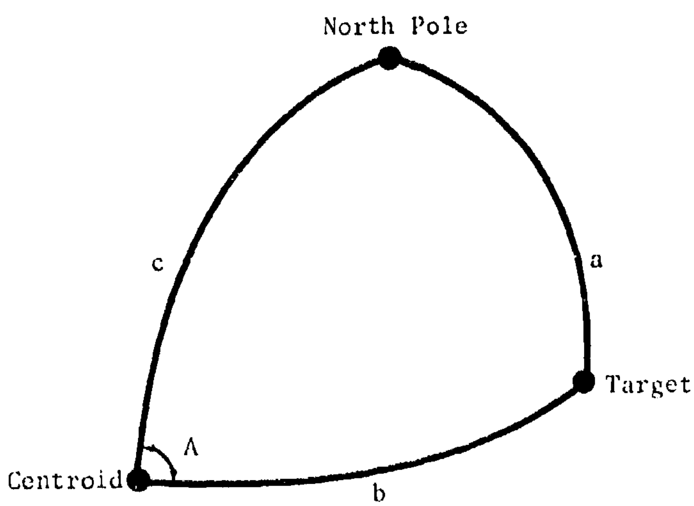


Fig. 102. Calculation of Launch Azimuth

462-546 O - 72 - 9

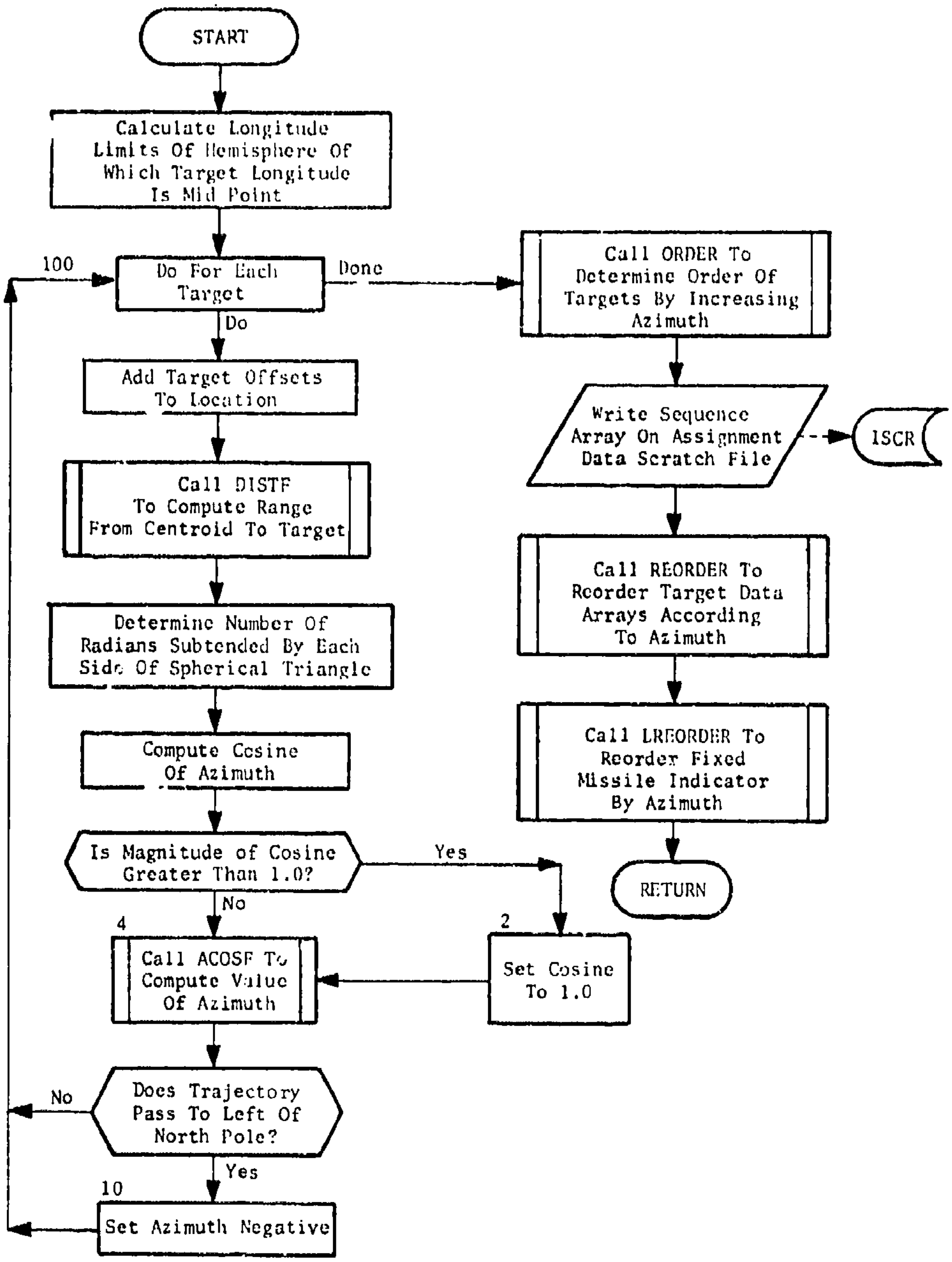


Fig. 103. Subroutine NEWCOOR

SUBROUTINE OPTBOOST

PURPOSE:	This routine generates the basic footprint(s) for each booster.
ENTRY POINTS:	OPTBOOST
FORMAL PARAMETERS:	None
COMMON BLOCKS:	STRKSUM, RAIDATA, 4, CONTROL, FOOTIO, FOOTSAVE, INDEX, LOADATA, PARAMETR, POTENT, WPNTGT, 1, 2, 3, DEBUG, PRINT
SUBROUTINES CALLED:	GOPRINT, HITIT, TEST, MISSIT, EVAL, REVAL, IMPROVE, ORDER, INPOT, ADDRV
CALLED BY:	FOOTPRNT

Method

This subroutine generates the footprint assignment for each booster. The routine creates the footprint by an incremental method. That is, targets are added to the footprint until either the booster is carrying a full load or no more targets can be added without violating the fuel constraints. Subroutine IMPROVE is then called to investigate possible improvements.

The primary user-input parameters which affect processing in OPTBOOST are MAXFOOT and FRACLOOK. The former controls the degree of effort to be expended by OPTBOOST in footprint generation. The subroutine will generate up to two separate footprints from the potential target list. The better footprint will be selected for improvement and later processing. The absolute value of MAXFOOT determines the number of footprints generated. (MAXFOOT must be -2, -1, +1, or +2.) The smaller absolute value saves processing time but limits the number of distinct footprints generated. The sign of MAXFOOT determines the use of the "look-ahead" feature. If MAXFOOT is negative, no targets are added to the potential list after subroutine BOOSTIN. If the variable is positive, OPTBOOST retrieves targets from the RAIDATA list and has them entered into the potential list. The targets to be retrieved are those assigned to the next booster to be processed. This look-ahead feature allows the consideration of targets with similar launch azimuths which are assigned to a later booster. The number of targets entered is controlled by FRACLOOK. This variable is the fraction of the next booster assignment which is entered.

The subroutine can be divided into five parts as shown in figure 104.

Part I: Determination of Best Initial Point

This part determines the best starting point for each footprint. It begins by ordering the target potential indices by decreasing value of worth (VAL). An index, JONE, is kept to point to the target currently under consideration as the initial point. The targets are considered in order of decreasing worth. The routine evaluates the feasibility of each target until a feasible target is obtained. If no feasible target can be found (and the first footprint for the booster is being processed) the routine exits after printing an error message. When the best initial point is found, the distance and value matrices are recomputed by subroutine REVAL. OPTBOOST then redetermines the order of targets by decreasing worth.

Part II: Addition of Targets

This part is the heart of subroutine OPTBOOST. It attempts to add as many targets as possible within the fuel constraints. The processing operates as follows:

1. Retrieve index to next target

2. If all unassigned targets have been investigated, go to improvement section

3. Test feasibility of adding this target

4. If infeasible, return to step 1

5. Reevaluate matrices and reorder unassigned target indices according to decreasing worth

6. Return to step 1 unless booster is carrying maximum load.

The improvement phase follows the addition phase. Subroutine IMPROVE is called to determine the benefits of minor changes in the footprint.

Part III: Selection of Best Footprint

According to the absolute value of MAXFOOT, either one or two distinct footprints are generated from the potential list. The two footprints are distinct in that no target is assigned to both footprints. The processing in this part involves saving and retrieving the indices of the targets in the footprints. A second footprint is considered only if the number of unassigned targets in the potential list is not less than the number of targets in the first footprint. Only then can the second footprint be an improvement over the first.

After both footprints have been generated, the routine determines the better one. This decision is made by selecting the footprint with the greater number of targets. If both footprints have the same number of targets, the routine selects the one with a higher sum of target values (RVAL).

During this phase, the target indices are stored in temporary storage. Array IFOTSAVE in common /FOOTSAVE/ is used for indices for the first footprint. Array IF2SAVE in the same block is used for the second footprint.

After selection of the best footprint the routine checks the sign of MAXFOOT. If positive, the look-ahead feature is implemented.

Part IV: Look-Ahead to Next Booster Assignment

This feature allows OPTBOOST to consider targets assigned to the next booster to be processed. On the first pass, these are the targets with the launch azimuths next larger than those in the current footprint. In the second pass, these are the targets with next smaller launch azimuth.

In order that targets added on look-ahead not be immediately removed by subroutine BOOSTIN before processing the next booster, the save indicator, JSAVOPT, is set to 1 before the targets are entered into the potential list. This indicator directs subroutine INPOT to place these target indices into the JSAVE array in common /INDEX/. BOOSTIN will not remove targets whose indices are in the JSAVE array.

The number of targets added is determined by the user input, FRACLOOK, which is the fraction of the next booster to be added. In any case, the number added cannot exceed either the total assignment to the next booster, or the number of available cells in the potential target arrays.

After the targets are entered, the distance and value matrices are completely recomputed so that all data for the new targets are added. The target indices are reordered according to decreasing worth, and control returns to Part II, Addition of Targets.

Part V: Termination

There are two parts to this section. The first retests footprint feasibility. Processing by IMPROVE may cause the target sequence to be jumbled. If this perturbation affects feasibility, an error message is printed at this point, but processing continues. The second part tests for fulfillment of the minimum load constraint. If the free load option (LOADOPT = 0) is not in effect and the assignment has not met the minimum load, subroutine ADDRV is called to increase the number of re-entry vehicles assigned to the booster.

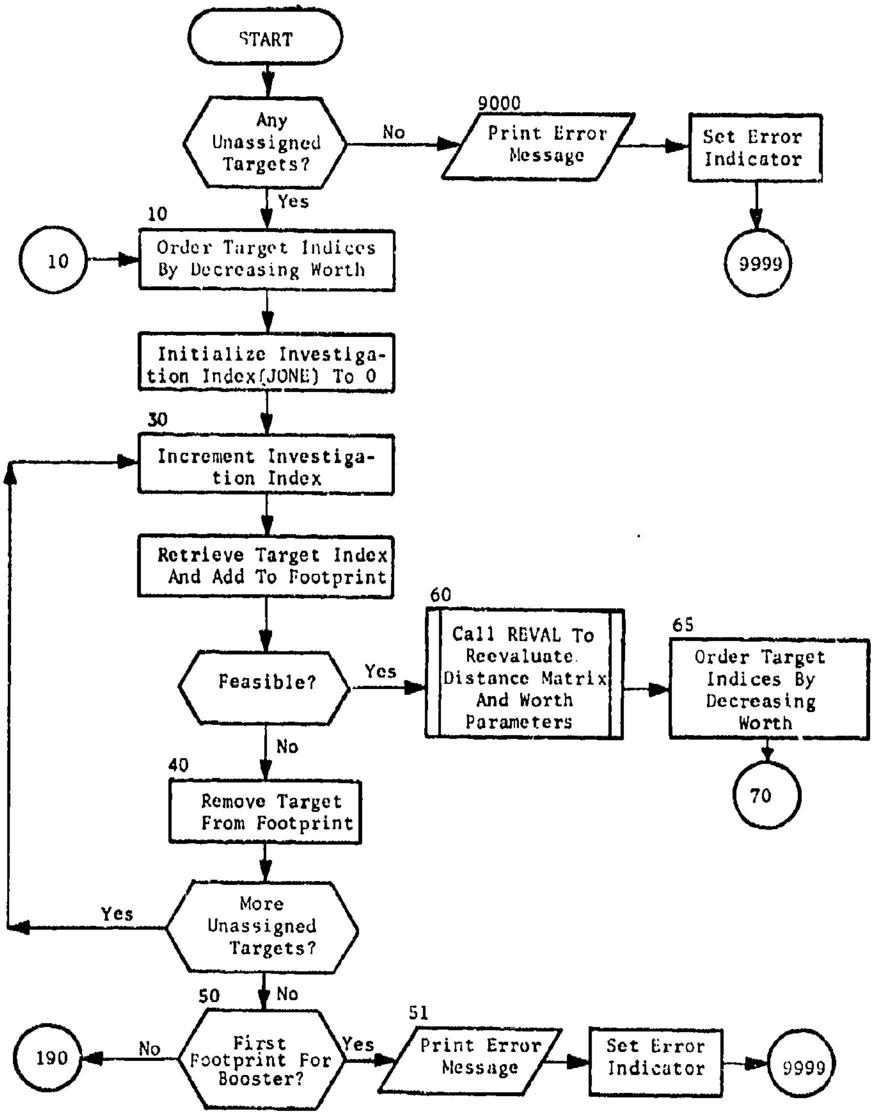


Fig. 104. Subroutine OPTBOOST
Part I: Determination of Best
Initial Point

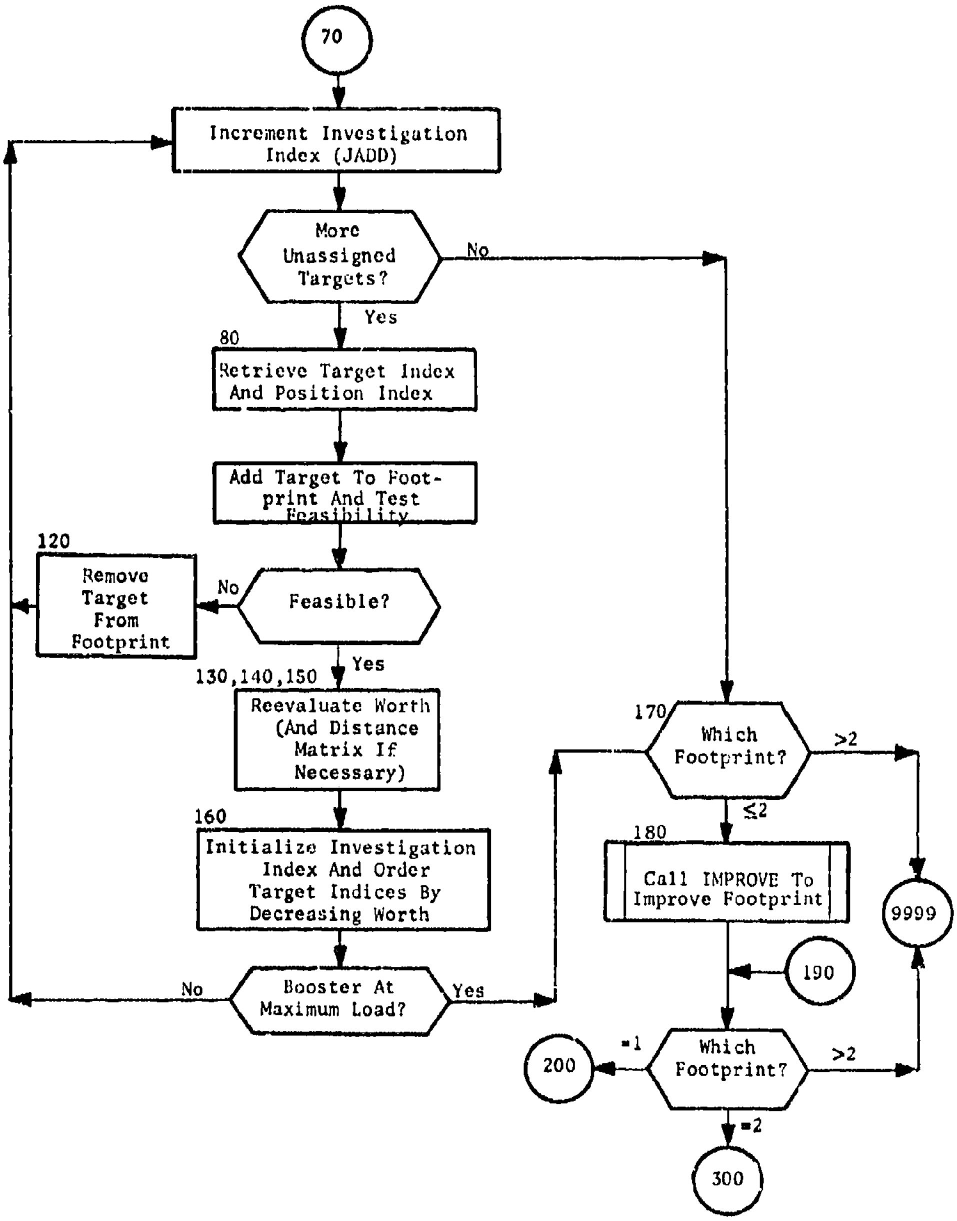


Fig. 104. (cont.)
Part II: Addition of Targets

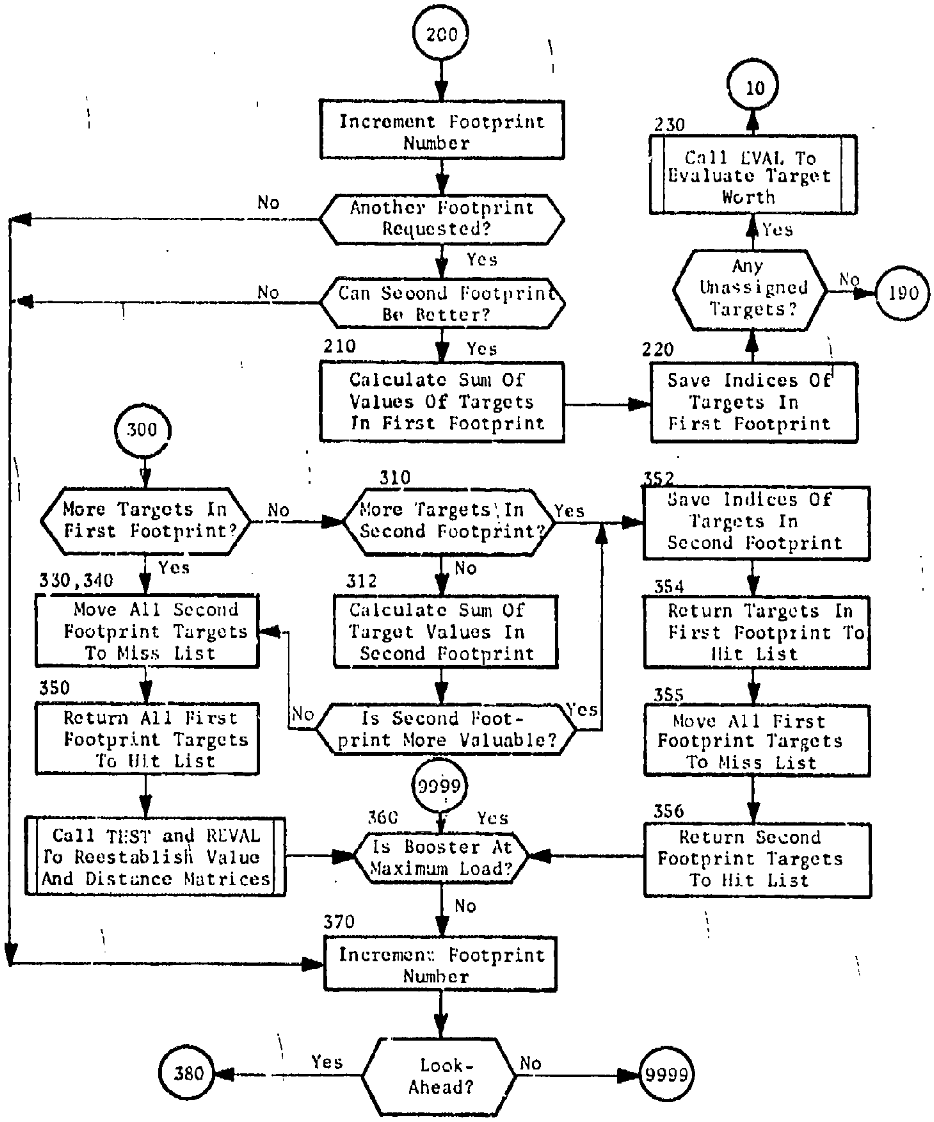


Fig. 104. (cont.)
Part III: Selection of Best Footprint

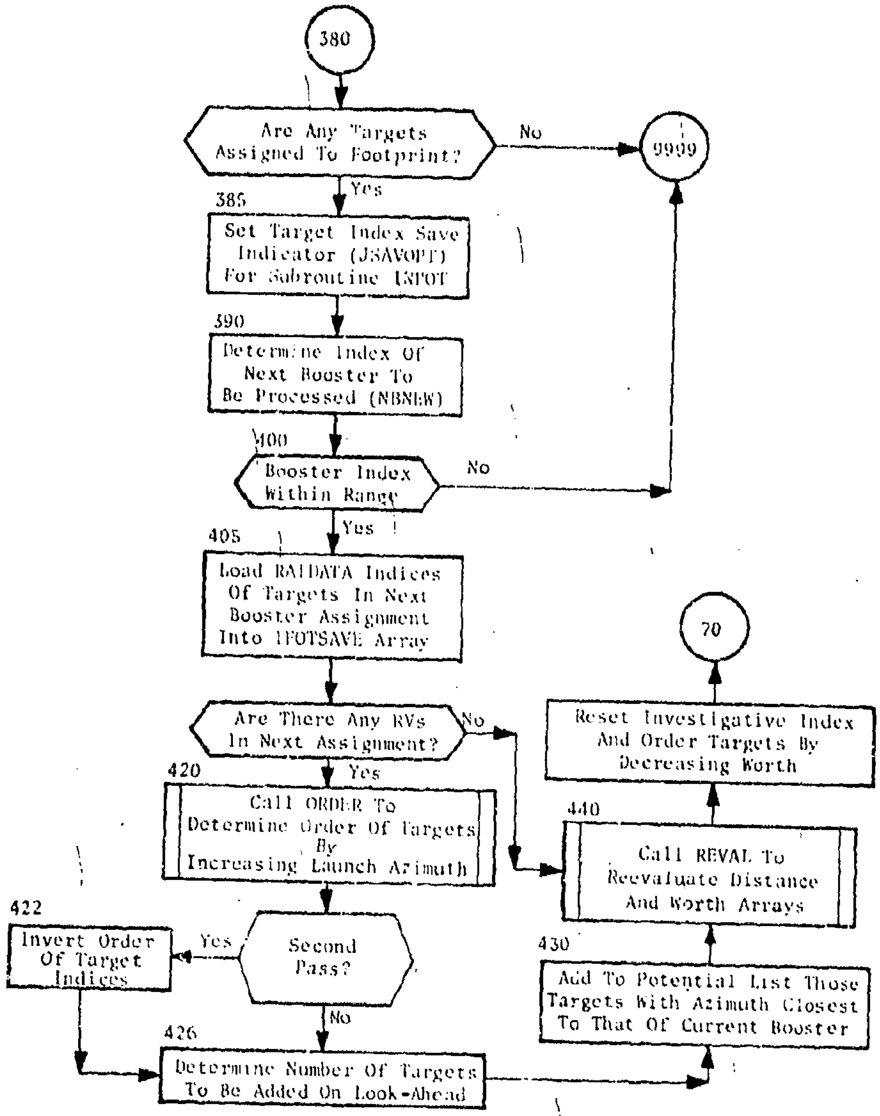


Fig. 104. (cont.)
Part IV: Look-Ahead to Next Booster Assignment

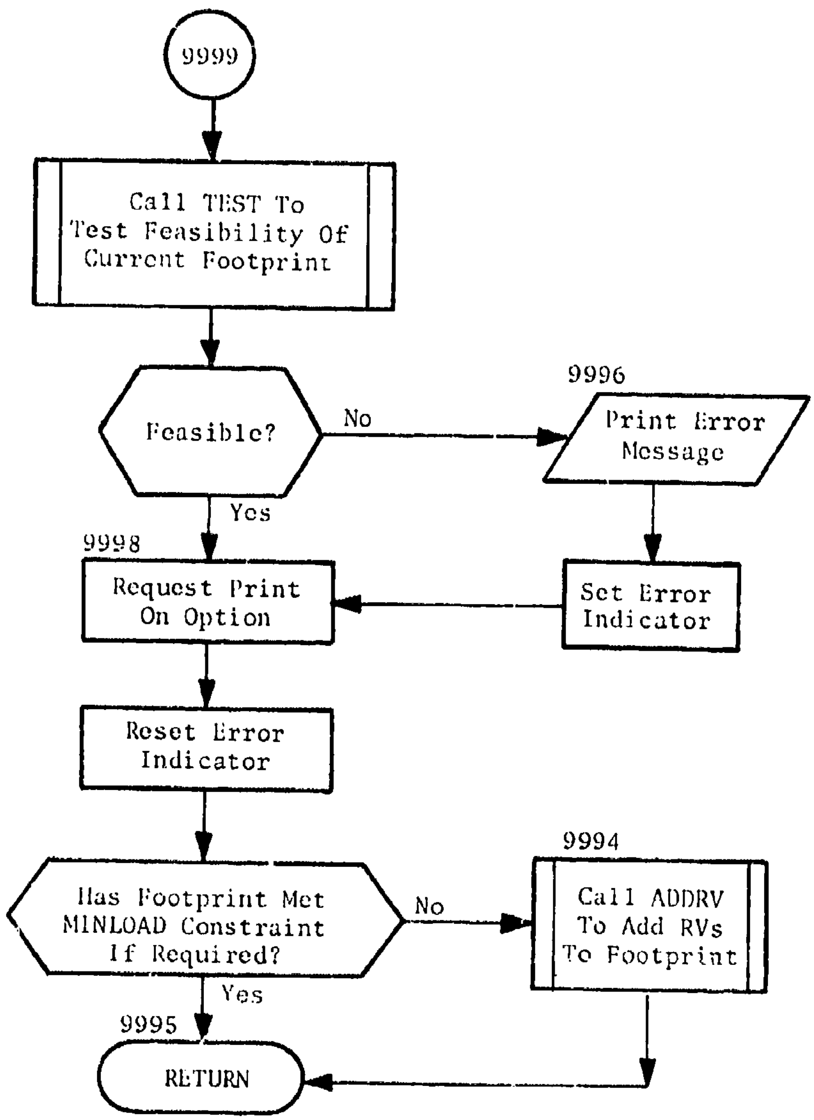


Fig. 104. (cont.)
Part V: Termination

SUBROUTINE PRINTSET

PURPOSE: This routine controls activation of print requests.

ENTRY POINTS: PRINTSET

FORMAL PARAMETERS: None

COMMON BLOCKS: STRKSUM, CONTROL, WPNTGT, FLAG

SUBROUTINES CALLED: None

CALLED BY: FOOTPRNT

Method

This subroutine is called once for each booster on each pass for each group. It sets flags (array IFLAG in common /FLAG/) which control the printing by subroutine GOPRINT.

The routine first sets all flags to false (zero) for no print requested. The print requests read by subroutine RDCARDF are then examined to determine which requests are active for the current booster, pass, and group. Each active request sets its flag to true (one) which will cause a print to be generated. If print request 14 (subroutine call chain for all subroutines) is activated, flags 16 through 35 inclusive are set true since these requests are the call chains for each individual subroutine. See subroutine RDCARDF for a discussion of the nature of the print requests.

Subroutine PRINTSET is illustrated in figure 105.

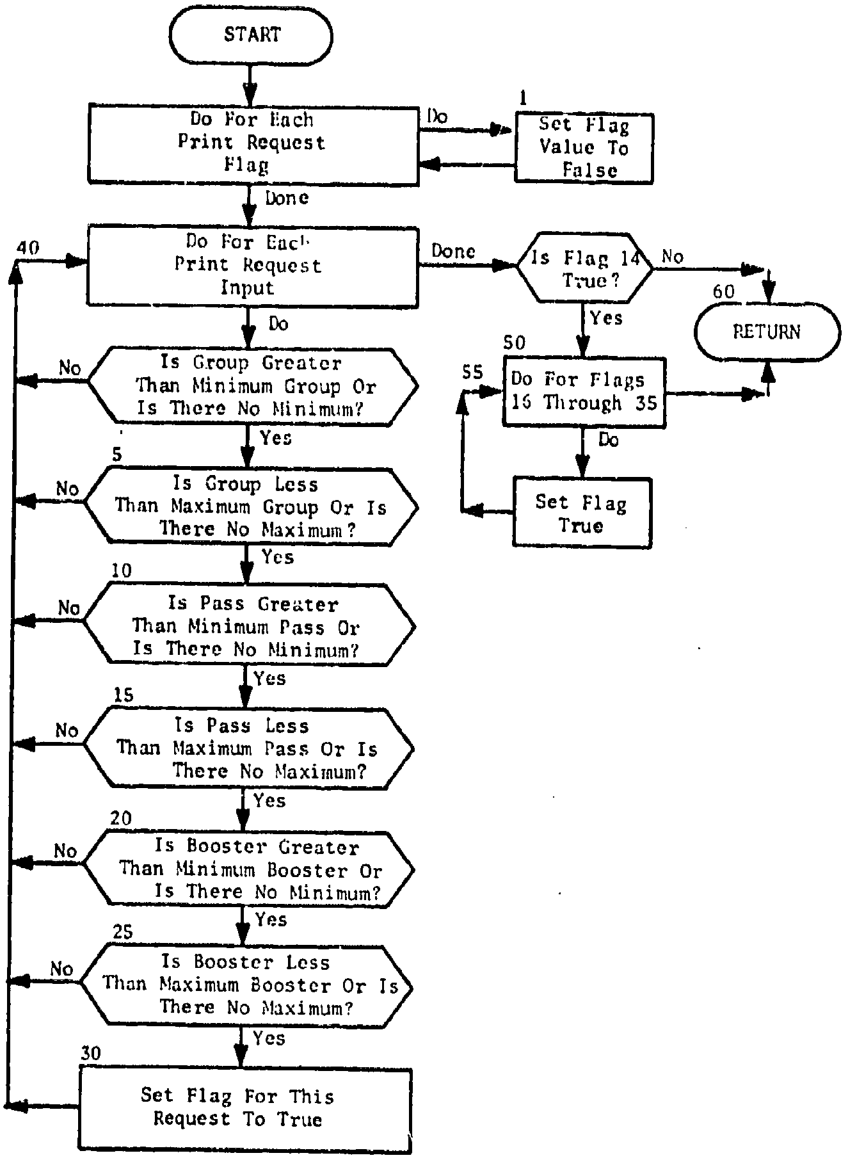


Fig. 105. Subroutine PRINTSET

SUBROUTINE PRNTREQ

PURPOSE: This routine prints the print requests.

ENTRY POINTS: PRNTREQ

FORMAL PARAMETERS: None

COMMON BLOCKS: FLAG

SUBROUTINES CALLED: None

CALLED BY: RDCARDF

Method

This subroutine is called by RDCARDF to display the print requests in common /FLAG/.

Table 29 shows the format of the print requests. If any of the last six variables in the table are blank or zero, no checking is done on that parameter.

The meaning of each print request is explained in the User's Manual, Volume II, Chapter 3, Plan Generation Subsystem, Program FOOTPRNT, Output.

Subroutine PRNTREQ is illustrated in figure 106.

Table 29. Format for Print Requests

USER-INPUT PARAMETER NAME	INTERNAL VARIABLE NAME	DESCRIPTION
PRINT	IPRNT	Print request number
NOPRINT	-	Default request to be removed
GSTARTP	IFG	First group to activate print
PASSTART	IFP	First pass to activate print
BOOSTART	IFB	First booster to activate print
GENDP	ILG	Last group to activate print
PASSEND	ILP	Last pass to activate print
BOOSTEND	ILB	Last booster to activate print

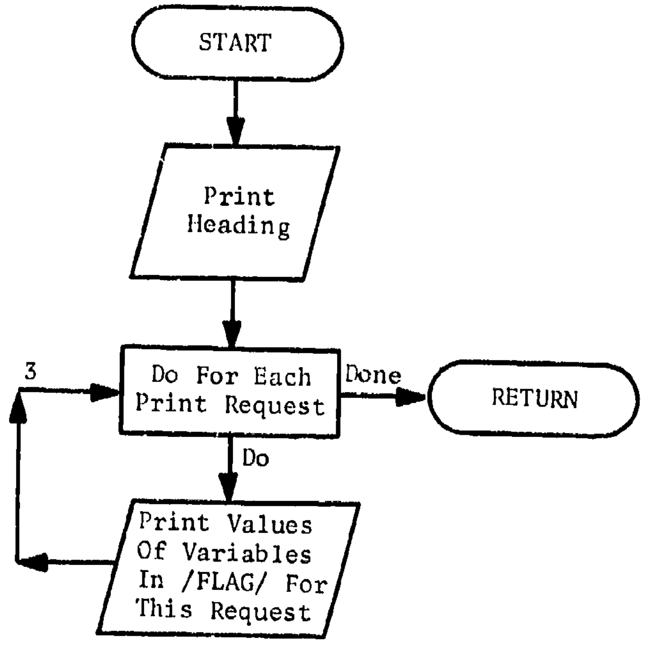


Fig. 106. Subroutine PRNTREQ

SUBROUTINE RDCARDF

PURPOSE:	This routine reads and interprets the user-input parameter cards for the assignment module.
ENTRY POINTS:	RDCARDF
FORMAL PARAMETERS:	None
COMMON BLOCKS:	RAIDATA, 4, CONTROL, FLAG, LOADATA
SUBROUTINES CALLED:	GETVALU, ITLE, NUMGET, PRNTREQ, TABLINPT, LOADREAD
CALLED BY:	FOOTPRNT

Method

The user-input parameter cards are in the QUICK free field format described in utility subroutine GETVALU. This utility routine is used to generate variable name-value pairs for the user-input parameters. The user-input parameters are the print requests and the algorithm control variables. The user-input parameters for print requests are displayed in table 29. The print modifying parameters, GSTARTP, GENDP, PASSTART, PASSEND, BOOSTART, and BOOSTEND, modify the immediately preceding print request (PRINT) on the same card. That is, a print request and its modifiers must be contained on one card. In order to remove a default print request, the parameter NOPRINT is used to specify the request.

The program control variables are IGSTART, IGEND, LOADOPT, MAXFOOT, FRACLOOK, DELAGE, PN, PURGE, and IDUMP.

The first two of these are used to determine the beginning and ending of processing for program FOOTPRNT. The first parameter, IGSTART, specifies the first group for which program FOOTPRNT will look on the TMPALOC file. The latter parameter, IGEND, determines the last group that the program will consider on that same file. These parameters are used when the user wants the program to process only a subset of the total number of groups output by program ALOCOUT.

The next parameter, LOADOPT, is used to determine the booster loading option that the user desires for processing. When its value is FREE*,

*The values listed for parameter LOADOPT are the input parameter values. Subroutine RDCARDF changes these internally as follows: ADDON to 1; MINLDREQ to 3; FREE (or otherwise) to 0.

462 546 O - 72 - 40

the program will consider only a maximum load constraint on each individual booster. That is, any loading of re-entry vehicles to the booster that does not exceed a specified maximum load will be deemed acceptable by the program. A value of ADDON* for this parameter will cause the program to attempt also to meet a minimal load constraint for each booster. If the potential assignment for a booster requires a number of re-entry vehicles that is less than the minimum load, the program will attempt to utilize more re-entry vehicles on the booster by placing extra re-entry vehicles on targets already included in the footprint. This operation continues until the minimum load constraint has been met. If the program determines it is unable to use enough extra re-entry vehicles on the booster to meet the minimum load constraint, then the booster assignment is output without meeting that constraint. If the value of the parameter LOADOPT is equal to MINLDREQ*, then the program will not output a booster assignment with less than the minimum load. If the program is unable to assign a footprint to a booster with the minimum required number of re-entry vehicles, then no assignment at all will be the output for that booster. In all cases, the program will abide by the maximum load constraint.

The next parameter, MAXFOOT, controls the amount of effort expended in subroutine OPTBOOST in generating a footprint for each booster. If the value of this parameter is negative, there will be no "look-ahead" to later boosters to determine if targets previously assigned can possibly be added to the current footprint. If the parameter is positive, the fraction, FRACLOOK, of the targets assigned to the next booster to be processed is added to the potential target arrays during OPTBOOST processing. This look-ahead feature enables the subroutine to consider targets which might be assigned to a later booster but can profitably be added to the current one. The absolute value of this parameter, MAXFOOT, determines the number of separate footprints which are generated from the potential target arrays by subroutine OPTBOOST. If MAXFOOT is equal to ± 1, then only one footprint will be generated from these arrays. If MAXFOOT is equal to ± 2 (which is the maximum absolute value that can be assigned this parameter) then two separate footprints are generated from the potential target arrays. The program then selects for further processing the footprint with either the greater number or greater value of targets assigned. If more than one footprint is desired from any potential target array set, the program produces two footprints which share no targets. The next parameter, FRACLOOK, is the fraction of the target assignments for the next booster to be processed that will be added to the potential target arrays during the look-ahead function of subroutine OPTBOOST. (The value of this parameter should not exceed one.)

Two of the remaining parameters on the control card deal with the deletion of weapons from the potential target list. The first, DELAGE, is a factor

*The values listed for parameter LOADOPT are the input parameter values. Subroutine RDCARDF changes these internally as follows: ADDON to 1; MINLDREQ to 3; FREE (or otherwise) to 0.

which multiplies the variable AGE as each booster is processed. This variable, AGE, is used to divide the nominal worth of maintaining a target in the potential target arrays. As the AGE of a target increases, it becomes less and less valuable for retention in the potential target arrays. AGE is modified as each booster is processed by multiplication by the factor DELAGE. As DELAGE increases, targets will remain in the potential target arrays for fewer boosters. (The value of DELAGE should always be greater than one.) The other variable which controls the deletion of weapons from the potential target arrays is PURGE. This parameter is the fraction of the targets in the potential target arrays that are to be removed at the end of the processing for each booster. Subroutine BOOSTIN removes this fraction of the potential target arrays before adding the targets which are assigned to the next booster to be processed. If, however, the number of targets in the potential target arrays is less than the average booster load, then no targets are removed. This feature prevents an excessive value for the parameter PURGE from eliminating all targets from the potential arrays except those which are in the previous assignment. PURGE however should not be set too small, since the processing time for the program is greatly affected by the number of targets in the potential target arrays. As this number increases, processing time increases according to the square of the number of targets in the potential target arrays.

The factor PN is a weighting factor for the value line used in the function VALF. As the value of PN increases, the value scheme gives more weight to targets with many close neighbors. (That is, targets with many close neighbors are deemed more worthy to be included in the current footprint.) The specific function of this parameter is to determine the fraction of the length from the minimum load (MINLOAD) to the number of average targets per booster (NARV) which becomes the Y intercept of the value control lines. This value control line is a straight line whose Y intercept is determined by PN and whose X intercept is the maximum load value (MAXLOAD). This line determines the input weighting factor to the function VALF. The parameter IDUMP is used to abort the run with a memory dump following the print.

After reading and printing the values of the user-input parameters, the subroutine calls subroutine TABLINPT to read the footprint parameter tables. These tables comprise the footprint constraints which are to be imposed by this program. The format for these data is discussed in the section covering subroutine TABLINPT. If the loading option is different from the free load option (i.e., LOADOPT≠0)*, then subroutine LOADREAD is called to read any data on booster loadings that are required. (At present, no further data are required and subroutine LOADREAD would return without reading any further data cards. This subroutine is merely included to provide for expansion to other booster loading options.)

Subroutine RDCARDF is illustrated in figure 107.

---

*Input value=FREE

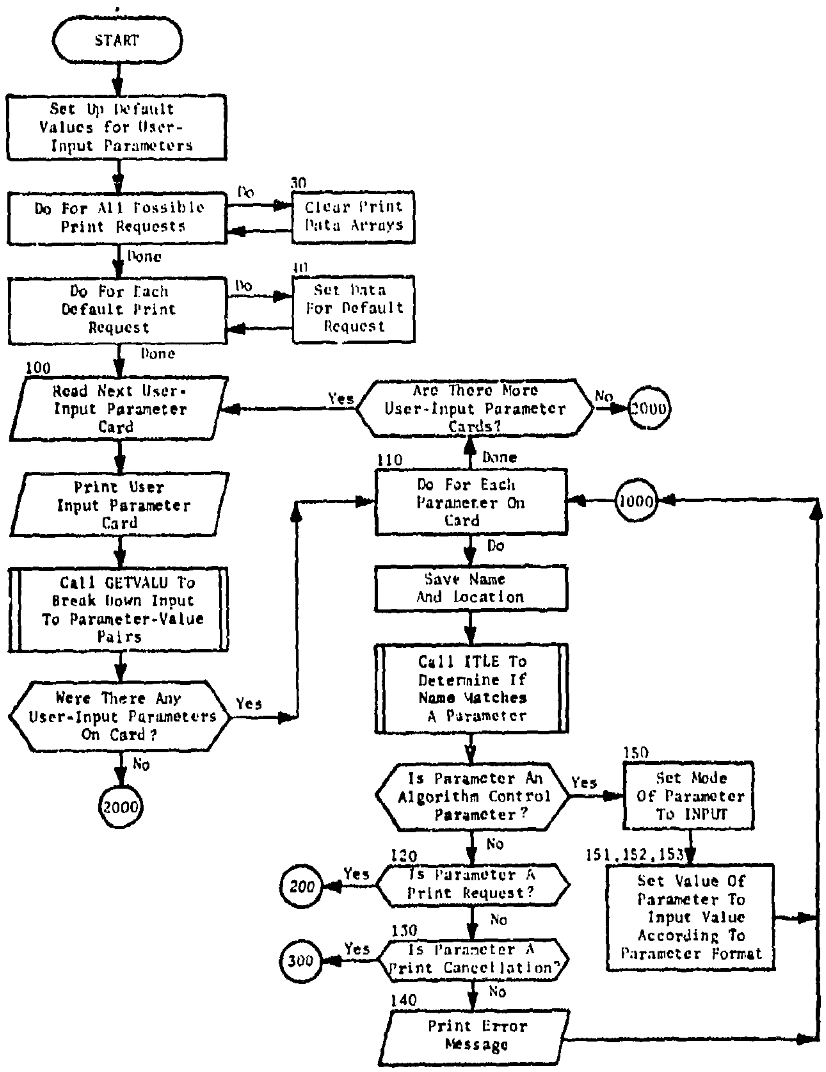


Fig. 107. Subroutine RDCARDF
Part I: Initialization and Processing
of Algorithm Control Parameters

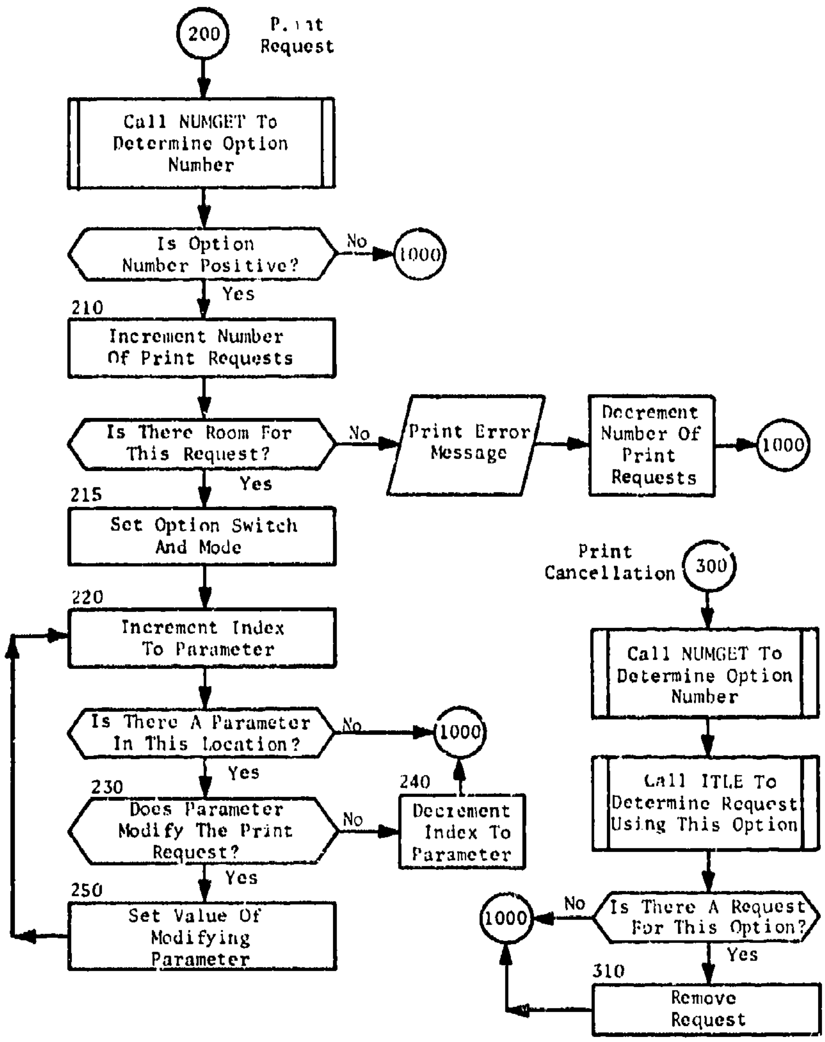


Fig. 107. (cont.)
Part II: Print Request and Cancellation Processing

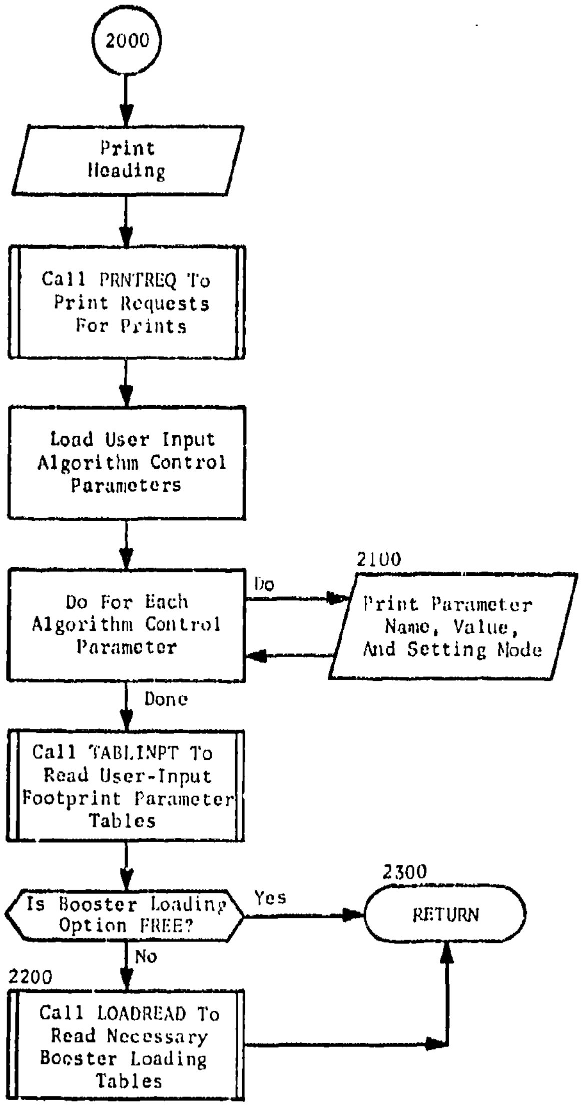


Fig. 107. (cont.)
Part III: Termination Processing

SUBROUTINE REMOVE

PURPOSE: This routine removes a target from its booster assignment.

ENTRY POINTS: REMOVE

FORMAL PARAMETERS: None

COMMON BLOCKS: RAIDATA, 4, PERFORM, WPNTGT, 2, DEBUG, PRINT

SUBROUTINES CALLED: GOPRINT

CALLED BY: FOOTPRNT, INPOT

Method

This subroutine performs the list pointer manipulation required to remove a target from its booster assignment in the RAIDATA list. The target whose RAIDATA index is in the variable JTGTD in common /WPNTGT/ is removed. The forward and backward pointers are recalculated, the number of targets assigned (total and by booster) is decremented, and the maximum index indicator (by booster) is updated if necessary.

Subroutine REMOVE is illustrated in figure 108.

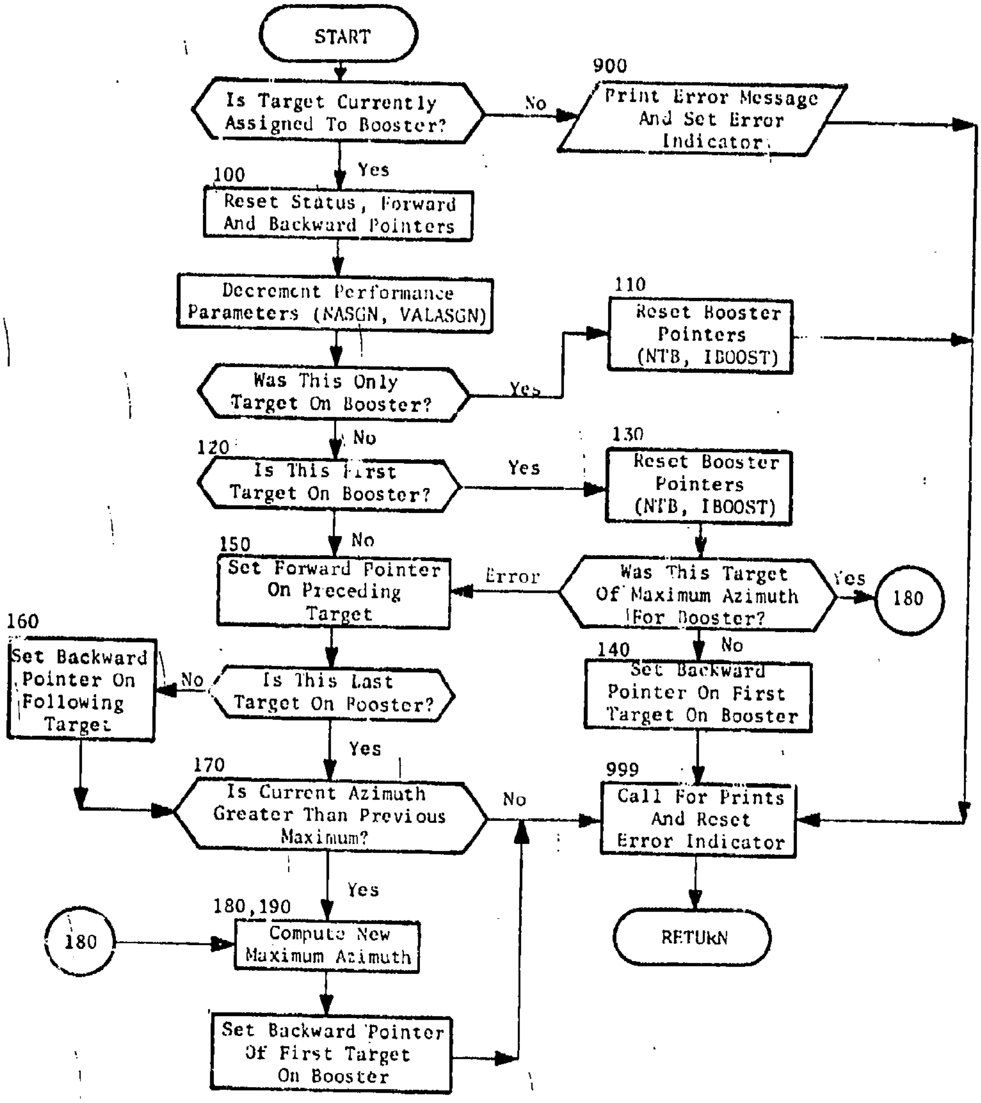


Fig. 108. Subroutine REMOVE

SUBROUTINE SETDATA

PURPOSE: This routine retrieves (from the ITABL file) and loads the correct footprint processing data into the footprint test arrays for use by subroutines FOOTEST and GOPRINT.

ENTRY POINTS: SETDATA

FORMAL PARAMETERS: I - A system identification number, IMIRV

COMMON BLOCKS: PARAMETER, FOOTDATA, SHRTDATA, PENADD, FILES, TSCRATCH, Filehandler Blocks (ITP, MYIDENT, TWORD, NOPRINT, FILABEL)

SUBROUTINES CALLED: ABORT, SKIP, SETREAD, RDARRAY, RDWORD, TERMTAPE

CALLED BY: FOOTPRNT, PRNTABLE

Method

This routine merely moves data from the footprint parameter scratch file, ITABL, to the footprint test arrays. (See table 30.) As subroutine TABLINPT reads the footprint parameter data, it writes them on the ITABL file.

SETDATA first retrieves the system MTYPE and IDATA from those arrays in common /PARAMETR/, using the formal parameter I, as an index.

SETDATA then determines if the correct system data are already in the footprint testing storage. If so, the routine exits. Otherwise the ITABL file is searched for the correct data. (Table 31 shows the format of this file.) If the data are not found, the filehandler will abort the run. Upon finding the data, SETDATA transfers them to the appropriate array as listed in table 30.

Subroutine SETDATA is illustrated in figure 109.

Table 30. Footprint Parameter Data Transmission

MTYPE	SYSTEM	FOOTPRINT TESTING ARRAY	ARRAY LENGTH	FOOTPRINT TESTING COMMON BLOCK
1	Long-Range	ISLD	LLNGDAT	/FOOTDATA/
2	Short-Range	ISSD	LSHTDAT	/SHRTDAT/
3	Long-Range With Penetration Aids	ISRFD ISLD	LPENDAT	/PENADD/ /FOOTDATA/

Table 31. Format for Footprint Parameter Data Scratch File

Each unique system is output on the ITABL file in the following format:

VARIABLE	LENGTH	DESCRIPTION
MTYPE	1	MIRV system functional type
IDATA	1	MIRV system data set number
"LENGTH"*	1	Length of footprint parameter table for this system
"TABLE"**	LENGTH	Footprint parameter table

*For MTYPE=1, LENGTH is LLNGDAT; MTYPE=2, LENGTH is LSHTDAT; MTYPE=3, LENGTH is LPENDAT (see table 30).

**For MTYPE=1, TABLE is the ISLD array; MTYPE=2, TABLE is the ISSD array; MTYPE=3, TABLE is the ISFRD and ISLD arrays (see table 30).

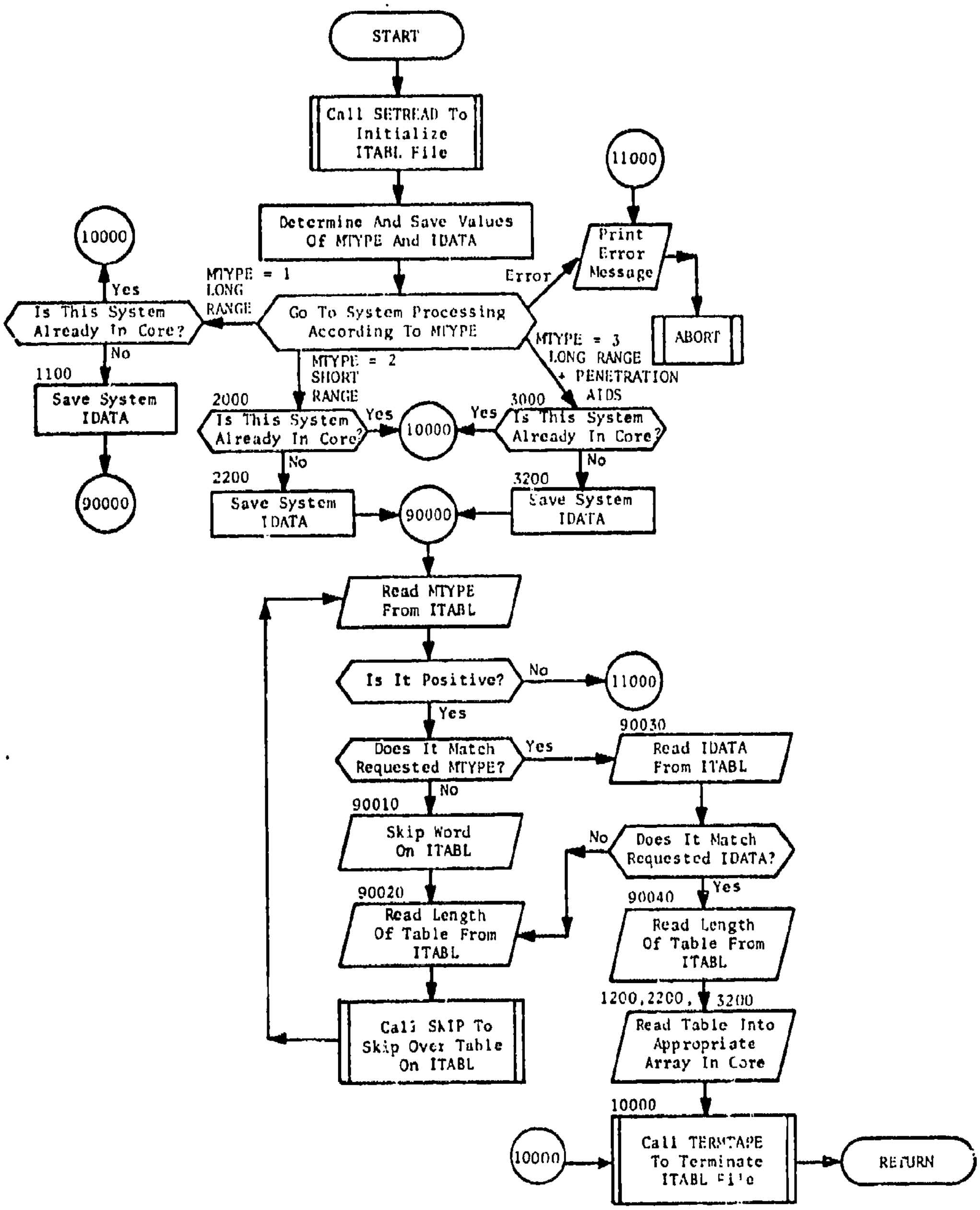


Fig. 109. Subroutine SETDATA

SUBROUTINE TABLINPT

PURPOSE:	This routine reads and prints the footprint parameter tables, and saves them on the ITABL file.
ENTRY POINTS:	TABLINPT, PRNTABLE
FORMAL PARAMETERS:	None
COMMON BLOCKS:	RAIDATA, 4, PARAMETR, FOOTDATA, SHRTDAT, PENADD, FILES, TSCRATCH, Filehandler Blocks (ITP, MYIDENT, TWORD, NOPRINT, FILABEL)
SUBROUTINES CALLED:	NUMGET, SETDATA, SETWRITE, WRARRAY, WRWORD, TERMTAPE
CALLED BY:	RDCARDF (TABLINPT), GOPRINT (PRNTABLE)

Method

This routine reads the footprint parameter tables and stores the data in common block /PARAMETR/ and outputs the data on the footprint parameter data scratch file, ITABL. Entry PRNTABLE calls subroutine SETDATA to transfer the data from the ITABL file to the footprint testing parameter arrays (in /FOOTDATA/, /SHRTDAT/, and /PENADD/) and then prints the data. Common /RAIDATA/ is used as temporary scratch storage by this subroutine. The local array IGOT is used to store the IMIRV numbers of those systems whose parameters have been read. NGOT is the number of system data sets that have been read.

Each MIRV system with a unique IMIRV number must be defined with a system title card.* The data on this card are stored in common /PARAMETR/. (See table 25 for definitions of the variables in this block.)

The data required for each data set depend on the system type number, MTYPE. This variable defines the system to be long-range, short-range, or long-range with penetration aids. (See table 25) Within each type, there may be many different data sets identified by the data set number, IDATA. (The values of this parameter need be unique only within each MTYPE value. The values need not be consecutive.) As each data set is read, it is output on the ITABL file according to the format shown in table 31. A data set need be read only once regardless of the number of systems that use it. If the values of MTYPE and IDATA read from a system title card match values already read, then the routine merely reads the next title card.

*For each value of the attribute IMIRV, there should be one title card. The Hollerith name of the system (IHNAME in common /PARAMETR/) is used only to identify the system in the print of the footprint parameter tables. It has no effect on footprint generation.

Entry PRNTABLE retrieves the data for each defined system and prints the data.

Each formula's data cards are preceded by one system title card requesting that formula. The reading of data is terminated by a title card with a zero or negative IMIRV value. The systems can be input in any order.

If more than one IMIRV value refers to a specific formula for footprint test (see below), then the data for that formula must follow immediately the first occurrence of a system title card requesting the use of that formula. Succeeding title cards with the same formula definition need no data following them.

A formula for footprint testing is defined by two variables input on the system title card. The first, MTYPE, references the functional form of the formula to be used. If MTYPE = 1, the exponential functions of the long-range system are used. MTYPE = 2 requests the short-range functions. MTYPE = 3 requests the long-range system with a full load of area penetration aids. Within each type, there are data sets for the parameters used in the function. Thus, formula definition requires MTYPE, the functional form indicator, and IDATA, the index to the parameter set. For example, if two long-range systems are desired there would be two formula definitions: MTYPE = 1, IDATA = 1; MTYPE =1, IDATA =2.

The formulae and data for both long-range and short-range systems have been taken from "Strategic Offensive Weapons Employment In The Time Period About 1975 (U)", (TOP SECRET) Weapons Systems Evaluation Group Report, R-160, August 1969, Volume VI, Allocation of MIRV Systems.

Long-Range System -- MTYPE=1

The long-range system can have either one, two, or three re-entry vehicles on a booster.

The system functions are defined by a series of regression coefficients which, when applied to these functions, produce results which fit the actual physical characteristics of the MIRV system. These coefficients, (e.g., RBASIC, RADD, etc.) have no names in the aforementioned document, but since the form of the equations is the same in this program and in the document, a correspondence between the two is easily determined.

The system functions are as follows:

1. <u>Fuel Load at Booster Separation (Pounds)</u>: Constant with number of RVs.

2. <u>Maximum Booster Range (RM in Nautical Miles)</u>:

   RM = RBASIC + RADD * SINE(AZIMUTH)

RBASIC and RADD are functions of the number of RVs and the sign of the azimuth.

3. Range Extension Consumption: number of Nautical miles traversed per pound of fuel

NM/FUEL = RX + RAXX * SINE(AZIMUTH)

RX and RAXX are functions of the number of RVs and the sign of the azimuth.

4. RV Toss Equations: nautical miles per unit fuel

NM/FUEL = G * (TOSSC1 + TOSSC2 * SINE(AZIMUTH))

where

$$G = \text{EXPF}\left(\text{TEONE} * \left(\frac{\text{RM-R}}{\text{TDENOM}}\right)^{\text{**TETWO}}\right)$$

where

RM = maximum booster range (nautical miles)
R = range to initial target (nautical miles)

TOSSC1 and TOSSC2 are functions of number of RVs originally on board, number of RVs currently on board, and sign of launch azimuth.

TEONE and TETWO are functions of number of RVs originally on board and number currently on board.

5. Crossrange to Downrange Multiplier (CROSSDWN):

CROSSDWN = G * (CONE + CTWO * SINE(AZIMUTH))

where

$$G = \text{EXPF}\left(\text{EONE} * \left(\frac{\text{RM-R}}{\text{DENOM}}\right)^{\text{**ETWO}}\right)$$

CONE and CTWO are functions of the number of RVs currently on board and the sign of the azimuth.

EONE and ETWO are functions of the number of RVs.

DENOM is a constant.

Short-Range System -- MTYPE=2

This system does not consider launch azimuth. It considers configurations containing from 1 to 16 RVs on board. The system functions are as follows: (Let R be the distance in nautical miles from the launch base to the initial target in the footprint.) The parameters for this type are also coefficients calculated by a curve fit to observed physical data.

1. Fuel Load at Booster Separation:

   $$TF = BETATWO * R^2 + BETONE * R + BETAZ$$

   The parameters are functions of the number of RVs on board.

2. Maximum Booster Range: This is a parameter, MAXRBOOST, as a function of the number of RVs carried to the first target.

3. RV Toss Consumption Equations:

   $$NM/unit\ fuel = ALPHATWO * R^2 + ALPHAONE * R + ALPHAZ$$

   Those parameters are functions of the number of RVs on board.

4. Crossrange to Downrange Multiplier:

   $$CROSSDWN = GTWO * R^2 + GONE * R + GZERO$$

   These parameters are constant.

5. Uprange to Downrange Multiplier:

   $$UPDOWN = DONE * R + DZERO$$

   These parameters are constant.

Long-Range System With Penetration Aids: MTYPE=3

This system is similar to the long-range system (MTYPE=1). The equation forms are the same except for the first set, fuel load at booster separation. All the other constraints have the same functional form as the previous type.

Calculation of the fuel load at booster separation is as follows:

1a. Fuel Available for Footprinting: (FAFF in pounds)

   $$FAFF = TGAS - SRF$$

TGAS - Total fuel load on board last state (pounds)
SRF - Fuel required to space and release penetration aids and re-entry vehicles

1b. Spacing and Release Fuel: (SRF in pounds)

SRF = G * (SRFC1 + SRFC2 * SINE(AZIMUTH))

where

$$G + EXPF \left(SRFEXP1 * \left(\frac{RM-R}{SRFDEN} \right) **SRFEXP2 \right)$$

where

RM = maximum booster range in nautical miles
R = range from launch base to first target in footprint.

TGAS and SRFDEN are constants.

SRFC1, SRFC2, SRFEXP1, and SRFEXP2 all depend on the number of RVs initially on board the booster.

Note: The long-range system with MTYPE=1 is a special case of this type. For the former system, the spacing and release fuel is considered to depend only on the number of RVs initially on board. Thus the detailed computation of this fuel is unnecessary.

Subroutine TABLINPT is illustrated in figure 110.

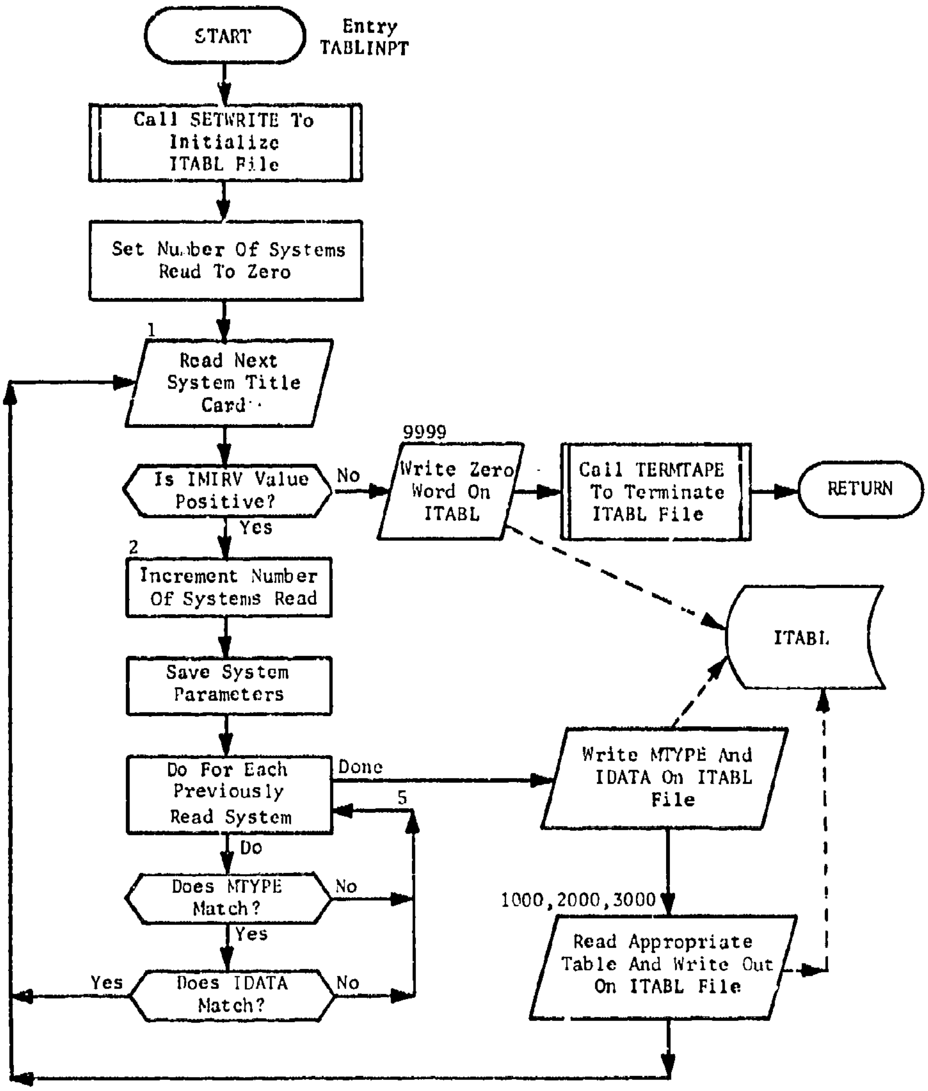


Fig. 110. Subroutine TABLINPT
Part I: Entry TABLINPT

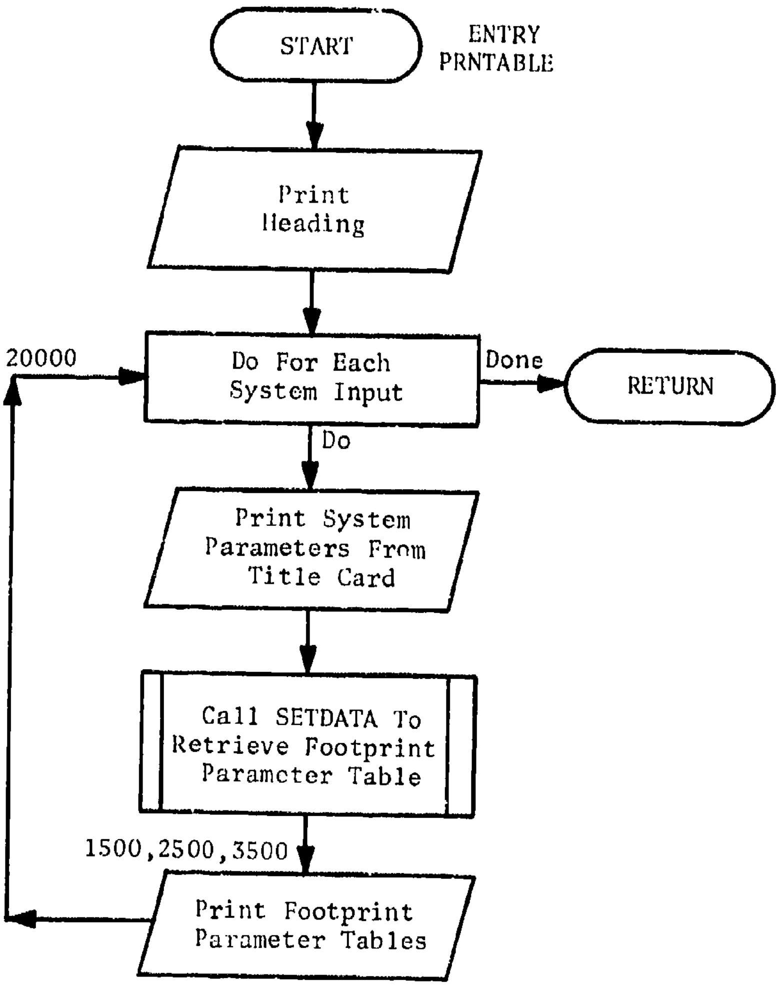


Fig. 110. (cont.)
Part II: Entry PRNTABLE

SUBROUTINE TEST

PURPOSE:	This routine sets up the test arrays in common /FOOTIO/ for footprint testing.
ENTRY POINTS:	TEST
FORMAL PARAMETERS:	None
COMMON BLOCKS:	FOOTIO, LOADATA, PARAMETR, POTENT, 1, 3, DEBUG, PRINT
SUBROUTINES CALLED:	GOPRINT, FOOTEST
CALLED BY:	CHKSEQ, FUELSAVE, IMPROVE, OPTBOOST

Method

This subroutine is the interface between the assignment section of the program and the testing section. It loads the RIN and THIN arrays in common /FOOTIO/ from the data in the hit list (IHIT in /POTENT/) and the temporary data arrays (RP and TP in /3/).

The only logical complication to this routine is the result of the booster loading options that require that the minimum load constraint must be met by every booster; i.e., LOADOPT≥3*. In this case, if there are not enough targets in the hit list to meet the requirement, subroutine TEST adds the needed RVs to the first target in the list. Since it uses the same method to add RVs as subroutine ADDRV, this addition guarantees that no footprint will be declared feasible unless it can contain at least the required minimum load. The number of RVs added is stored in the local variable NOFFSET. This variable is used to manipulate the data arrays to show the correct entries for each real target in the footprint. If the footprint with the added vehicles proves feasible, one of the added vehicles is removed and FOOTEST is called again. This second call is required to load the correct values in the DELRAFT and TOFLY arrays.

Subroutine TEST is illustrated in figure 111.

*Input value=MINLDREQ

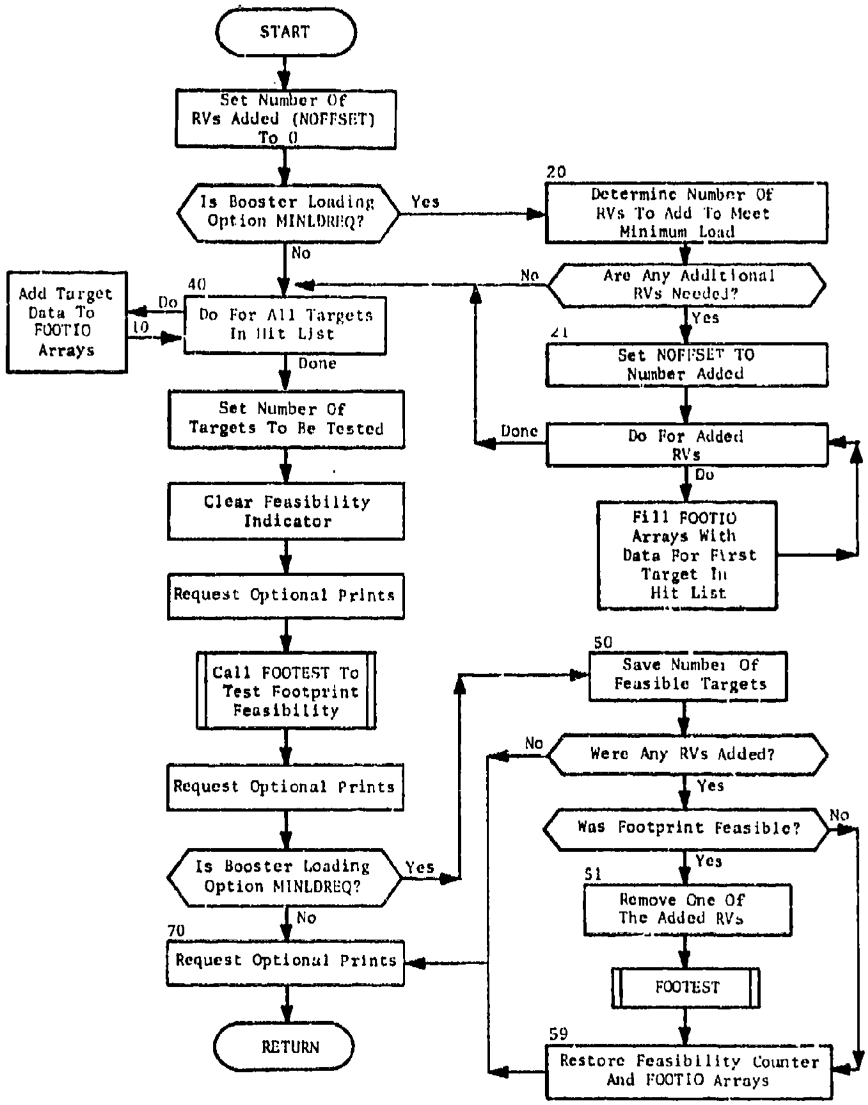


Fig. 111. Subroutine TEST

SUBROUTINE TRANSFER

PURPOSE:	This routine transfers blocks of data from the TMPALOC file to the ALOCGRP file.
ENTRY POINTS:	TRANSFER, INITRANS
FORMAL PARAMETERS:	N - See below
COMMON BLOCKS:	RAIDATA, 4, DEBUG, PRINT, Filehandler Blocks (ITP, MYIDENT, TWORD, NOPRINT, FILABEL)
SUBROUTINES CALLED:	OOPRINT, RDARRAY, WRARRAY
CALLED BY:	FOOTPRNT

Method

These two entries are used to transfer data from the TMPALOC file to the ALOCGRP file.

Entry INITRANS

The formal parameter N is the logical unit number of the file to which data are to be transferred. This unit number is saved in variable IWRITE, and control is returned to the calling program.

Entry TRANSFER

For this entry, the formal parameter N specifies the number of words of data that are to be read from file ITP (or IREAD) and written on logical file number IWRITE. The words are merely transferred from one tape to the other. TRANSFER assumes subroutine SETWRITE has been called for file IWRITE.

Note: The length of common /RAIDATA/ from the beginning to LRAID is stored in LRAID. Since TRANSFER uses this length in determining the size of temporary storage, changes in common /RAIDATA/ should be reflected in this variable.

Subroutine TRANSFER is illustrated in figure 112.

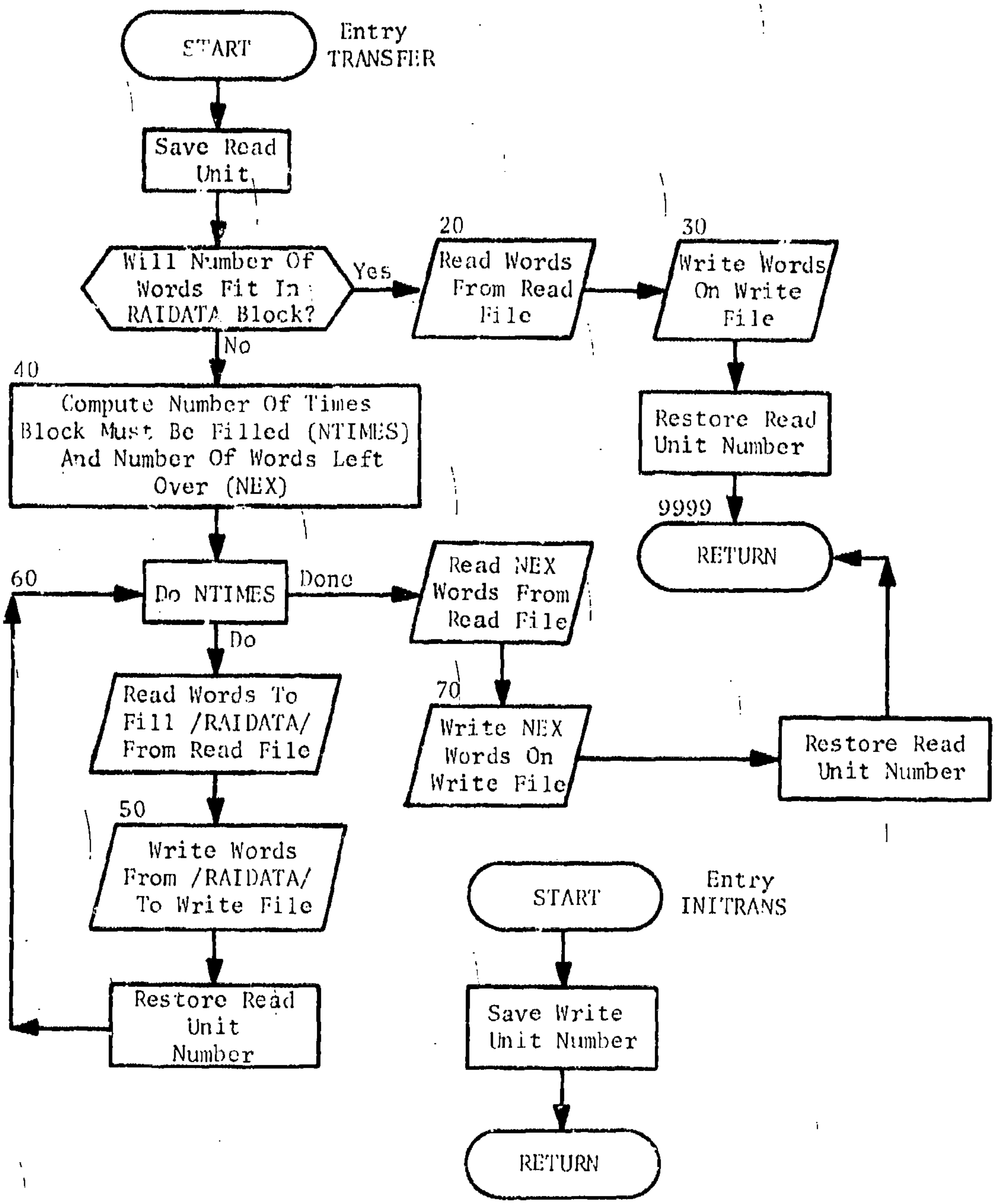


Fig. 112. Subroutine TRANSFER

FUNCTION UPTODOWN

PURPOSE:	This function computes the multiplier by which uprange distance must be multiplied to calculate equivalent downrange distance.
ENTRY POINTS:	UPTODOWN
FORMAL PARAMETERS:	I - System type - MTYPE R - Range to first target (nautical miles) AZ - Launch azimuth of booster (radians) N - Number of re-entry vehicles carried
COMMON BLOCKS:	FOOTDATA, SHRTDAT, PENADD
SUBROUTINES CALLED:	None
CALLED BY:	BOOSTIN, EVAL, FOOTEST

Method

This function computes the uprange-to-downrange distance multiplier for use by the footprint testing subroutines. It uses the equations displayed in the discussion of subroutine TABLINPT. This function merely uses a computed GO TO statement to direct processing to the correct equation. The system type (formal parameter I) determines which equation is used. The remaining formal parameters provide the data for the multiplier calculation.

Function UPTODOWN is illustrated in figure 113.

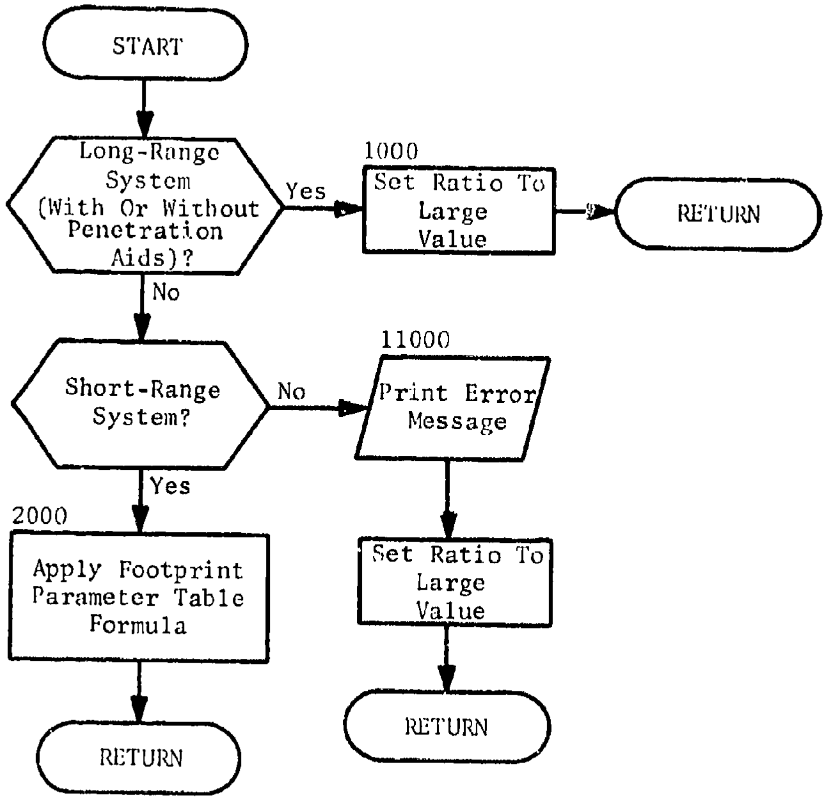


Fig. 113. Function UPTODOWN

FUNCTION VALF

PURPOSE: This function provides intertarget values for use in in the worth calculation.

ENTRY POINTS: VALF

FORMAL PARAMETERS: X - A ratio of distances
FN - A weighting parameter

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: FOOTPRNT, EVAL, BOOSTIN

Method

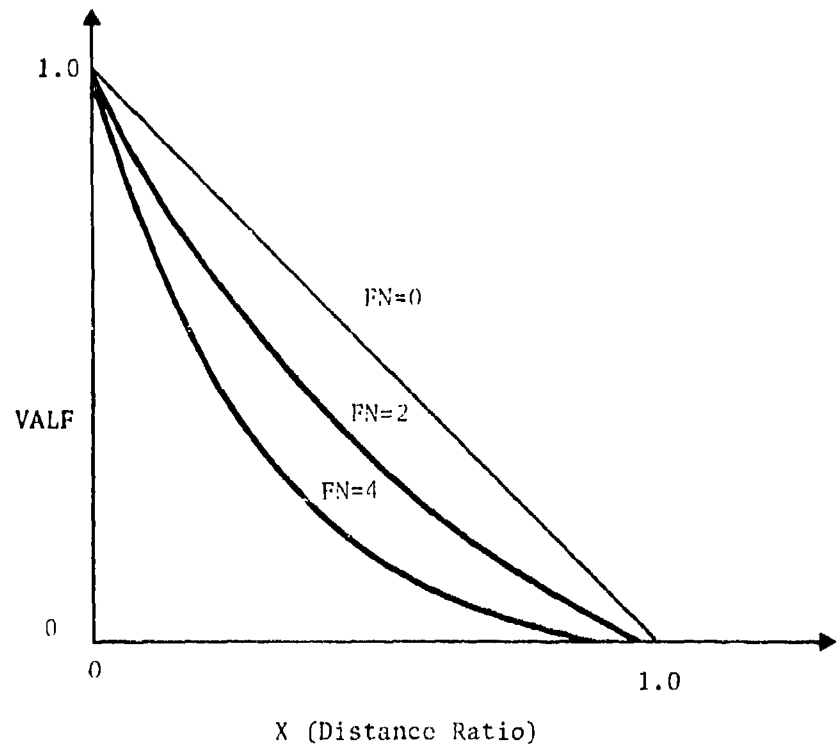
This function merely computes values for the following equation:

$$VALF = \begin{cases} (1 - X)/(1 + (X * FN)) & X \leq 1 \\ 0 & X > 1 \end{cases}$$

Figure 114 displays representative curves for this function for three values of FN.

The formal parameter X (usually called ALPHA in the calling program) is a ratio of intertarget equivalent downrange distance to a maximum feasible equivalent downrange distance. The formal parameter FN is a weighting parameter. As FN increases, the value declines more rapidly with increasing values of X. Increasing FN has the effect of increasing the worth of targets with many close neighbors.

Function VALF is illustrated in figure 115.



$$VALF = \frac{1-X}{1+X*FN}$$

Fig. 114. Value Function Implemented in VALF

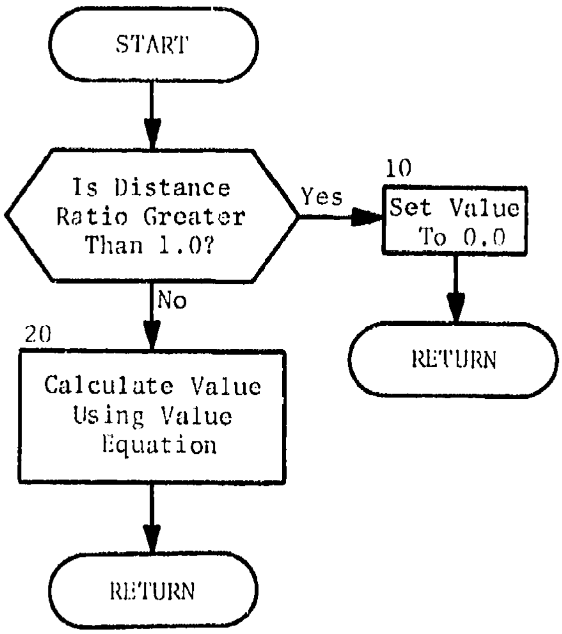


Fig. 115. Function VALF

CHAPTER 7
PROGRAM POSTALOC

PURPOSE

The purpose of the post-allocator, program POSTALOC, is to write missile and bomber delivery plans from the weapon-to-target allocations developed by the allocator, program ALOC. In the case of missiles, this is a simple process since the missile flight plans (as required by the Simulator) are completely determined once the target and launch coordinates are known. In the case of bombers, the process is more complicated. The development of bomber sorties requires the association of several strikes in a single sortie. Moreover, it is necessary to associate each sortie with specific launch and recovery bases and to select a flight profile which specifies where low-altitude capability should be used. Since the allocator does not distinguish between bombs and air-to-surface missiles (ASMs) carried by the same aircraft, it remains for the post-allocator to determine which targets should be targeted with bombs and which with air-to-surface missiles.

INPUT FILES

POSTALOC uses two input files: BASFILE and either ALOCGRP or TMPALOC. BASFILE is written by program PREPALOC. If MIRVs are present, POSTALOC uses the ALOCGRP file which contains the target allocation from ALOC, as rearranged and rewritten by programs ALOCOUT and FOOTPRNT. If no MIRVs are present, POSTALOC uses the TMPALOC file which is output by program ALOCOUT.

Subroutine GETGROUP reads the following data from the BASFILE:

1. Common /MASTER/, containing basic information about the data base, such as number of weapon groups, number of penetration corridors, number of targets, etc. (See table 34 for a complete description of this and all other common blocks.)

2. Common /FILES/, containing the logical unit numbers of all the files in the Plan Generator.

3. Common /CORRCHAR/, containing the general characteristics of each penetration corridor.

4. Common /ASMTABLE/, containing the characteristics of each of the ASM types.

5. Common /PAYLOAD/, defining the payloads of the various bomber types.

6. Common /DPENREF/, containing the coordinates of the depenetration and refuel points.

Subroutine GETGROUP skips subsequent BASFILE data until it reaches the word of Hollerith Z's which marks the beginning of the weapon group data. It then reads for each weapon group common /GRPDATA/, containing basic data about the weapon group and its bases, and common /GRPTYPE/, containing type characteristics of the bomber or missile.

The reading of ALOCGRP or TMPALOC for missile groups takes place in subroutine MISASGN. For bomber groups, subroutine PRERAID reads common /STRKSUM/, which is a summary of the weapon allocation by penetration corridor. PRERAID makes a call on subroutine GENRAID to process each penetration corridor, and GENRAID reads from the ALOCGRP common /3/, which contains the data on all the targets assigned through the given penetration corridor.

OUTPUT FILE

The output file for POSTALOC is the STRKFILE, the format for which is shown in tables 32 and 33. Subroutine OUTSRT writes one record for each bomber sortie, describing the sortie plan and characteristics, and subroutine MISASGN writes the missile event plans.

The file contains one record for each bomber and missile flight plan generated. In the case of bombers which refuel, two records are present: the first for the primary, refueled plan and the second for the alternate plan to be used in the event of a refuel abort.

The end of data on the file is signalled by a dummy bomber record which has a group number of 201.

Table 32. STRKFILE Format (Missile Record) Written From Array EVTDATA

WORD	DESCRIPTION
1	Side
2	Command and control index
3	Group index
4	Time of launch
5	Payload index
6-8	Zero
9	Missile type
10	ICLASS=1
11	Launch region
12	Alert status
13-16	Zero
17	Number of missiles
18	Number of targets
19-36	Missile indices
37-54	Site indices
55-72	Target indices
73-90	Offset latitude
91-108	Offset longitude
109-126	Flight times in hours
127-144	Weapon site latitude
145-162	Weapon site longitude
163-180	Target latitude
181-198	Target longitude
199-216	Designator code of target
217-234	Task code of target
235-252	Country code of target
253-270	Flag code of target

Table 33. STRKFILE Format (Bomber Record)
Written From Common OUTSRT

WORD	DESCRIPTION
1	Sortie index
2	Group index
3	Corridor index
4	Vehicle index
5	Refuel index
6	Depenetration index
7	Payload index
8	Base index
9	Weapon type
10	Base latitude
11	Base longitude
12	Number of targets
13-22	Type of target
23-32	Latitude of target
33-42	Longitude of target
43-52	Latitude of weapon offset
53-62	Longitude of weapon offset
63-72	Index of target
73-82	Designator code (DESIG) of target
83-92	Task code of target
93-102	Country code of target
103-112	Flag of target

Table 33. (cont.)
(Sheet 2 of 2)

WORD	DESCRIPTION
113-122	Local attrition
123-132	Cumulative survival probability
133	Low-altitude range (precorridor legs)
134	Low-altitude range (before first target)
135	Low-altitude range (after first target)
136	Speed at low altitude
137	Speed at high altitude

CONCEPT OF OPERATION

The sortie definitions developed by POSTALOC are generated by weapon groups, one penetration corridor at a time. They consist of an ordered list of the targets to be struck by each sortie, a specification of which targets to strike with ASMs, and an estimate of the low-altitude range allotted for use before, versus after, the first target (and in any legs preceding the corridor origin). The sortie definition does not include the actual coordinates for the events. Thus for the bomber events it remains for PLNTPLAN to add these coordinates, calculate release points for ASMs, and compute time of entry into defense zones.

Figure 116 shows the relationship among the various major subroutines in the post-allocator. The arrows in the figure point from each subroutine to the subroutines it calls. Thus the arrows illustrate simply the calling sequence hierarchy.

The basic driver program POSTALOC, see figure 117, serves only to define the order in memory of all common blocks in the program. The actual processing begins with GETGROUP, which initializes all files, reads in the basic reference data, and then sets up the reference data for the first (or next) group to be processed (BASFILE). At this point, the processing splits. If the group is a missile group, a call is made on MISASGN which handles all the rest of the processing for the group. If the group is a bomber group, PRERAID is called. PRERAID then remains in control throughout the processing of the group.

When MISASGN is called, it reads ALOCGRP or TMPALOC to obtain all the strikes assigned to the group. These strikes are then assigned to specific missiles in the group. An effort is made to assign the strikes so that each base and each squadron in the group will have a fair share of both high and low priority strikes. The resulting assignments are then formatted as starting events for the Simulator, and appropriate launch times are assigned.

When PRERAID is called, the process is considerably more complex. Like MISASGN, PRERAID reads in the strikes assigned to the group. However, it reads them one corridor at a time; and after the strikes for a corridor have been read in, it calls GENRAID to process the raid in that corridor before proceeding to the next.

Thus at the level of GENRAID, the processing deals with a single raid consisting of aircraft from one group by way of not more than one corridor. It is useful to think of the remaining subroutines as being divided into two major sets:

462-546 O - 72 - 12

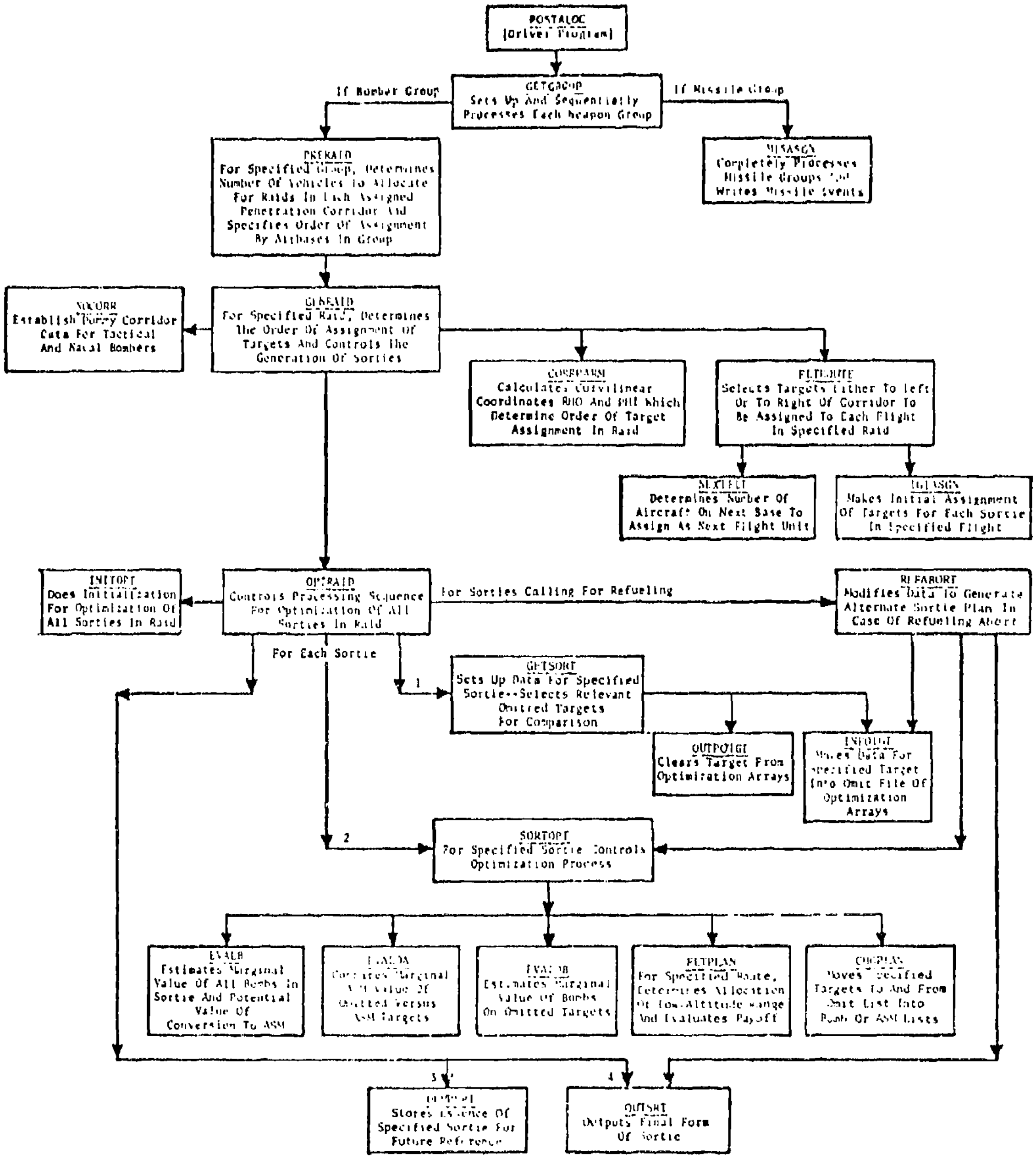


Fig. 116. POSTALOC Calling Sequence

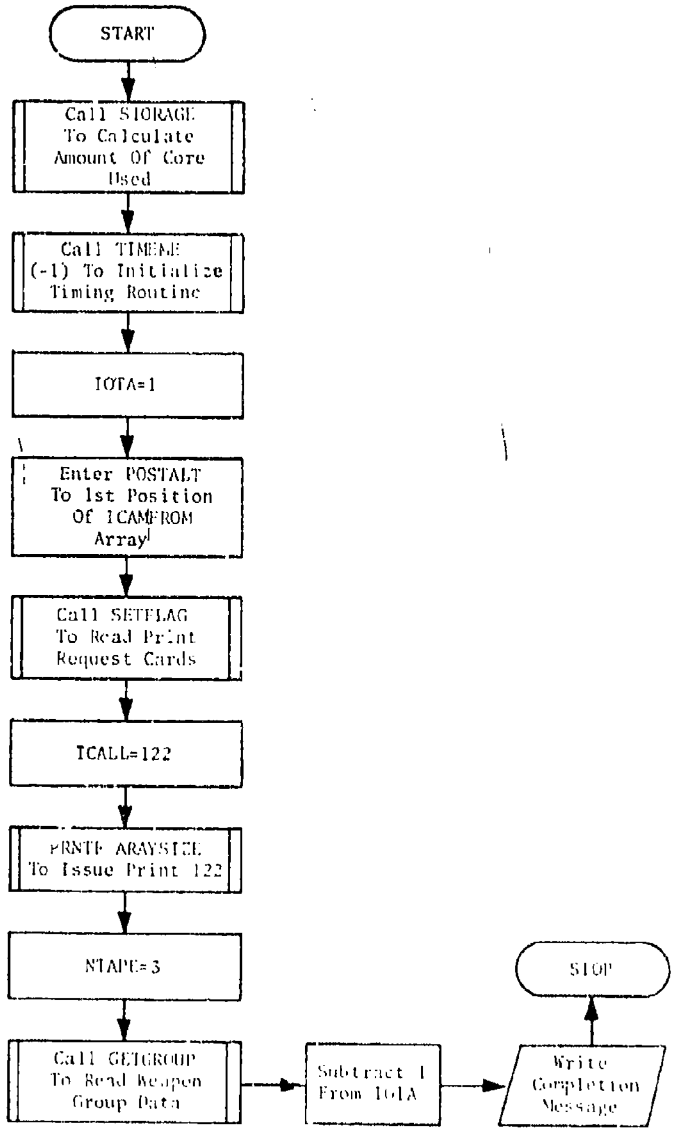


Fig. 117. Program POSTALOC

1. GENRAID and all the subroutines it calls directly or indirectly, with the exception of

2. OPTRAID and all the subroutines it calls directly or indirectly.

The first set of subroutines deals with the raid as a whole and is concerned with a rough division of the strikes in the raid among the available vehicles and bases.

Subroutines in the second set are concerned with one sortie at a time. They deal with each sortie in considerable detail, taking into account range, estimated attrition rates, low-altitude capability, and the option for use of ASMs or bombs on specific targets. During this process, provision is made to omit strikes that seem unprofitable. Each strike omitted may be picked up in processing later sorties, so that some refinement of the initial rough division of the strikes usually takes place in this phase.

From a design point of view, this division of POSTALOC was dictated by computer memory considerations. The computer memory is entirely adequate to deal with a complete representation of any single sortie, but would be totally inadequate to deal with such a representation of all sorties at once. Thus where one must deal with an entire raid, the sorties are represented in a very skeletal form. When any specific sortie is being processed in detail, the skeletal form is expanded and used to fill out a more detailed representation.

Perhaps the most important single consideration in the above approach is the problem of evaluating distances between targets. The relevant distances for a sortie are used over and over again in the optimization of the sortie. It would be too time-consuming to recompute each distance. Conversely, it is clearly impractical to provide space to store and retrieve all distances between all targets. In the QUICK system, space is provided to store all distances between all targets (and other route points) considered to be relevant to a single sortie. Each distance is computed by the function DIFF only once (the first time it is used). Thereafter it is simply retrieved from storage. However, such intertarget distances are retained only for the currently relevant set of route points.

The following discussion of POSTALOC is divided into the sections:

1. Raid generation

2. Setup for sortie optimization

3. Optimization of sorties

4. Development of missile plans

5. Program conventions for indexing and bookkeeping.

Raid Generation in POSTALOC

The ultimate control of processing by POSTALOC always resides with subroutine GETGROUP (see the calling sequence hierarchy, figure 116). However, this subroutine actually does nothing but control the sequencing from one weapon group to the next.

The significant processing of bomber sorties is done at the level of the raid, rather than at that of the group.* A secondary sequencing subroutine PRERAID has been supplied to sequence the processing between separate raids for the same group of bombers. PRERAID uses the strike summary information, supplied by ALOCOUT, to determine what share of the group's vehicles and warheads should be allocated to each raid. PRERAID also sorts the launch bases for the group on the computed distance from each base to the entry point of the corridor being processed. Thus, in the case of a retaliatory strike when launches are simultaneous, the vehicles available for the raid are processed in order of their time of arrival at the corridor.

In the case of tactical bombers or naval bombers (i.e., bombers assigned the attribute-value pair PKNAV > 0.0), a penetration corridor is not used. To preserve the logic of the program, a dummy corridor index is defined to indicate no corridor usage. This corridor index is tested before doing distance calculations, strike assignments, etc., so that the appropriate substitutions are made in the method of processing. In this case, time of arrival is no longer of obvious relevance to the order of processing. The launch bases for this case are sorted in the same way as the targets, as described below.

Program ALOC is set to assign a few extra strikes to each group of bombers in excess of the number of warheads available. The surplus assignment provides flexibility for sortie generation and assures the availability of targets for the last bomber in a raid, even when the strikes assigned to the corridor do not come out to an integral number of bombers. To a first approximation, the number of warheads assigned to each penetration corridor by PRERAID is proportional to the number of strikes assigned

---

*Actually, program ALOC tends to concentrate the weapons from each group in a small number of raids in the most efficient corridors for the group, but POSTALOC must be prepared to deal with cases in which a number of raids in different corridors are generated by the same group.

(by ALOC) in each corridor. However, if this number of warheads does not correspond to an integral number of delivery vehicles, the necessary additional warheads required to produce an integral number of delivery vehicles are assigned to each corridor as it is processed. Since the corridors are delivered for processing in order of decreasing number of strikes assigned, this rule puts a slightly higher ratio of bombers to targets in corridors with large raids. In this way, bombers assigned to corridors where there are few other bombers will have more flexibility to select from the geographically sparse target set assigned. In the extreme case where a corridor happens to have only one or two isolated strikes assigned, the corridor will probably be skipped in the assignment of bombers from the group, so that isolated individual bombers are less likely to be assigned to such a corridor.

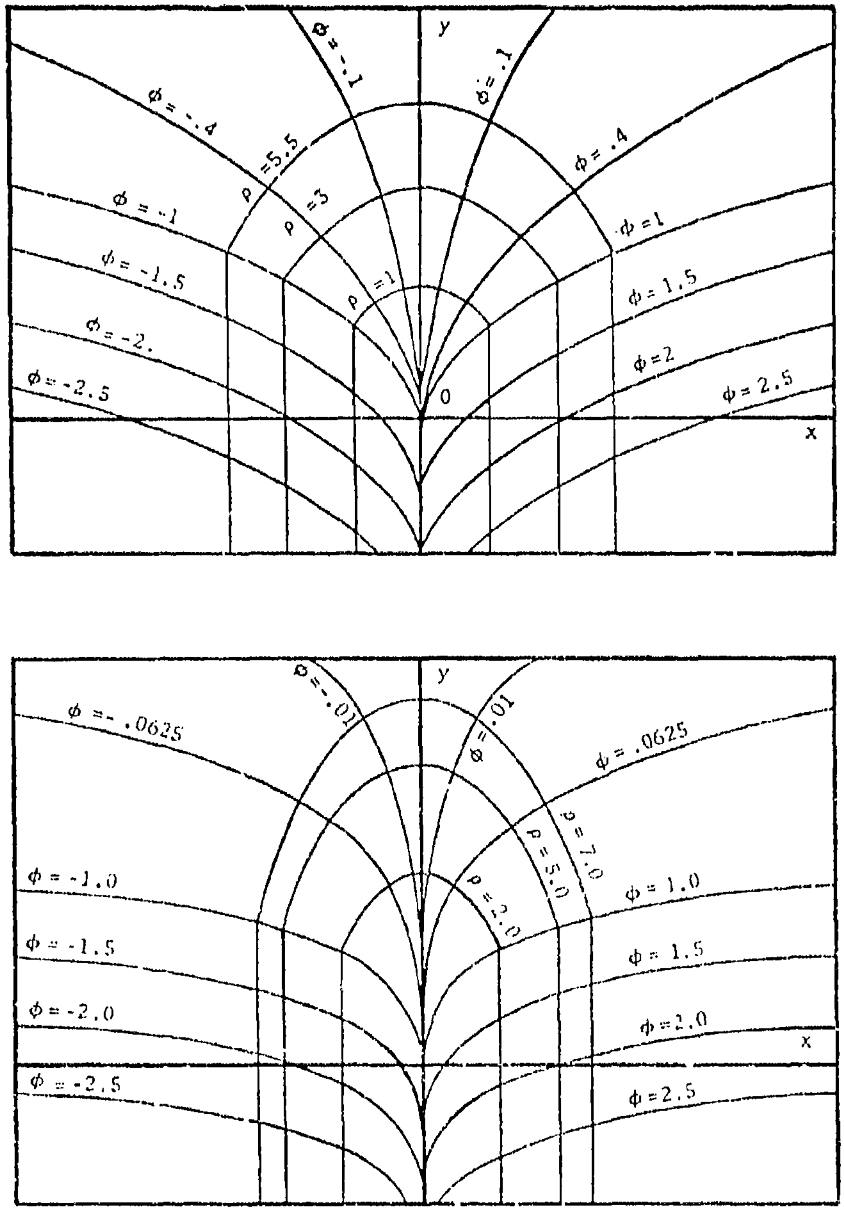
The next necessary task is to assign strikes within the raid to individual sorties. This requires the assignment of individual weapons to individual targets in accordance with the location of the targets relative to the penetration corridor. The assignment is accomplished through the use of curvilinear coordinate systems chosen to parallel typical flight paths within the penetration corridor.

Figure 118 illustrates two examples of the coordinate system employed in planning corridor penetrations. The coordinate system shown is established with the x=0, y=0 position corresponding to the last route point or origin of the penetration corridor. The Y axis is parallel to the corridor axis defined by the origin and the coordinates of the corridor point (or the head of the corridor arrow).

In the tactical or naval bomber case, the x=0, y=0 position corresponds to a point which is chosen as follows: centroids are computed for the group of launch bases and for the group of targets. Let the distance between these centroids be DISTC. The desired point of origin is the point on the line running through both centroids, at a distance DISTC from the base centroid, going away from the targets. This allows us to define a coordinate system on which we can locate the bases as well as the targets.

The ϕ (PHI) coordinates (lines of constant value of ϕ) are roughly parallel to the type of flight paths penetrating bombers should use. Thus, a single bomber should be assigned to targets that have roughly the same value of ϕ.

The two graphs shown correspond to different values of the parameter k, (known as KORSTYLE in the Fortran) and illustrate some of the flexibility provided. A high value of k is appropriate where saturation of defenses is desired, while a low value is appropriate if greater importance is attached to minimizing the distance to the targets. For tactical or naval bombers, for instance, k is set to 1.



(A) K=2

(B) K=4

Fig. 118. Illustrative Curvilinear Functions

GENRAID rearranges the strikes in order of increasing value of PHI. The rearrangement is accomplished by calling subroutine CORRPARM, which computes values of PHI and RHO for each target, and then calling ORDER and REORDER, which sort the strikes, together with their associated target data, on the value of PHI.

After the reordering is complete, the assignment of strikes is simple. Subroutine FLTROUTE calls subroutine NEXTFLT to determine the number of vehicles to be assigned to the next flight and calls subroutine TGTASGN to make the initial assignment of targets to vehicles in that flight. When a penetration corridor is used, FLTROUTE processes launch bases (previously sorted by PRERAID) in order of their distance from the corridor's entry point, so that the vehicles are processed in order of their time of arrival.

To provide an approximation of saturation and roll back tactics, each flight is assigned as a unit either to one side of the corridor or the other. The first flights are usually assigned to shallow targets (for which the absolute value of ϕ is high) while later flights are assigned to deeper targets (for which the absolute value of ϕ is low). Even if the density of strikes on the two sides of the corridor is quite different, the flights going to opposite sides are kept roughly in balance by comparing the value of ϕ before deciding to which side to assign the next flight. In order to maintain this balance, it is desirable to have at least five or six flights. Thus if there are four or fewer bases, two flights are sent from each base.

If there is no penetration corridor defined, the launch bases are processed in order of their absolute values of ϕ alternating from one side of the coordinate system to the other, in an attempt to make the sortie paths approximate as closely as possible the direction of the PHI lines.

Within each flight, strikes are assigned to one sortie at a time working through the list of unassigned strikes. Before any strike is assigned, however, it is checked against all strikes previously assigned to the sortie to be sure it would not duplicate a previously assigned target (where multiple strikes may be allocated to the same target). If such duplication would occur, the strike is skipped, and later strikes on the list are processed to get the specified quota for the sortie. Processing for the next sortie in the flight always begins with the first unassigned strike and continues from there. Strikes actually assigned to each sortie are always arranged in the sortie in order of increasing RHO. This gives the initial time order or sequence of the strikes which is used as a starting point for the optimization of the sortie.

The principal subroutines used for raid generation are: PRERAID, GENRAID, CORRPARM, FLTROUTE, NEXTFLT, TGTASGN, and NOCORR.

Setup for Sortie Optimization

Before describing the optimization of individual sorties, it is necessary to describe briefly the way the information is structured during the optimization.

During the optimization of a sortie, all targets relevant to the sortie are entered into a detailed computation array. This array (common /SORTYTGT/ -- targets for this sortie) includes not only the targets or strikes originally assigned to the sortie but also any targets omitted by prior sorties that may be relevant as target alternatives. The SORTYTGT arrays include not only the index to the targets in the basic target list, but also the coordinates, value, and defense characteristics for the targets together with temporary scratch-pad data, used in estimating the value of the sortie. Common /SORTYTGT/ has a capacity for 25 separate target entries.

The index positions in the SORTYTGT array that do not contain targets are said to be "available" and are listed in a file named "IAVAIL." The remaining positions in the SORTYTGT array will contain points of the present sortie listed in a file named "IHIT" and possible alternative targets listed in a file named "IOMIT." Actually, positions 1, 2, and 3 of the SORTYTGT array are reserved for nontarget points in the sortie. Position 1 (IORIG = 1) is used to represent the origin of the penetration corridor. Position 2 (IRECOVER = 2) is used to represent the recovery point. Position 3 (IDITCH = 3) is used to represent termination of the mission.

These conventions make it possible to define a sortie with a simple list of numbers. This sortie definition is contained in common block /CURSORTY/ (current sortie). The following table illustrates such a sortie definition.

	1	2	3	4	5	6	7	8
IFLY	1	7	7	8	5	5	2	3
IHIT	1	7	-9	8	-5	-4	2	3
IOMIT	6	13						
IAVAIL	10	11	12	15	14			

Illustrating Definition of Sortie (Common /CURSORTY/)

By convention, a negative number in the IHIT list indicates an ASM, and a positive number indicates a bomb. Thus the IFLY, IHIT table illustrated represents the following operations:

1. Leave origin of corridor

2. Bomb target listed in position 7

3. From the vicinity of 7 launch an ASM at 9

4. Bomb target listed in position 8

5. Fly near 5 to hit 5 with ASM

6. Strike 4 with ASM launched from vicinity of 5

7. Recover

8. End of mission.

The omit list indicates possible alternative targets listed in positions 6 and 13, while the avail list indicates five empty cells that could be used if needed.

Using this structure, the sortie can be modified at will simply by changing the sortie definition. Changes in the sortie -- for example, replacement of target 8 with 6 in the sortie definition -- would not require any rearrangement of these targets in the SORTYTGT array. Moreover, any distances between targets already computed for targets in the SORTYTGT array are still valid and do not even have to be re-indexed.

When the sortie is first set up by TGTASGN, all elements in the IHIT list are positive. Later decisions during optimization are necessary before some targets are flagged with the minus sign to be struck with ASMs.

The principal subroutine used when setting up for sortie optimization is OPTRAID.

Sortie Optimization

Sorties are optimized by a heuristic programming technique. For each sortie a value VALSORTY is calculated; the definition of VALSORTY is given in the description of subroutine FLTPLAN. All decisions on the modifications of the sortie definition are based on the estimated effect the changes will produce in the value of VALSORTY.

The basic controlling subroutine for the optimization is subroutine SORTOPT. On the first call of SORTOPT for any sortie, the initial sortie definition may not be feasible. It may require too many warheads; it may require too much range; or it may specify all bombs whereas the aircraft may carry ASMs. Thus the task of SORTOPT is to revise the sortie definition to produce a feasible sortie with the highest possible expected value of VALSORTY. To accomplish this, SORTOPT makes use of:

1. FLTPLAN - A subroutine which accepts any sortie definition, selects an optimal or near-optimal flight profile (low versus high altitude for legs of the mission), and then evaluates the expected value of the sortie, VALSORTY.

2. CHGPLAN - A subroutine which is called when changes in the sortie definition are required. CHGPLAN can be called to add or delete either a bomb or an ASM from the sortie definition. Specifically, it transfers targets between the hit and omit lists of the sortie definition, CURSORTY.

3. The EVAL routines, which estimate the probable change in the value of a sortie if changes are made in the sortie definition. The specific routines used are:

   - EVALB, which estimates the contribution of each bomb to the value of the sortie.

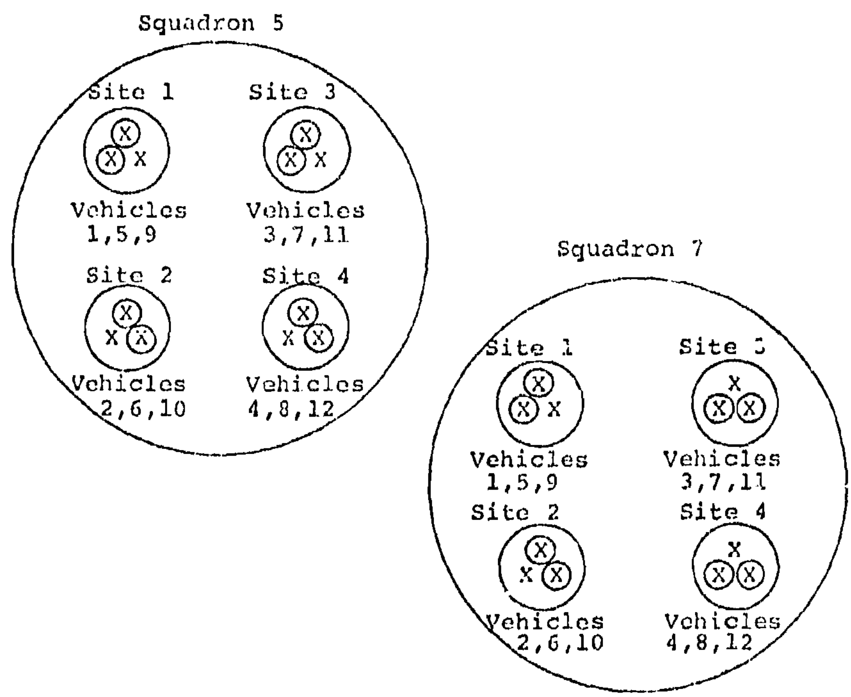
   - EVALOA, which estimates the contribution of each ASM now in the sortie and the potential contribution of each omitted target as a potential ASM target.

   - EVALOB, which estimates the potential contribution of each omitted target as a potential target for a bomb.

The estimated changes in VALSORTY by the EVAL routines are based on extrapolation of derivatives and thus are considered only as approximations which must be recomputed by FLTPLAN before they are accepted as final.

Development of Missile Plans

Subroutine MISASGN carries out the assignment of specific strikes to specific delivery vehicles within a weapon group. Figure 119 illustrates the structure of a typical group that MISASGN is designed to handle. The group may include several squadrons (two shown) and a squadron may include several sites (four per squadron shown). Each site may have one or more vehicles (three shown). Vehicles are considered to occupy the same site if they are so close together that they would have to be targeted as a



Ⓧ Vehicles in Group

X Vehicles not in Group

NOPERSQN = Total Vehicles in Squadron.
NBASE = Number of Bases (or Squadrons) in Group.
NWPSITE = Number of Weapons per Site.
ISTART = Lowest Vehicle Index in Group for Each Squadron.
NWPNS = Total Vehicles in Group.

Fig. 119. Configuration of Missiles in a Typical Group

single element target. For example, the Polaris squadron of 16 missiles on one submarine is considered to occupy one site, while the Minuteman squadron of 50 missiles occupies 50 separate sites.

On the other hand, any nonalert missiles in a squadron will constitute a separate weapon group. Since the vehicle indices within a squadron may not start from one, the starting vehicle index ISTART for each squadron is supplied as an input to the missile assignment phase. This and the other input parameters defining the available weapons for the program are shown in the figure. The strikes assigned to the group by program ALOC are placed in order by decreasing values of RVAL. The strikes are then assigned in this order beginning with the vehicle index ISTART for the first squadron. The next strike goes to the next squadron until all squadrons have one strike assigned. Then the vehicle index is incremented, and strikes are again allocated to all squadrons until all weapons are used.

For efficiency in the Simulator, all the missile launch operations from a single squadron are packed into one Simulator event -- unless the capacity of 18 strikes per event would be exceeded. In this case, more than one event is required for each squadron.

A description of MIRV processing is given under Subroutine MISASGN.

Program Conventions for Indexing and Bookkeeping

The list of targets and associated data are input to POSTALOC from the ALOCGRP file, which is the output from program FOOTPRNT, or from the TMPALOC file, which is the non-MIRV output from program ALOCOUT.

When read in, the targets are in a meaningless order for the purpose of POSTALOC. CORRPARM is called by GENRAID to compute the curvilinear coordinates RHO and PHI for each target (as discussed in the earlier section on Raid Generation in POSTALOC). The target data in common /3/ (elsewhere referred to as the RAIDSTRK arrays) are then sorted and reordered by ascending value of PHIs. Throughout POSTALOC, it is the position of the target in these arrays, to be referred to as the RAIDSTRK index, which is used to identify the target, rather than INDEXNO.

It is the function of TGTASGN, after processing has begun for a given corridor of a given group, to divide up the entire list of targets in the RAIDSTRK list among the sorties assigned to that corridor. It does this by assigning the closest targets to the earliest flights, alternating sides of the corridor, and working back to the farthest targets for the last flights. This is accomplished by taking targets from both ends of the list and working towards the middle. The RAIDSTRK indices for the targets are placed in a doubly dimensioned array known as the JTGT array in common /CURRAID/. This array permits up to ten targets to be assigned to each sortie.

There will occasionally be more targets initially assigned to a sortie in this array than there are warheads on the vehicle. This is because the number of weapons allocated is always slightly larger than it should be, to allow for some flexibility in the sortie plans produced by POSTALOC.

The array MYASGN (common /CURRAID/), with dimensions and indexing corresponding to the RAIDSTRK arrays (common /3/), gives the assignment of each target. (Because of its dimensions, MYASGN is included in the RAIDSTRK debugging print rather than the CURRAID print.) The values that MYASGN may assume are as follows:

- -1 Initial value; not yet looked at by TGTASGN
- 0 Looked at by TGTASGN; not assigned to sortie
- 1 Assigned to sortie; not currently in SORTYTGT array
- 2 Assigned to sortie; currently in SORTYTGT arrays.

When the individual sortie processing begins, GETSORT moves the target assigned to the sortie into a "potential target" array. It does this by calling INPOTGT for each target it wants brought in. INPOTGT enters the RAIDSTRK index of the target in the first available cell of MYPOTGT, which is a list of targets to be considered for this sortie.

Array IAVAIL in common /CURSORTY/ supplies the next available index of the MYPOTGT array at any time. This array originally contains all the indices of MYPOTGT (i.e., 1 through MAXPT, the maximum number of potential targets, stored in reverse order).

NAVAIL, the number of available spaces, is originally set to MAXPT. Each time a target enters or leaves the MYPOTGT array, NAVAIL is incremented or decremented respectively. Thus, at any time, IAVAIL (NAVAIL) contains the next available cell of MYPOTGT.

After the RAIDSTRK index has been entered in the MYPOTGT array the target is identified by position in the MYPOTGT array; this position will be called the SORTYTGT index.

Each target in the MYPOTGT list is also in either the IHIT or the IOMIT array, hereafter referred to as the hit or omit lists. After entering a target in the MYPOTGT list, INPOTGT enters its SORTYTGT index in the omit list. There is a variable NOMIT which contains the number of targets in the omit list at any time.

An additional array, LKHITMT, provides a cross-reference between indices of the MYPOTGT array and the hit and omit lists. For each cell of the MYPOTGT array containing a target (i.e., each cell not in the IAVAIL array), the corresponding cell of the LKHITMT array contains the index of the target in the hit or omit list. If the target is in the hit list, this number is positive; if in the omit list, it is negative.

The first cell of the hit list always contains 1, representing the origin of the corridor (as the first route point). The last filled cell always contains 3, representing the point at which the bomber will land. The next-to-last cell may or may not contain a 2, depending on whether or not recovery is planned. If recovery is not planned, this 2 will have been moved to the omit list. It is only in this event that the last cell (containing 3) becomes significant, since it indicates landing without recovering, or ditching. Hereafter, the 3 will be referred to as the ditch point. Unlike the recovery point, the origin and ditch points are always in the hit list, never in the omit list.

After INPOTGT has brought the target into the MYPOTGT array and the omit list, GETSORT calls CHGPLAN to move it into the hit list. CHGPLAN may be called with any of the following options: OTB, to move a target from the omit list to the hit list as a bomb target; OTA, to move a target from the omit list to the hit list as an ASM target; BTO, to move a bomb target from the hit list to the omit list; ATO, to move an ASM target from the hit list to the omit list.

If the target is to be hit with an ASM, its SORTYTGT index in the hit list will be negative, as opposed to being positive for a bomb. All targets are inserted after the origin (1), and before the recovery (2) or ditch point (3). The targets are stored in this array in the order that the vehicle expects to fly.

There is a companion array to the hit list called IFLY. This array contains the SORTYTGT index of the "fly point" at which the weapon is released. If the weapon is a bomb, the fly point is the target itself. If the weapon is an ASM, the fly point is the earliest bomb target which is within range of the ASM target. (This rule is sufficient for calculations and bookkeeping in POSTALOC. PLNTPLAN actually picks the optimal launch point for the ASM later.) If there is no bomb target within range, the bomber will have to fly within range of the target. In this event, the target itself is used as its fly point.

There is one other array used in the target bookkeeping. This is the LOSTTGT (lost target) array. For each sortie, GETSORT searches the range surrounding the targets assigned, in the MYASGN array, to see if there were any targets which were rejected or dropped (i.e., if MYASGN = 0 for any targets). If so, that target is brought into the LOSTTGT array and

MYASGN is set to 1. Then if space remains, as many lost targets as there is room for are brought into the potential target arrays.

As each sortie is processed, half of the targets remaining in the omit list from the last sortie are dropped by calling OUTPOTGT. The least valuable targets are selected and OUTPOTGT removes each target from the omit list, puts its SORTYTGT index back into the IAVAIL list, and sets MYASGN = 0.

COMMON BLOCK DEFINITION

External Common Blocks

The common blocks used by program POSTALOC in processing input/output (I/O) files are shown in table 34.

Since the input missile record does not use all of common block /3/, subroutine MISASGN redefines that common block and stores its output record there (in array EVTDATA). In that subroutine, all elements of this array EVTDATA after the 18th are equivalenced as shown in table 35.

Internal Common Blocks

In addition to the common blocks associated with I/O operations, the common blocks described in table 36 are used internally by program POSTALOC.

Table 34. Program POSTALOC External Common Blocks
(Sheet 1 of 9)

INPUT DATA FROM BASFILE

BLOCK**	VARIABLE OR ARRAY*	DESCRIPTION
/ASMTABLE/		ASM characteristics
	IWHDASM(20)	Warhead index } for ASMs
	RANGEASM(20)	Range } for ASMs
	RELASM(20)	Reliability } for ASMs
	CEPASM(20)	CEP } for ASMs
	SPEEDASM(20)	Speed } for ASMs
/CORRCHAR/		Corridor characteristics
	PCLAT(30)	Latitude of corridor point
	PCLONG(30)	Longitude of corridor point
	PCZONE(30)	Defense zone in which corridor origin is located
	RPLAT(30)	Latitude of corridor origin
	RPLONG(30)	Longitude of corridor origin
	ENTLAT(30)	Latitude of corridor entry
	ENTLONG(30)	Longitude of corridor entry
	CRLENGTH(30)	Distance from corridor entry to corridor origin
	KORSTYLE(30)	Parameter to adjust mode of corridor penetration
	ATTRCORR(30)	High-altitude attrition per nautical mile unsuppressed
	ATTRSUPP(30)	High-altitude attrition per nautical mile suppressed
	HILOATTR(30)	Ratio low- to high-altitude attrition (less than 1)

*Parenthetical values indicate array dimensions. All other elements are single word variables.
**Ordered alphabetically, not by position in core.

462-546 O - 72 - 13

Table 34. (cont.)
(Sheet 2 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/CORRCHAR/ (cont.)	DEFRANGE(30)	Characteristic range of corridor defense (nautical miles)
	NPRCRDEF(30)	Number of attrition sections this corridor
	DEFDIST(30,3)	Distance of each precorridor leg
	ATTRPRE(30,3)	Attrition in each precorridor leg
	NDATA	Number of words in common /CORRCHAR/
/DPENREF/		Depenetration and refuel points
	DPLINK(50)	Depenetration point link
	DPLAT(50)	Depenetration point latitude
	DPLONG(50)	Depenetration point longitude
	REFLAT(20)	Refuel point latitude
	REFLONG(20)	Refuel point longitude
/FILES/		Logical unit number and maximum length for all Plan Generator files
	TGTFILE(2)*	Target data file
	BASFILE(2)	Data base information file
	MSLTIME(2)	Fixed missile timing file
	ALOCTAR(2)	Weapon allocation by targets file
	TMPALOC(2)	Temporary allocation file
	ALOCGRP(2)	Allocation by group file
	STRKFIL(2)	Strike file

*In two-word arrays, first word is logical unit number; second word is maximum file length in words. Single variables are logical unit numbers.

Table 34. (cont.)
(Sheet 3 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/FILES/ (cont.)	EVENTAPE *	Simulator events tape
	PLANTAPE *	Detailed plans tape
/GRPDATA/		Characteristics of weapon groups
	IGROUP	Group number
	NWPNS	Number of weapons
	NVEHGRP	Number of vehicles
	IREG	Region
	ITYPE	Weapon type
	IALERT	Alert status
	IREFUEL	Refuel code
	YIELD	Yield
	ISTART	Starting weapon index
	NBASE	Number of bases
	IBASE(150)	Base index number
	BLAT(150)	Base latitude
	BLONG(150)	Base longitude
	IPAYLOAD(150)	Payload index
	VONBASE(150)	Number on base
/GRPTYPE/		Characteristics of weapon types
	ISIMTYPE	Hollerith type name
	RANGE	Range
	CEP	CEP
	SPEED	Speed
	ALERTDLY	Alert delay
	NALRTDLY	Nonalert delay
	RANGEDEC	High/low altitude fuel consumption ratio

*These files are output on magnetic tape.

Table 34. (cont.)
(Sheet 4 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/GRPTYPE/ (cont.)	ICLASS	Weapon class
	NOPERSQN	Number per squadron
	SPDHI	High-altitude speed
	SPDLO	Low-altitude speed
	SPDASH	Dash speed
	RANGREF	Refueled range
	NMPSITE	Number per site
	IREP	Reprogramming index
	IRECMODE	Recovery mode
	IPENMODE	Penetration mode
	FUNCTION	Function code
/MASTER/		Run ID, and quantity of QUICK entities
	IHDATE	Date of run initiation
	IDENTNO	Run identification number
	ISIDEM	Attacking side
	NRTPT	Number of route points
	NCORRM	Number of penetration corridors
	NDPEN	Number of depenetration corridors
	NRECOVER	Number of recovery bases
	NREF	Number of directed refuel areas
	NBNDRY	Number of boundary points
	NREG	Number of command and control regions
	NTYPE	Number of weapon types
	NGROUP	Number of weapon groups
	NTOTBASE	Total number of bases
	NPAYLOAD	Number of payload types
	NASMTYPE	Number of ASM types
	NWHDTYPE	Number of warhead types

Table 34. (cont.)
(Sheet 5 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/MASTER/ (cont.)	NTANKBAS	Number of tanker bases
	NCOMPLEX	Number of complex targets
	NCLASS	Number of weapon classes (presently two)
	NALERT	Number of alert conditions (presently two)
	NTGTS	Number of targets
	NCORTYPE	Number of penetration corridor types
	NCNTRY	Number of distinct country codes
/PAYLOAD/		Payload description tables
	NOBOMB1(40)	Number of type 1 bombs
	IWHD1(40)	Type 1 warhead index
	NOBOMB2(40)	Number of type 2 bombs
	IWHD2(40)	Type 2 warhead index
	NASM(40)	Number of ASMs
	IASM(40)	ASM index
	NCM(40)	Number of countermeasures
	NDECOYS(40)	Number of terminal decoys
	NADECOYS(40)	Number of area decoys
	IMIRV(40)	MIRV system identification number
/PLANTYPE/		Type of plan, and coordination parameters
	INITSTRK	Indicator for first or second strike
	CORMSL	Coordination time parameter for missiles
	CORBOMB	Coordination distance for bombers

Table 34. (cont.)
(Sheet 6 of 9)

INPUT DATA FROM ALOCGRP FILE

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/STRKSUM/	KGROUP	Group number
	NTSTRK	Total number of strikes for this group
	NCORR	Number of corridors for this group (=1)
	NSTRK(30)	Number of strikes assigned to each corridor
/FIXALL/	IJFIX(1100) IJFIXR(1100) IKFIX(25)	Logical data for bombers indicating fixed weapon assignment
/3/*	NT	Number of strikes in corridor
	JGROUP	Group index number
	JCORR	Corridor index number
	INDEXNO(1100)	Target index numbers
	TLAT(1100)	Target latitudes
	TLONG(1100)	Target longitudes
	TIMEPREM(1100)	"COMPLEXD" target indicators
	IDEPEN(1100)	Depenetration corridor indices
	DISTOUT(1100)	Distances from targets to depenetration corridors
	DISTREC(1100)	Distances from targets to recovery points
	ATTRLOC(1100)	Local target defense potentials
	RVAL(1100)	Relative values of targets
	DELAT(1100)	Target offset latitudes
	DELONG(1100)	Target offset longitudes

*As used when processing a bomber record

Table 34. (cont.)
(Sheet 7 of 9)

BLOCK		VARIABLE OR ARRAY	DESCRIPTION
/3/* (cont.)		DESIG(1100)	Target designator codes
		CNTRYLOC(1100)	Target country location codes
		FLAG(1100)	Flag codes for targets
/3/**	Input record from ALOCGRP or TMPALOC file	NT	Total number of targets assigned to group
		JGROUP	Group number
		JCORR	Corridor number (=0)
		INDEXNO(1100)	Index numbers of targets (negative if first target assigned to booster)
		TLAT(1100)	Target latitude (degrees)
		TLONG(1100)	Target longitude (degrees)
		INTOT(1100)	Not used
		RVAL(1100)	Relative value of strike
		DLAT(1100)	Offset latitude (degrees)
		DLONG(1100)	Offset longitude (degrees)
		DESIG(1100)	Target designator code
		TASK(1100)	Target task code
		CNTRYLOC(1100)	Target country location code
		FLAG(1100)	Target flag code
		FTIME(18,50)	Flight time matrix
		EVTDATA(270)	Missile record as output to STRKFILE (see discussion of STRKFILE output for redefinition of this array)
		DUM(3230)	Unused

*As used when processing a bomber record
**As used when processing missile records

Table 34. (cont.)
(Sheet 8 of 9)

OUTPUT DATA FOR STRKFILE*

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/OUTSRT/	IOUTSRT	Sortie index
	MYGROUP	Group index
	MYCORR	Corridor index
	INDVEH	Vehicle index
	JREF	Refuel index
	JDPEN	Depenetration index
	KPAYLOAD	Payload index
	LNCHBASE	Base index
	ITYP	Weapon type
	BASELAT	Base latitude
	BASELONG	Base longitude
	NHAP	Number of targets
	HAPTYPE(10)	Type of target
	OBLAT(10)	Latitude of target
	OBLONG(10)	Longitude of target
	DLAT(10)	Latitude of weapon offset
	DLONG(10)	Longitude of weapon offset
	IOBJECT(10)	Index of target
	DSIG(10)	Designator number of target
	TSK(10)	Task number of target
	CNTRLC(10)	Country code of target
	FLG(10)	Flag of target
	ATTROUT(10)	Local attrition
	SURVOUT(10)	Cumulative survival probability

*The bomber records only are written from common block /OUTSRT/; the missiles are handled separately.

Table 34. (cont.)
(Sheet 9 of 9)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/OUTSRT/ (cont.)	DSTLOW1	Low-altitude range (precorridor legs)
	DSTLOW2	Low-altitude range (before first target)
	DSTLOW3	Low-altitude range (after first target)
	SPDLOW	Speed at low altitude
	SPDHIGH	Speed at high altitude
	RANGEX	Range of vehicle without refueling
	RANGEREF	Range of vehicle with refueling
	DELAY	Delay before takeoff
	IRG	Regional index
	ILRT	Alert status
	IDBOMBER	Bomber identification
	AVLOW	Available low-altitude range
	RNGDC	Range decrement at low altitude

Table 35. Format of Array EVTDATA in Common Block /3/ as Used by Subroutine MISASGN (the Missile Record to be Output to STRKFILE) (Sheet 1 of 2)

WORD OF EVTDATA	DESCRIPTION	EQUIVALENCED TO:*
1	Side	None
2	Command and control index	None
3	Group index	None
4	Time of launch	None
5	Payload index	None
6-8	Zero	None
9	Missile type	None
10	ICLASS=1	None
11	Launch region	None
12	Alert status	None
13-16	Zero	None
17	Number of missiles	None
18	Number of targets	None
19-36	Missile indices	KMISL(18)
37-54	Site indices	KSITE(18)
55-72	Target indices	KTGTIND(18)
73-90	Offset latitude	XDLAT(18)
91-108	Offset longitude	XDLONG(18)
109-126	Flight times in hours	FLTIME(18)
127-144	Weapon site latitude	WLAT(18)
145-162	Weapon site longitude	WLONG(18)
163-180	Target latitude	XTLAT(18)
181-198	Target longitude	XTLONG(18)
199-216	Designator code of target	KDESIG(18)

*Parenthetical values indicate array dimensions. All other elements are single word variables.

Table 35. (cont.)
(Sheet 2 of 2)

WORD OF EVTDATA	DESCRIPTION	EQUIVALENCED TO:
217-234	Task code of target	KTASK(18)
235-252	Country code of target	KCNTRYLC(18)
253-270	Flag code of target	KFLAG(18)

Table 36. Program POSTALOC Internal Common Blocks
(Sheet 1 of 13)

BLOCK	VARIABLE OR ARRAY*	DESCRIPTION
/ARAYSIZE/	MBASEPG	Maximum number of bases (or squadrons) per group (150)
	MC	Maximum number of corridors (30)
	MT	Maximum number of targets per group (1,100)
	ML	Maximum separate defended zones pre-entry (three)
	MSTRK	Maximum strikes per sortie (10)
	MSORTY	Maximum sorties per group (100)
	MSRT	Maximum sorties per group per corridor (100)
	MFLY	Maximum number of points in JHIT and IFLY lists (13)
	MAXPA	Maximum number of points allowed by array size (25)
/CHGPLAN/		(In effect, part of calling sequence for subroutine CHGPLAN)
	JDO	SORTYTGT index for target to be added or deleted by CHGPLAN
	IAIM	SORTYTGT index to flight point for ASM launch
	ISCAN	Controls number of sortie points scanned by EVAL routines
	JAFT	SORTYTGT index to target to precede insertion
	ATO, OTA, BTO, OTB	Calling parameters for CHGPLAN
/CONTROL/	EPSILON	Set to 1001; used in tests of significance

*Parenthetical values indicate array dimensions. All other elements are single word variables.

Table 36. (cont.)
(Sheet 2 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/CONTROL/ (cont.)	KWALSRT	Set to 1; not currently used
/CORRIDOR/	CLAT	Latitude of corridor oridntation point
	CLONG	Longitude of corridor orientation point
	CZONE	Defense zone for corridor origin
	PLAT	Latitude of corridor origin (route point)
	PLONG	Longitude of corridor origin (route point)
	ELAT	Latitude of corridor entry point
	ELONG	Longitude of corridor entry point
	CLENGTH	Distance from entry to corridor origin
	KORPWR	Power of Y vs. X in calculation of PHI
	CORATTR	Attrition of corridor with unsuppressed defenses at high altitude
	CORSATTR	Attrition of corridor at high altitude with defenses suppressed
	ATTRHILO	Ratio of low- to high-altitude attrition
	RNGDEF	Characteristic range of defense operations
	NPREDEF	Number of separate defended zones prior to corridor
	DISTDEF(3)	Length of Ith defended zone
	PREATTR(3)	Attrition in Ith defended zone
/CURRAID/	MYBASE(100)	Base index assigned to sortie
	NASGN(100)	Number of targets assigned to sortie

Table 36. (cont.)
(Sheet 3 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/CURRAID/ (cont.)	ITGT(ISTRK,ISRT) with ISTRK = 10 ISRT = 100	Index (IT) to target for ISTRKth strike in ISRTth sortie in present raid; negative for ASMs
	MYASGN(1100)	Assignment status of target
	INDEXVEH(100)	Vehicle index assigned to sortie
/CURSORTY/	NUMHIT	Total targets hit by current sortie plan
	NUMBOMB	Number of targets bombed
	NUMASM	Number of targets hit with ASMs
	NHIT	Number of sortie points -- NUMHIT + (Origin, Recovery, Ditch)
	IFLY(13)	SORTYTGT index to Ith route point or target
	IHIT(13)	SORTYTGT index to Ith flight point -- not the same as IHIT for ASMs
	LASTPAY	Sortie position of last paying sortie point, pre-ditch
	LASTTGT	Sortie position (=NUMHIT + 1) for last target
	NOMIT	Number of targets in potential target array not currently in sortie
	IOMIT(25)	SORTYTGT index of Ith omitted target
	NAVAIL	Number of spaces in potential target arrays available for new targets
	IAVAIL(25)	SORTYTGT index of Ith available space
	LHITMT(25)	Link for target J to sortie hit list if positive; omit list if negative
	IOLD	Pointer used in finding lost targets

Table 36. (cont.)
(Sheet 4 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/CURSORTY/ (cont.)	LOSTTGT(25)	SORTYTGT index to Ith lost target
	NLOSTTGT	Number of lost targets
	NWHDS	Number of warheads carried on this sortie
	NASMS	Number of ASMs among warheads on this sortie
	RNGASM	Range of ASMs this sortie
	DSTB	DISTB(IB) - distance from target to entry point
/DATA/	IPRINTNO(60)	Print request number
	IFSTSORT(60)	First sortie to activate print
	LSTSORT(60)	Last sortie to activate print
	LPASS(60)	Pass on which print is active (1 or 2)
	LCORR(60)	Penetration corridor on which print is active
	LGROUP(60)	Weapon group on which print is active
/DEBUG/	IOTA	Index to cell containing Hollerith name of subroutine currently in control
	ICAMFROM(20)	Array containing Hollerith subroutine names in order of calling hierarchy
/EVAL/	MINB	The lowest payoff for a bomb in the sortie, found by EVALB
	JDELB	The SORTYTGT index of the bomb with lowest payoff, MINB
	MAXDA	The maximum payoff increment by using ASM on omitted targets
	JADD	The SORTYTGT index of the target with maximum increment MAXDA

Table 36. (cont.)
(Sheet 5 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/EVAL/ (cont.)	MINDA	The minimum payoff for an ASM in the sortie
	JDEL	The SORTYTGT index of the target for ASM with minimum payoff, MINDA
	MAXOB	The maximum payoff increment for a bomb on an omitted target
	JADDB	The SORTYTGT index of the target showing maximum payoff, MAXOB
	MAXDAB	Maximum payoff increment obtainable by use of ASM instead of bomb
	JADDA	The SORTYTGT index of bomb target showing maximum payoff increment, MAXDAB
	VALSORTY	Estimated total sortie value (=VALDONE(LASTPAY))
	JAF	Index of target to precede insertion
	KALC	Signal to EVAL routines to skip repetition of calculations
	VALDIST	"Value" per unit distance of extra range as calculated by FLTPLAN
	VALMAX	Maximum possible value of sortie as currently defined
	JSEQERR	Index of target in sortie showing largest sequence error
	ISFINPLN	Not used
/FIXRANGE/	CENTLAT, CENTLONG	Latitude and longitude of centroid of launch bases in group
	DISTC	Distance between centroid of launch bases in group and corridor entry point
/FLAG/	IFLAG(200)	Set to 1 if Ith print is active; 0 if not

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/FLTPASS/	IFPASS	Index of sortie processing pass (limited to 100 sorties per pass)
	JVEHLO	Low-vehicle number, this pass
	JVEHHI	High-vehicle number, this pass
	NVEHPASS	Number of sorties this pass (equals 100, or all remaining sorties)
	NPASS	Total number of passes necessary
/IDUMP/	IDMPG	Group on which to ABORT
	IDMPC	Corridor on which to ABORT
	IDMPS	Sortie on which to ABORT
	IDMPP	Abort on primary (1) or alternate (2) plan
	IPLAN	Indicates whether primary or alternate in process
/INITOPT/	FLYLOW(3)	Low-altitude distance flown in Ith precorridor leg
	IMPORTD(3)	Order of importance of attrition per mile in Ith precorridor leg
	ATPDIST(3)	Average attrition per distance in Ith leg
	DISTINK	Distance from refuel point or entry point to corridor origin
	TDEFDIST	Total precorridor defended distance
	RNGE	Range of vehicles in this group (=RANGE or RANGREF)
/INDEX/	ISORTY	Index to sortie currently being processed
	IORIG	SORTYTGT index for corridor origin (=1)
	IRECOVER	SORTYTGT index for recovery point (=2)

462-546 O - 72 - 14

Table 36. (cont.)
(Sheet 7 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/INDEX/ (cont.)	IDITCH	SORTYTGT index for ditch point (=3)
	IT	Target index
	ICORR	Corridor index
	JTGTIN	SORTYTGT index of the latest target brought in by INPOTGT
/INPUTFL/	INPTFL	Logical unit number for TMPALOC or ALOCGRP file
	MYIDINFL	ID for TMPALOC or ALOCGRP file
/IRESRCH/	IRESRCH	Flag for subroutine TGTASGN
/ISKIPTO/	ISKIPTO	Group number with which POSTALOC processing is to begin
/KEYS/	KEYSTART	Keyword for retrieving ISTART
	KEYVBASE	Keyword for retrieving number of vehicles/base
/MISPRNT/	LK	Squadron number for current event
	LL	Current event number
	NEVTS	Number of events to be generated
/NEXTFLT/	NTAILS	Number of vehicles assigned to next flight
	JB	Index of launch base of next flight
	SPLIT	Equals 1 if less than 4 bases in group; otherwise =0 (if 1, causes each base to "split"; i.e., send flights down both sides of corridor)
	KB	Marker for side 2
	LB	Marker for side 1
/PCALL/	IPASS	OPTRAID pass in process (1 or 2)
	ICALL	Print request number

Table 36. (cont.)
(Sheet 8 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/POLITE/		Parameters for interpolation routines
	S1	Latitude of first point
	T1	Longitude of first point
	S2	Latitude of second point
	T2	Longitude of second point
	FACTOR	Fraction of distances to be interpolated
	SR	Result: latitude of interpolated point
	TR	Result: longitude of interpolated point
/PRNTF/	ATLEGHI	Not used
	ATLEG	Attrition on leg
	RNGSURP	Range surplus when whole flight is high altitude
	DISTANCE	Total length of current flight route
	RSVLOW	Amount of incremental range reserved for low altitude prior to the first target
	AVAILOW	Total amount of incremental low-altitude range (i.e., surplus high-altitude range converted to the number of miles of potential low altitude)
	VALIT	Value of sortie remaining, after first target has been reached
	FSTATTR	Attrition rate at point where sortie goes low on leg to first target, or highest rate if sortie does not go low
	VALTOTT	Total value of sortie

Table 36. (cont.)
(Sheet 9 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/PRNTF/ (cont.)	PRATRATE	Attrition rate in current pre-corridor leg
	VALONT	Value of sortie from current leg (after first target) to recovery
	CURVAL	Highest product of value-on and attrition of areas competing for low altitude; used in computing value per unit distance of low-altitude range
	CRITATT	Higher of the two products of value-on and attrition for pre-corridor legs and legs after first target; used in determining the amount of low-aititude range (ADDLOW) to be put in the first-target leg
	ADDLOW	Amount of low-altitude range currently being allocated
	JHONE	SORTYTGT index of first target in sortie
	DISTSV	Distance saved by omitting a target JH
	ATAREA	Difference in area attrition if a target JH is omitted
	ATLOCAL	Local attrition of target JH
	VALO	Value of omitting target JH, due to decreased attrition and range saved
	JX2	Index used in EVALOB for determining position of insertion for new target
	DISTAD	Distance added by inserting target JX in normal position
	ADDTEST	Distance added by inserting target JX in alternative position

Table 36. (cont.)
(Sheet 10 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/PRNTF/ (cont.)	ATTRNEW	Attrition added by inserting target JX in sortie
	ATNEWLG	Estimated attrition on leg to new target JX being inserted
	DVALO	Differential value of leaving new target in the omit list rather than inserting it
	JX1	Index used in EVALOB for determining position of insertion for new target
	DIST1	Length of leg preceding target JH
	DIST2	Length of leg from target JH to next target
/PRINTOPT/		(Options used as calling parameters for subroutine PRINTIT to produce the indicated prints)
	ISRTYTGT	Common /SORTYTGT/
	ICURSRTY	Common /CURSORTY/
	ICURRAID	Common /CURRAID/
	IRSTKCPM	Commons /RAIDSTRK/ and /2/(CORPARM)
	IEVAL	Common /EVAL/
	IRAIDSHR	Common /RAIDSHR/
	ICHGPLAN	Common /CHGPLAN/
	IINITOPT	Common /INITOPT/
	IINDEX	Common /INDEX/
	ITGTASGN	Common /TGTASGN/
	IGRPTYPE	Common /GRPTYPE/
	IGRPDATA	Common /GRPDATA/
	ICORCAR	Common /CORRCHAR/
	ISTRKSUM	Common /STRKSUM/
	ICORRSHR	Common /RAIDSHR/
	ICORIDOR	Common /CORRIDOR/

Table 36. (cont.)
(Sheet 11 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/PRINTOPT/ (cont.)	INEXTFLT	Common /NEXTFLT/
	IDEBUG	Common /DEBUG/
	JREFUEL	Common /REFUEL/
	JOUTSRT	Common /OUTSRT/
	JPAYLOAD	Common /PAYLOAD/
/RAIDSHR/	NVEH	Number of vehicles from current weapon group assigned to raid in this corridor
	NRVEH(200)	Number of vehicles in group on Ith base which are still unassigned
	DISTB(200)	Relative flight distance from Ith base to corridor entry
	NB	Number of bases in group
	VPRBASE	Average number of vehicles per base
	NWPV(200)	Number of warheads per vehicle on Ith base
	NWPC	Total number of warheads to enter by this corridor
	TGTSPWHD	Targets per warhead (=NT/NWPC)
	NASMPV(200)	Number of ASMs per vehicle on Ith base
/RUNCHECK/	RUNCHECK	VALSORTY accumulator
/SKIP/	KOLD	Contains Hollerith IDENT of last call on PRNTF
	ISKIP	Set to 1 to avoid printing header between each line of the same print
/SORTYTGT/	MAXPT	Maximum number of points permitted in potential target array
	MYPOTGT(25)	RAIDSTRK index of target J

Table 36. (cont.)
(Sheet 12 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/SORTYTGT/ (cont.)	D(J1, J2) where J1 = J2 = 25	Flight distance from target (J1) to (J2)
	VALB(25)	Assumed value of bomb on target J -- equals RVAL(J)
	VALA(25)	Assumed value (for defended targets) of ASM versus bomb
	V(25)	Value of target as hit in sortie
	RHOJ(25)	Value of RHO(MHPOTGT(J)), for target J
	ATLOCHI(25)	Assumed value of local attrition to target (ATTRLOC(MYPOTGT(J)))
	DISTLEG(25)	Distance of flight leg to target from preceding flight point
	DISTLOW(25)	Part of above distance flown at low altitude
	S(25)	Estimated bomber survival probability on preceding leg
	SURV(25)	Estimated total survival probability to target
	VALDONE(25)	Estimated sortie value to and including target J
	VALON(25)	Estimated sortie value target J and beyond if SURV(J) were 1.0
	DVALB(25)	Estimated sortie value added by bomb planned on target J
	DVALA(25)	Estimated sortie value added by planned ASM on target J
	DREC(25)	Distance from depenetration to recovery for target J as last target
/TGTASGN/	TGTSASGN	Cumulative targets now assigned to raid
	TGTLIM	Value of TGTSASGN not to be exceeded for sortie

Table 36. (cont.)
(Sheet 13 of 13)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/TGTASGN/ (cont.)	NTGT	Number of targets allocated to raid -- equals NT
	IFSTGT	First target in list to be processed on this call
	ILASTGT	Last target in list to be processed on this call
	IFSTVEH	Index of first sortie to be processed on this call
	LSTVEH	Index of last sortie to be processed on this call
	ISIDE	The corridor side to be flown down
/VAL/	VALRECVR	Ratio of recovery value to total sortie value
	MUSTREC	Parameter card input; if >0 all aircraft must recover
	VUNLOAD	Significance parameter for final alterations in sortie if other than default (.005)
/1/	X(1100), Y(1100)	Temporary storage for values used in computing RHO and PHI; reused in subroutine GETGROUP for local variables
/2/		(Formerly common /CORPARM/)
	PHI(1100), RHO(1100)	Value of curvilinear coordinates PHI and RHO for Ith target
/4/	ISEQ(1100)	Temporary storage used by utility subroutine ORDER

SUBROUTINE CENTROID

PURPOSE:	Computes the centroid of a given array of latitudes and longitudes.
ENTRY POINTS:	CENTROID
FORMAL PARAMETERS:	NP, XLAT, XLONG, CXLAT, CXLONG
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	NOCORR, GENRAID

Method

This subroutine sums the NP latitudes which are in array XLAT, and the longitudes in array XLONG, then divides both by NP to get the average latitude CXLAT and the average longitude CXLONG.

If the points being summed fall on both sides of the 360° longitude, 360° is added to all the longitudes less than 180° before summing, and subtracted out afterwards. If CXLONG is greater than 360°, then 360° is subtracted from it before returning.

Subroutine CENTROID is illustrated in figure 120.

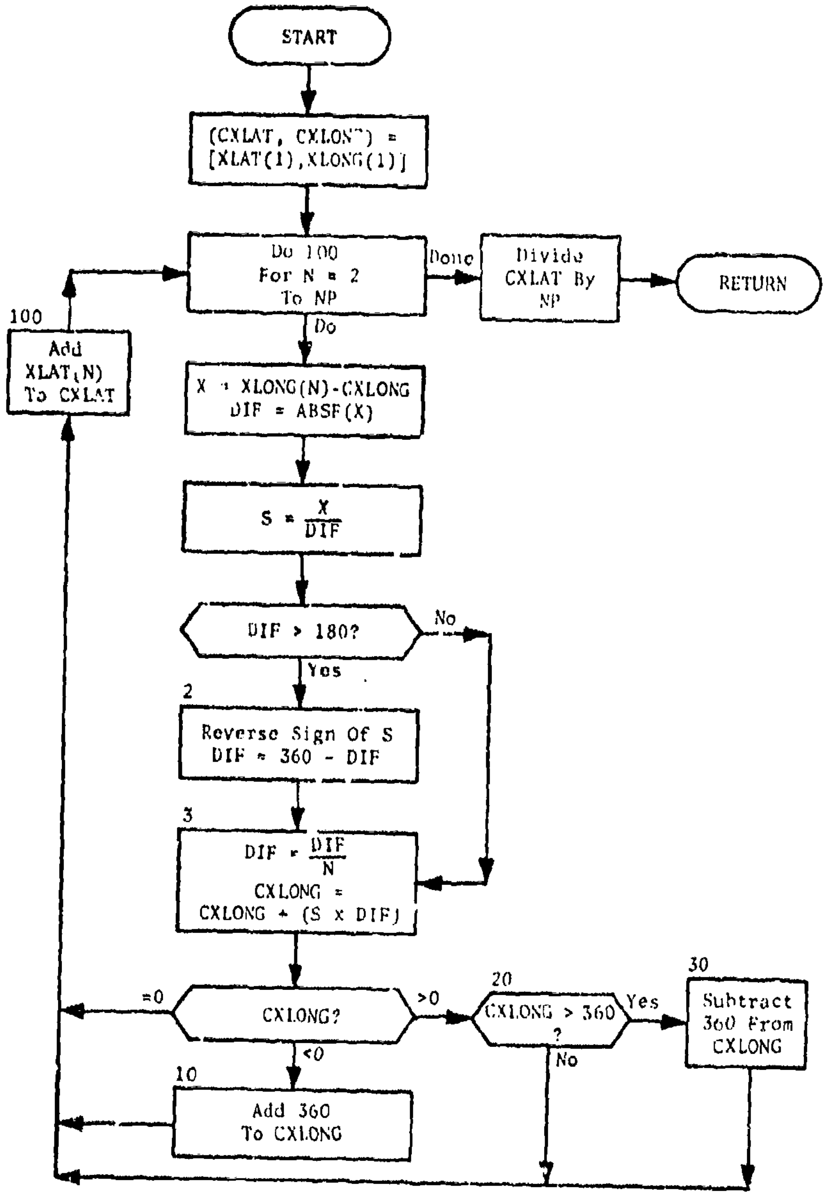


Fig. 120. Subroutine CENTROID

SUBROUTINE CHGPLAN

PURPOSE: To move a target from the omit list to the hit list, or from the hit list to the omit list.

ENTRY POINTS: CHGPLAN

FORMAL PARAMETERS: IOP

COMMON BLOCKS: CHGPLAN, CURSORTY, DEBUG, GRPTYPE, PCALL, PRINTOPT, SORTYTGT

SUBROUTINES CALLED: DIFF, PRINTIT

CALLED BY: GETSORT, SORTOPT

Method:

CHGPLAN may be called with IOP equal to any of the following four options:

Parameter	Value	Option
ATO	1	Move an ASM target from the hit list to the omit list
OTA	2	Move a target from the omit list to the hit list as an ASM target
BTO	3	Move a bomb target from the hit list to the omit list
OTB	4	Move a target from the omit list to the hit list as a bomb target.

The SORTYTGT index of the target to be moved is JDO in common /CHGPLAN/. If the target is being moved into the hit list, JAFT in common /CHGPLAN/ is the SORTYTGT index of the target in the hit list after which JDO is to be inserted. If the target being inserted into the hit list is to be an ASM strike, the variable IAIM in common /CHGPLAN/ is set to indicate the launch point of the ASM.

CHGPLAN makes the desired change in the hit list, then updates the following variables in common /CURSCRTY/: NOMIT, NUMASM, or NUMBOMB, NHIT, NUMHIT, LASTPAY, LASTTGT, and LKHITMT(JDC).

Subroutine CHGPLAN is illustrated in figure 121. Sheet 2 of the figure carries notes to facilitate interpretation of the flowchart.

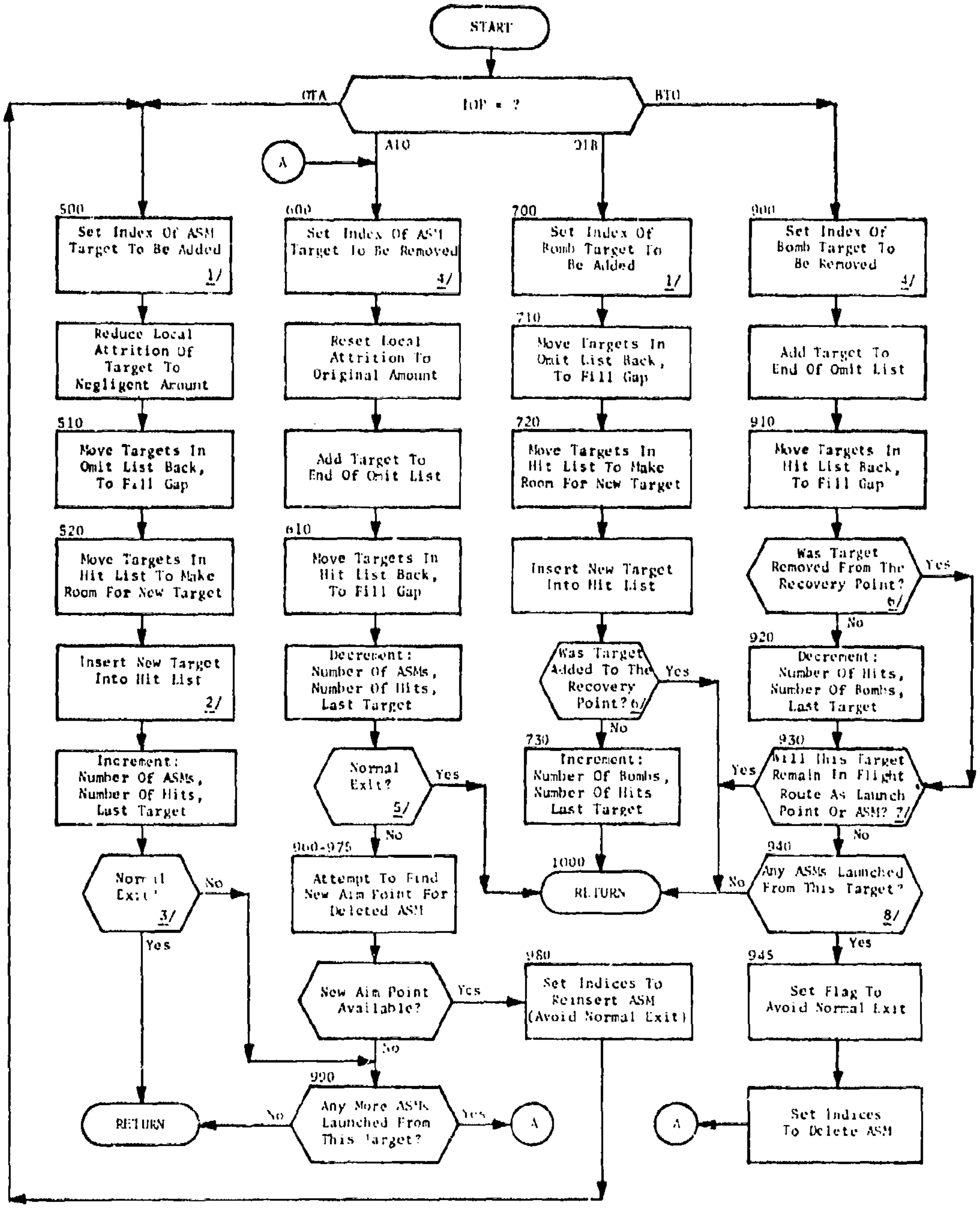


Fig. 121. Subroutine CHGPLAN
(Sheet 1 of 2)

Notes:

1. Local variable JADD is set equal to JDO for use within subroutine CHGPLAN. IADD is set to the position in the hit list which the target will occupy. IS is set to the target's current position in the omit list.

2. IHIT(IADD) is set to -JADD to indicate an ASM strike. IFLY(IADD) is set to IAIM, the target closest to the launch point (to be determined later in PLNTPLAN).

3. A normal exit (NORMAL = 1) is taken except when the ASM is being reinserted with a new launch point as a result of the omission of its former launch point from the hit list. (See BTO option.)

4. Local variable JDEL is set equal to JDO, and IDEL is set to the position in the hit list from which JDEL is being removed.

5. A normal exit (NORMAL = 1) is taken except when the ASM is being removed, to be reinserted with a different launch point as a result of the omission of its former launch point from the hit list on a BTO call.

6. Since the recovery point (IRECOVER) has a value associated with it, it may be removed or inserted into the hit list in the same way as the targets, in the attempt to increase the value of the sortie.

7. If EVALB finds that, in trading a bomb for an ASM on a target, the sortie will still have to fly to (within range of) the target, it sets IAIM negative as an indicator to CHGPLAN that it need not relocate any ASMs already being launched from the vicinity of that target, since that point will remain in the flight route. In this case, IAIM is reset positive and CHGPLAN exits.

8. If the fly point of the target which followed the deleted target (i.e., now occupies that target's position) is the same as the deleted target, it must be an ASM launched from that point. Since the launch point has now been omitted from the flight route, the ASM target must be deleted and reinserted if a new launch point can be found.

Fig. 121. (cont.)
(Sheet 2 of 2)

SUBROUTINE CORRPARM

PURPOSE:	To calculate a curvilinear coordinate system for use in assigning weapons to targets relative to the penetration corridor.
ENTRY POINTS:	CORRPARM
FORMAL PARAMETERS:	KORR, TLAT, TLONG, IST, IFN
COMMON BLOCKS:	CORRIDOR, 2, 1, PRINTOPT, DEBUG, PCALL
SUBROUTINES CALLED:	DELLONG, DISTF, PRINTIT
CALLED BY:	GENRAID

Method:

The new coordinate system has as its y-axis the corridor direction arrow, with the corridor origin (PLAT, PLONG) serving as the origin of the coordinate system. The corridor origin and the orientation point (CLAT, CLONG) or the tip of the arrow, come to subroutine CORRPARM in common /CORRIDOR/, which contains the characteristics of corridor KORR. IST and IFN are the first and last indices of the targets for which the calculations are to be done. The TLAT and TLONG arrays contain the coordinates for these targets.

The theory associated with determining the desired shape is given in the general discussion of program POSTALOC.

The following equations define the new coordinates ϕ and ρ (refer to figure 118).

$$\phi = x/y^k \tag{1}$$

$$\rho = y^2 + kx^2 \tag{2}$$

If ϕ exceeds 1.0 for a target, ϕ and ρ are redefined as follows:

If $x \geq 0$:

$$\phi = 1 + |x|^{1/k} - y, \tag{3}$$

$$\rho = |x|^{2/k} + kx^2 \quad . \tag{4}$$

If $x < 0$:

$$\phi = -\left(1 + \left|x\right|^{1/k} - y\right), \tag{5}$$

$$\rho = \left|x\right|^{2/k} + kx^2 \quad .$$

CORRPARM begins its calculations by translating the latitudinal and longitudinal coordinates of the earth so that the origin of the new system is at the last route point of the penetration corridor. At the same time, the longitudes are modified to conform to a planar model (by multiplying longitudinal distance by the cosine of PLAT).

The translated coordinate system then is rotated so that the ordinate is superimposed on the line connecting (PLAT, PLONG) and the corridor point (CLAT, CLONG). This is accomplished through use of the formulas:

$$x = x' \cos \alpha + y' \sin \alpha$$

$$y = -x' \sin \alpha + y' \cos \alpha \quad .$$

Referring to figure 122, we see that

$$X = -C\left(\frac{B}{S}\right) + A\left(\frac{D}{S}\right) = \frac{AD - BC}{S}$$

$$Y = -(-C)\left(\frac{D}{S}\right) + A\left(\frac{B}{S}\right) = \frac{AB + CD}{S} .$$

The constant $F' = \frac{3600}{2R}$, where R is the characteristic range of defense operations in nautical miles, is used to adjust the distance units.

The distance S is incorporated into this term, so that $F = \frac{F'}{S}$ = (3600/2*R*S) and

X = F*(A*D-B*C)

Y = F*(A*B+C*D)

After calculating X and Y, if Y is greater than zero, ϕ is calculated using equation (1) above, and R2 is set to Y^2. If the value of ϕ is greater than 1.0, it is recalculated using equation (3) or (5) depending on the sign X. R2 is set to $|X|^{2/k}$ in this case. ρ is then calculated using R2 as the first term in equation (2) or (4).

482-546 O - 72 - 15

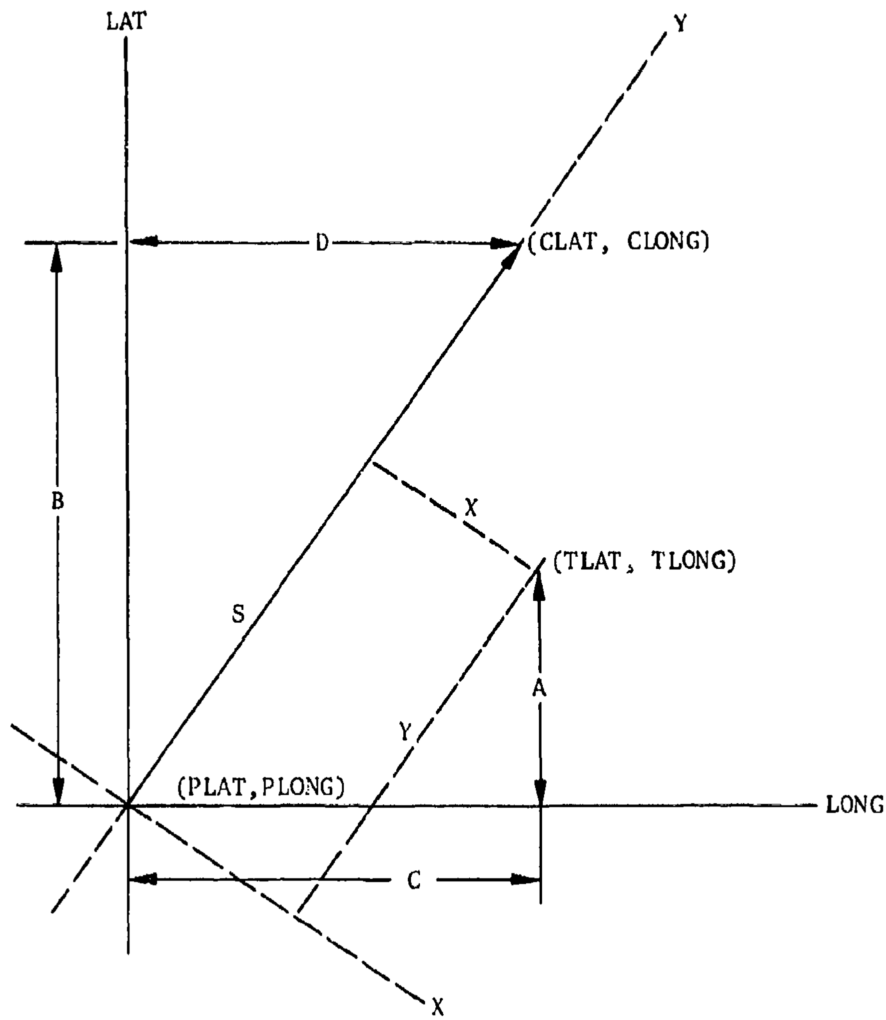


Fig. 122. Transformation of Coordinates (TLAT, TLONG) to (X, Y)

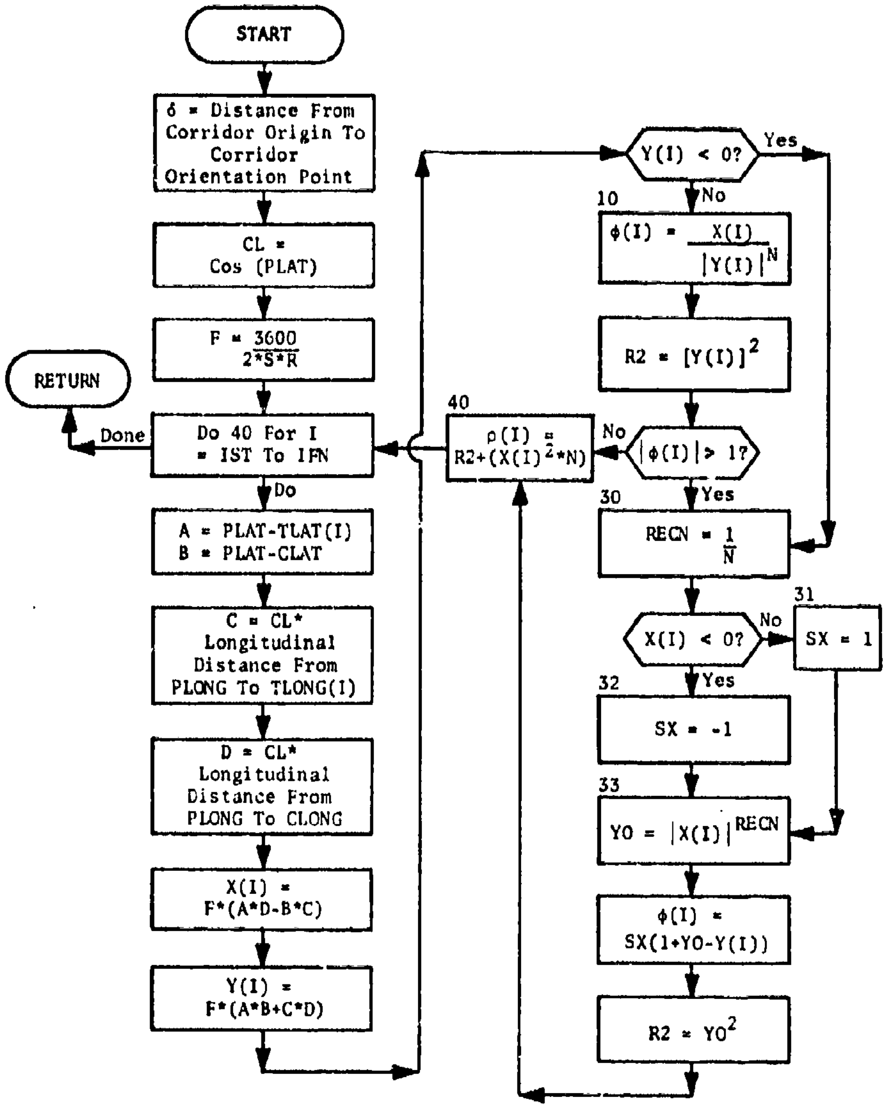


Fig. 123. Subroutine CORRPARM

FUNCTION DIFF

PURPOSE: To retrieve previously computed distances between potential targets from the distance array in common /SORTYTGT/.

ENTRY POINTS: DIFF

FORMAL PARAMETERS: II, JJ

COMMON BLOCKS: CORRIDOR, CURSORTY, 3, DEBUG, DPENREF, INDEX, PCALL, PRINTOPT, SORTYTGT, STRKSUM

SUBROUTINES CALLED: DISTF

CALLED BY: CHGPLAN, EVALB, EVALOA, EVALOB, FLTPLAN, GETSORT

Method

II and JJ are the SORTYTGT indices of the points between which the distance is to be computed. DIFF tests the (II, JJ) cell of the array (or the (JJ, II) cell if II >JJ) to see if the distance has been previously computed. The array is initialized to -10, so if the value is negative, it has not been computed. DIFF uses the RAIDSTRK indices (MYPOTGT(II) and (JJ)) to look up the TLATs and TLONGs and calls DISTF (LAT1, LONG1, LAT2, LONG2) to calculate the great circle distance. It then stores the result in D(II, JJ).

Function DIFF is illustrated in figure 124.

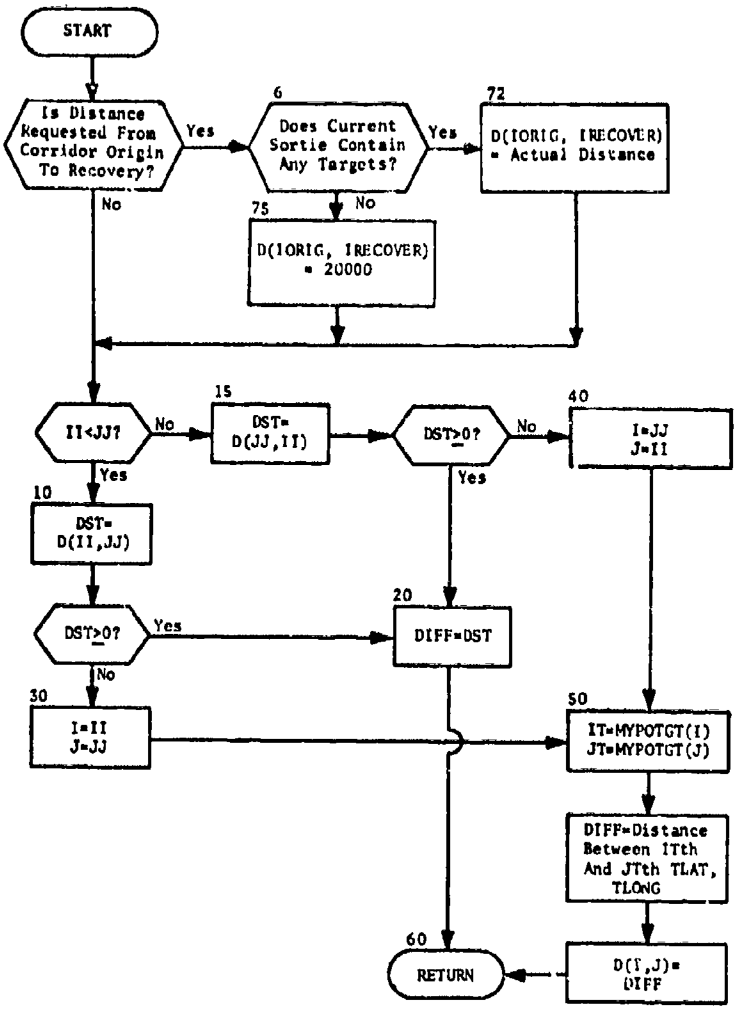


Fig. 124. Function DIFF

SUBROUTINE DUMPSRT

PURPOSE: To record the final (optimum) sortie plan in the JTGT array in common /CURRAID/.

ENTRY POINTS: DUMPSRT

FORMAL PARAMETERS: None

COMMON BLOCKS: CURRAID, CURSORTY, DEBUG, INDEX, PCALL, PRINTOPT, SORTYTGT

SUBROUTINES CALLED: PRINTIT

CALLED BY: OPTRAID

Method

DUMPSRT essentially copies the hit list in common /CURSORTY/ into the JTGT array for the current sortie in common /CURRAID/, changing the relevant counters (NAVAIL and NASGN) and returning the SORTYTGT indices to the AVAIL list. RAIDSTRK indices are used in the JTGT array rather than the SORTYTGT indices used in the hit list, but negatives are used in both places to indicate ASMs. NASGN is set negative if recovery has been omitted from the sortie plan, and IRECOVER is then removed from the omit list (GETSORT automatically reinserts it into the hit list for the next sortie).

Subroutine DUMPSRT is illustrated in figure 125.

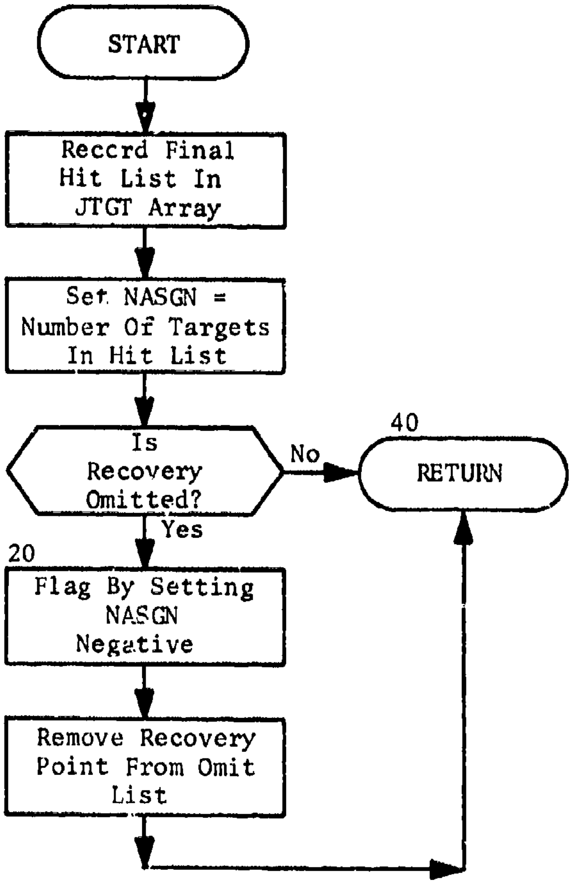


Fig. 125. Subroutine DUMPSRT

SUBROUTINE EVALB

PURPOSE:	To estimate marginal values of bombs in the sortie and the potential advantage of using ASMs instead.
ENTRY POINTS:	EVALB
FORMAL PARAMETERS:	None
COMMON BLOCKS:	CHGPLAN, CORRIDOR, CURSORTY, DEBUG, EVAL, FIXALL, GRPTYPE, INDEX, PCALL, PRNTF, PRINTOPT, SORTYTGT
SUBROUTINES CALLED:	DIFF, PRINTIT, PRNTF
CALLED BY:	SORTOPT

Method

Subroutine EVALB is called by SORTOPT to do one or more of the following functions:

- To determine which targets assigned bombs should be converted to ASMs when not all ASMs are assigned
- To determine which remaining bombs are of least value and should be deleted if too many strikes are assigned
- To determine which route points (recovery or bomb targets) are of negative value to the sortie and should be deleted.

The essential output of EVALB is stored in common /EVAL/. These outputs include:

MINB	Marginal value of least valuable bomb (or recovery point)
JDELB	Index to target whose value if MINB
JSEQERR	Index to route point in wrong geographic sequence if non-zero
MAXDAB	Largest estimated increase in value of sortie by changing from bomb to ASM on a target
JADDA	Index to target associated with MAXDAB
JAF	Index to route point which ASM should follow in revised sortie definition

EVALB does its processing in one large cycle over all route points in succession, down to and perhaps including the recovery point. Three indices JH1, JH, and JH2, are carried along representing three consecutive route points. Local attrition is computed for target JH. DISTSV, the distance saved by omitting target JH, is then calculated. A check is made at this point using previously computed distances to see if the distance traveled would be shorter if JH1 and JH were switched. Next, using DISTSV, the area attrition reduction due to the omission of JH is calculated. An estimate is then made of the total value of omitting JH, due to increased low-altitude range, and decreased attrition rate. If there are ASMs which use the target JH as a launch point, an attempt is made to relocate them. Finally, the differential value DVALB of a bomb on target JH is calculated. If this value is the least so far, JH is flagged. At this point, if there are no unused ASMs, the program recycles to process the next target.

If there are available ASMs, the target JH is evaluated as a potential ASM target. The differential value of the target for an ASM(DVALA) is calculated, and then the marginal improvement (DAB) if an ASM is used rather than a bomb. The target with the greatest DAB is flagged for SORTOPT.

Figure 126, in two parts, illustrates the operation of the subroutine via a macro flowchart. A detailed flow of EVALB is shown in figure 127; sheets 5 and 6 of this figure are notes to facilitate reading of the flowchart.

The processing of each route point is handled in two parts. In Part I (figure 126) the marginal value of the route point as a target for a bomb is evaluated. In Part II, the value of the same route point is calculated as a potential ASM target and the marginal value of changing it to an ASM target is estimated. Clearly when Part II of the program is to be used, ISCAN must be set so that the recovery point is not included in the evaluation.

When all ASMs have been assigned, there may still be too many strikes for the available warheads. The subroutine may be called again, still excluding the recovery point, to select the least valuable remaining bomb which could be deleted. The subroutine is called once more with ISCAN set to include the recovery point to be sure that all route points including the recovery make a positive contribution to the payoff.

Computational Method Used in EVALB

Part I evaluates the marginal value of a route point. The value of reaching the route point, multiplied by the probability of surviving to reach it, is compared with the cost of doing so.

This cost consists of two elements:

- Change in the probability of reaching succeeding targets because of local attrition, if any, at this target, or because of additional area attrition over the added distance required to flv to this target
- Reduction in the amount of low-altitude flight available because of the extra distance to the target, which in turn can affect penetration probability to all targets.

In analyzing each target, EVALB considers an alternate flight route which bypasses the target and goes directly from the preceding to the succeeding target. The effect of this route on the expected payoff for succeeding route points can be directly evaluated. The change in attrition is known, so the change in the cumulative survival probability SURV to the succeeding target can be computed and the value VALON of the remainder of the sortie is available (having been computed by FLTPLAN).

The change ΔV in VALSORTY, due to change in available low-altitude capability, is only estimated. The estimate is based on the amount of distance saved by skipping the target DISTSV multiplied by the quantity VALDIST, the marginal value of distance as estimated by FLTPLAN. However, where the saving in distance is very large, this type of linear extrapolation with a constant VALDIST can be quite misleading, and could even exceed the full value of all targets in the sortie. Obviously, the value of the sortie can never exceed the actual value VALMAX of all route points, and with one target, k, omitted could not exceed VALMAX-V(k). Consequently, the value, VALO, of omitting a target, k, cannot exceed POTVALO= VALMAX-V(k)-VALSORTY. This quantity POTVALO is therefore used to establish a limiting value for the value of saving distance. The quantity VALDIST is used to give the derivative for small values of DISTSV. The actual form used for estimating ΔV for distance saved is:

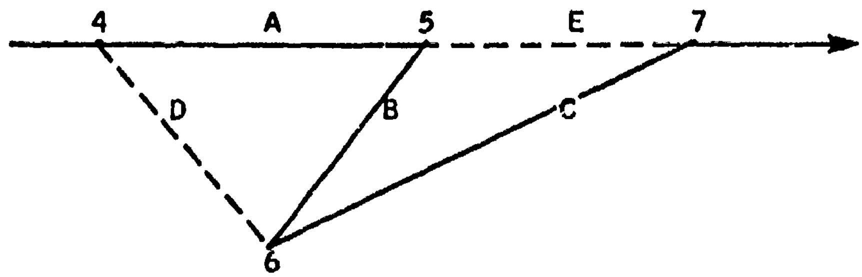
ΔV = POTVALO * [1.0 - 1.0/(1.0 + TEMP)]

where TEMP = VALDIST * DISTSV/POTVALO

In the second phase of the process -- to estimate the value of the target as an ASM target -- the time premium for using an ASM on the target is added into the basic value, RVAL, of the target, and the survival probability used is that for the earliest possible launch point in range of the target.

Special Considerations

Determination of the value of omitting a route point requires calculation of the distance saved. Once this information has been computed for two successive route points, the necessary distances have been computed that are required to determine whether the two points are out of order on the route. Therefore this determination is made during Part I of EVALB. The following figure illustrates the method used.



The figure illustrates a route:

4 via leg A to 5

5 via leg B to 6

6 via leg C to 7

We wish to consider the possibility of reversing the order of points 5 and 6 on the route. The present distance is A + B + C, the revised distance would be D + B + E, using dotted alternative legs D and E.

If the reversed path is shorter, then $D + B + E < A + B + C$ or $A + C - D - E > 0$. When we consider omitting 5 we compute DISTSV = A + B - D. When we consider omitting 6 we compute DISTSV = B + C - E.

Adding the two values of DISTSV and subtracting 2B we obtain A + C - D - E. Therefore if this value is positive the two route points are out of order, and the flag JSEQERR is set to indicate one of the two targets for possible temporary omission by SORTOPT. Usually the first target is flagged. (The presumption is that a later call by SORTOPT on EVALOB will result in the replacement of such a target in its proper position in the sortie.) However, if the first target is also a launching point for ASMs, even temporary omission would be complicated; thus, rather than seek an alternative launch point for the ASMs, the second target will be flagged instead. If both route points are also ASM launching points, no flag is set; and the current order of targets is not changed.

The problem of route points serving double duty as ASM launch points also arises in EVALB when the marginal value of omitting route points is being estimated. Therefore after the original value VALO is estimated in Part I of EVALB, a check is made to see if the point is used as an ASM launch point. If so, the value VALO of omitting the point is decremented to reflect changes in the marginal value of the ASM, for which a new and probably inferior launch point must be found. If such an alternate launch point cannot be found, the entire value of the ASM is charged to VALO. Except in the most extreme cases, this is sufficient to preclude omission of this target.

This treatment is paralleled in the subroutine CHGPLAN. If SORTOPT asks CHGPLAN to delete a bomb where the same route point is used as a launch point for an ASM, CHGPLAN seeks an alternate launch point for the ASM. However, if it cannot find one the ASM is omitted also. Thus the payoffs estimated by EVALB correspond correctly to the options that would be implemented by CHGPLAN.

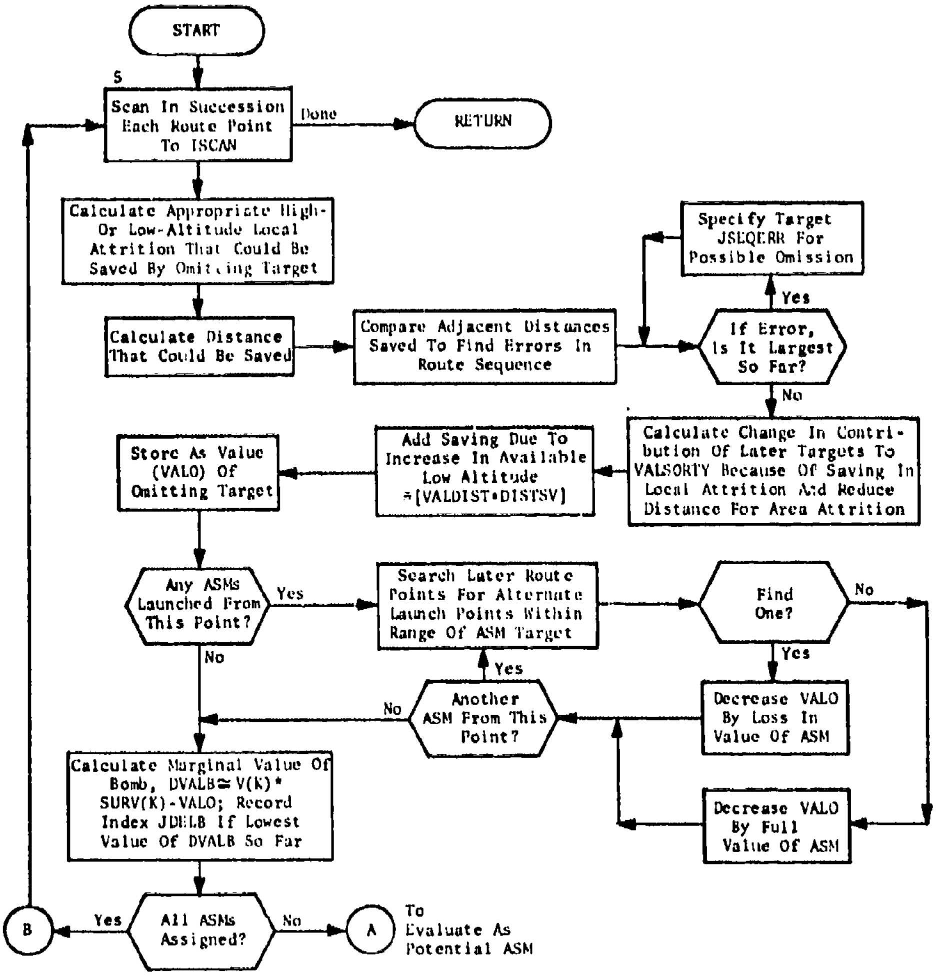


Fig. 126. Subroutine EVALB (Macro Flowchart)
Part I: Estimate Marginal Value--Route Point

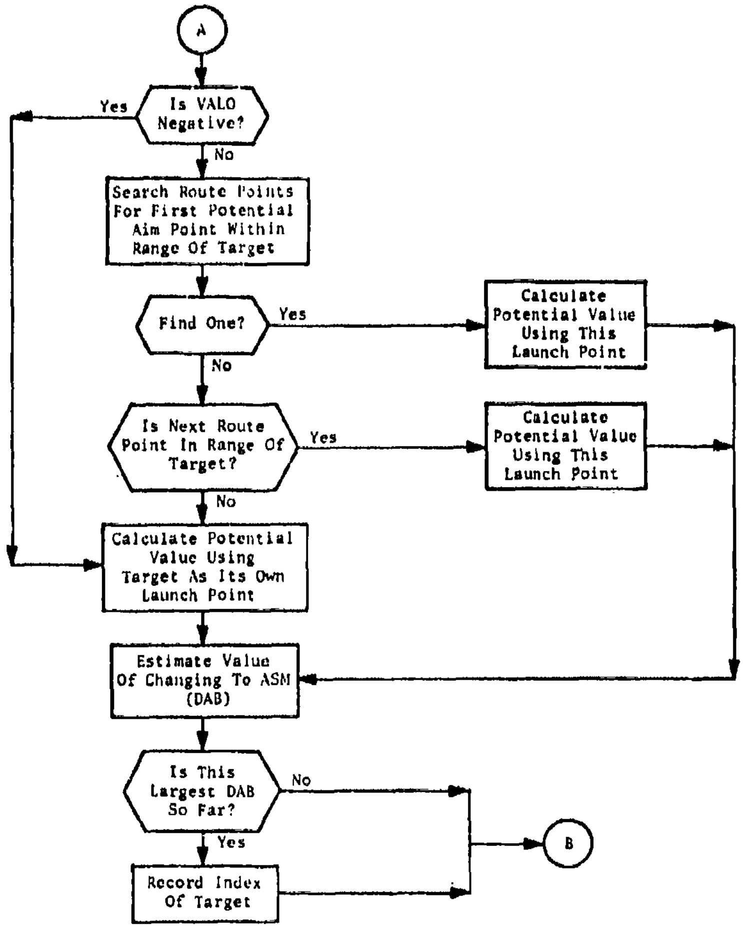


Fig. 126. (cont.)
Part II: Estimate Potential Value--ASM Targets

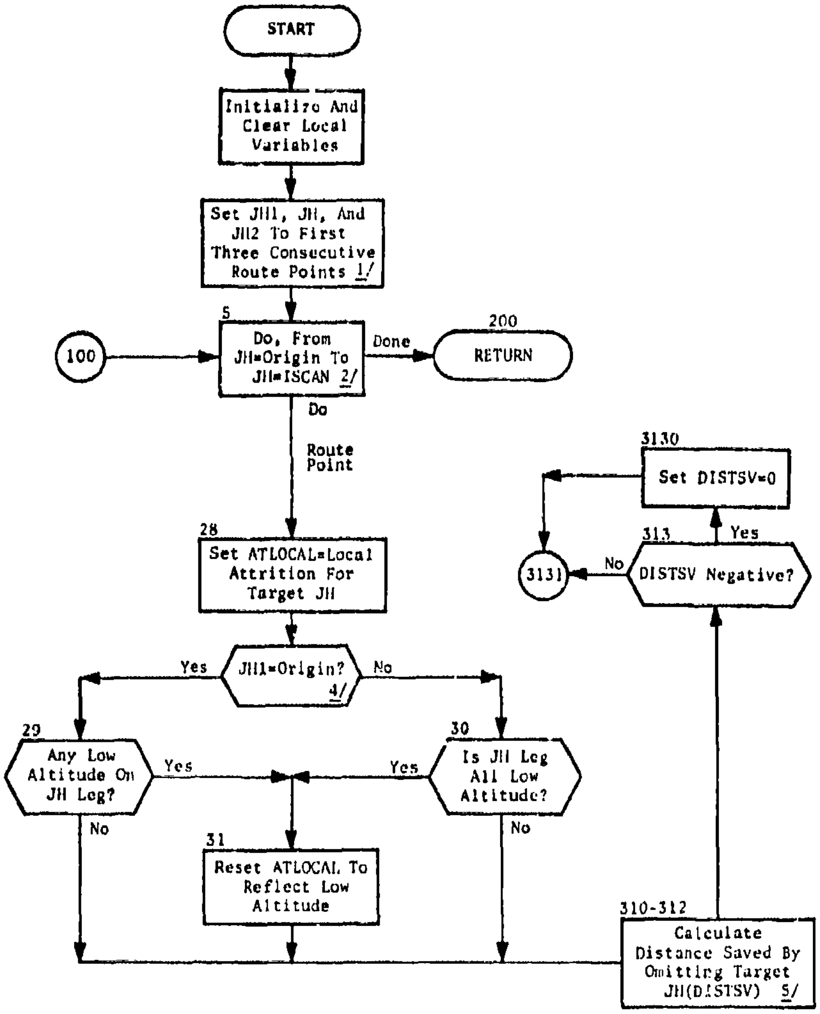


Fig. 127. Subroutine EVALB (Detailed Flowchart)
(Sheet 1 of 6)

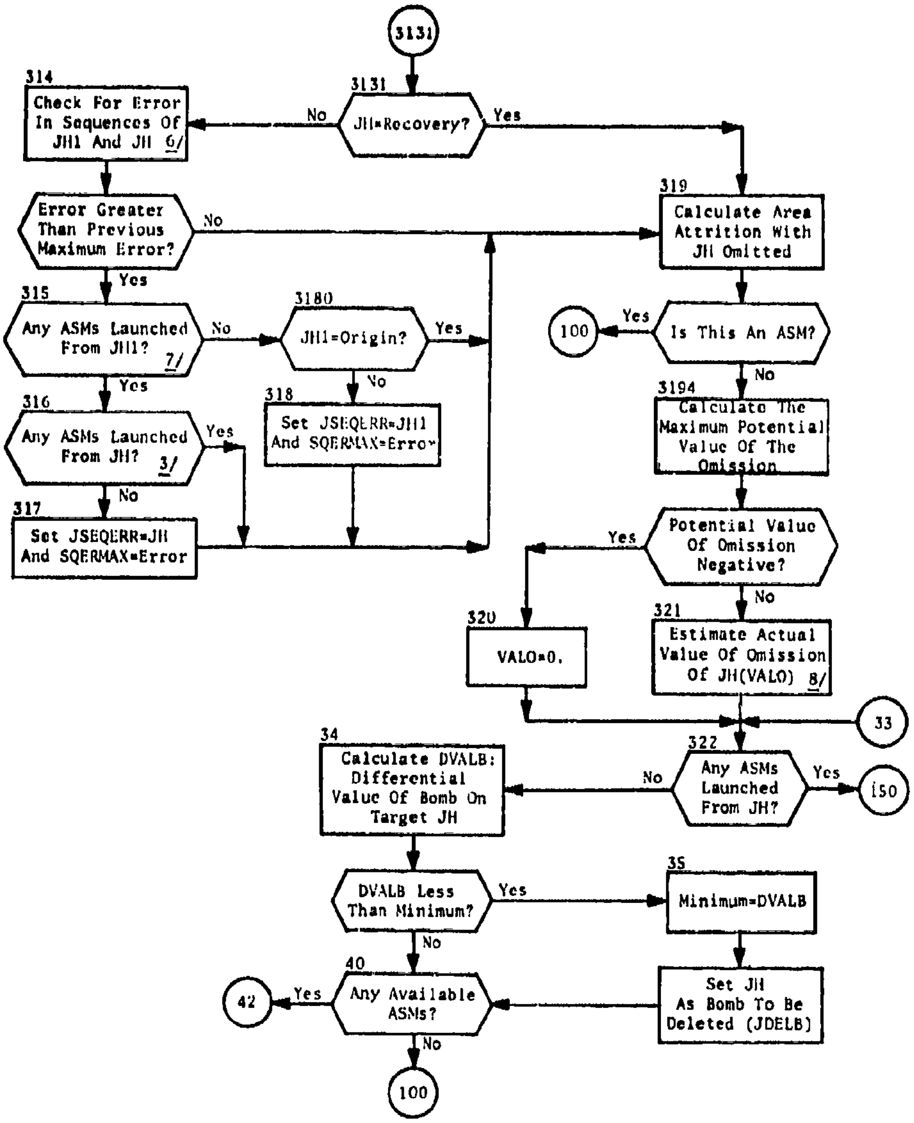


Fig. 127. (cont.)
(Sheet 2 of 6)

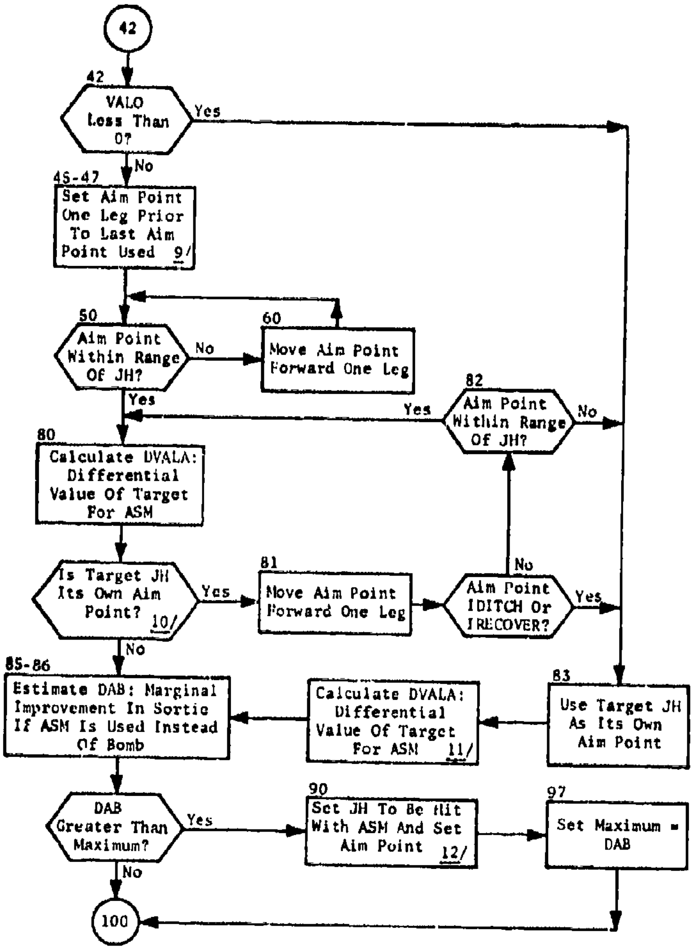
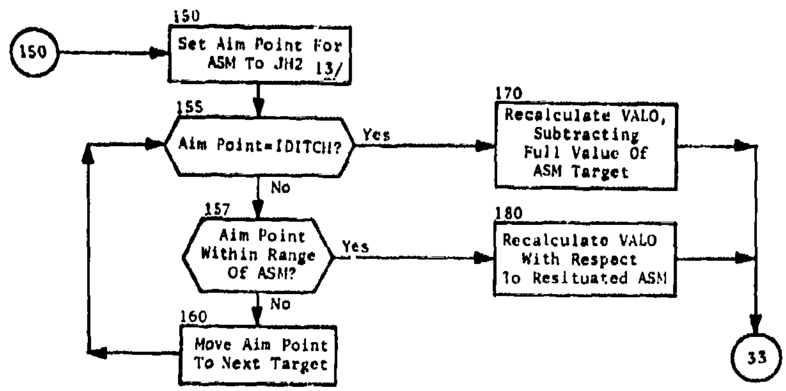


Fig. 127. (cont.)
(Sheet 3 of 6)



Local Variable Definitions:

JX1, JX2 - The SORTYTGT indices of two consecutive route points between which target JX may be added

IX1, IX2 - The hit list indices of JX1, JX2

DIST1 - Distance from JX1 to JX

DIST2 - Distance from JX to JX2

DISTAD - Distance added to flight route by inserting target JX

JXTEST - The SORTYTGT index of the target on the other side of the closer of JX1 and JX2 to JX, i.e., either IX1-1 or IX2+1

IXTEST - The index of JXTEST in the hit list

DISTEST - Distance between JXTEST and JX

ADDTEST - Distance added by inserting JX in alternate position

Fig. 127. (cont.)
(Sheet 4 of 6)

Notes:

1. I1 is kept as the index in the hit list for target JH2. IA is used as the index to the current "fly point" JA.
2. This "do-loop" is over all actual route points, as opposed to ASM targets.
3. If two consecutive fly points are the same, the second (at least) is an ASM launched from that point.
4. If JH1 = IORIG, JH is the first target and low altitude on that leg is assigned starting from the target and working back to the origin. Thus, any low altitude at all implies low altitude at the target. All other legs, however, are assigned low altitude working from the previous target to the current one, and the entire leg must be at low altitude for the target to be at low altitude.
5. DREC(J) is the undefended distance from the depenetration point associated with target J to the corresponding recovery point. This distance is weighted to assume half the importance of the defended distance. If JH2 is recovery, JH is the last target in the sortie, and omitting JH may change the depenetration route if JH1 is closer to another. Thus, DREC must be considered in computing the distance saved in this case.
6. Using OLDSAVE (which was DISTSV on the previous cycle) and the current DISTSV, the effect of switching the order of JH1 and JH is calculated. JSEQERR is a measure of the distance saved by this switch.
7. If there is a sequence error, either JH1 or JH may be omitted under the assumption that it will be picked up later in the optimizing process. JH1 is normally dropped, unless there are ASMs being launched from that point. In that case JH is checked. If it also is a launch point for ASMs, the error is ignored. If JH1 is the origin, the error is ignored.
8. The value of omitting JH is computed in such a way that if the value of distance times the distance saved (VALDIST * DISTSV) is small, VALO is nearly equal to that product; and that as that product increases, VALO approaches, but never exceeds, the potential value of the omission POTVALO. The value of the sortie due to increased survival probability is also included in VALO.
9. As EVALB proceeds in its overall loop, JA is kept set as the first route point within ASM range of the current target. As each new target is processed, the assumption is made that no point earlier than the one immediately prior to the last JA will be within the range of the current JH, so JA is moved back one position and testing begins from there, to find one within range. A check is made to assure that a new route point is tested each time.

Fig. 127. (cont.)
(Sheet 5 of 6)

10. A test is made to see if the first target within range was JH itself. If so, the succeeding target is tested. If the target is within range and is not the recovery or ditch point, it is used for the aim point. Otherwise, the JH target is used as its "own aim point"; i.e., the sortie plans to fly within ASM range of the target.

11. In computing DVALA when the sortie must fly to within range of an ASM target, the VALB component of DVALA is computed as if it were a bomb, with the exception of the local attrition.

12. JAF is set to indicate to SORTOPT, and hence to CHGPLAN, the target in the hit list after which an ASM or bomb should be inserted. For an ASM, it is normally the target used as launch point. In the case where the ASM target is its own aim point, JAF is set to the preceding target and IAIM, which indicates the fly point, is set negative as a flag for CHGPLAN, signifying that in omitting this bomb (for reinsertion as an ASM) it need not drop or relocate any associated ASMs since the route point will not be deleted.

13. If the target chosen to be switched from a bomb to an ASM strike is already a launch point for other ASMs, new launch points must be located for these ASMs. The effect of this change, or the total value of the ASMs if new launch points cannot be found, must be figured in the value of omitting the bomb (VALO).

Fig. 127. (cont.)
(Sheet 6 of 6)

SUBROUTINE EVALOA

PURPOSE:	To evaluate the advantage in substituting a target from the omit list for a current ASM target.
ENTRY POINTS:	EVALOA
FORMAL PARAMETERS:	None
COMMON BLOCKS:	CHGPLAN, CURSORTY, DEBUG, EVAL, FIXALL, GRPTYPE, INDEX, PCALL, PRINTOPT, PRNTF, SORTYTGT, 3
SUBROUTINES CALLED:	DIFF, PRINTIT, PRNTF
CALLED BY:	SORTOPT

Method

EVALOA estimates the desirability of using an ASM on one of the omitted targets. It can be called by SORTOPT either to find a target for an unused ASM or to evaluate the value of substituting an omitted strike point as the target for an ASM already assigned. The main outputs of EVALOA are stored in common /EVAL/. They are:

MINDA	The minimum marginal value for ASMs as now assigned
JDEL	The index to the target with value MINDA
MAXDA	The maximum marginal value for ASMs on any omitted target
JADD	The index for the target with value MAXDA
IAIM	The index for the launch point for the ASM
JAF	The index for the route point the ASM should follow in the sortie

EVALOA does not deal with any changes in the bomber route. In this way the values of changes considered by EVALOA can be evaluated exactly.

The operation is divided into two portions. First, EVALOA scans all targets in the mission currently assigned for ASMs, skipping any target used as its own launch point, since its omission would change the bomber route. The marginal value of the others is determined by multiplying the value of the strikes as ASM targets by the survival probability for the aircraft to the launch point. During this phase EVALOA determines the strike JDEL with the lowest marginal ASM value, MINDA.

In the second part of the subroutine, all omitted strikes are evaluated as ASM targets. The method of evaluation is exactly the same. The only complication is that a suitable launch point must be found. The routine simply takes the first route point within range of each target as the potential launch point. As it proceeds through this part of the program, it keeps a record of the strike, JADDA, with the highest marginal ASM payoff, MAXDA, and the associated launch point IAIM. Of course, strikes are disqualified for such consideration if another strike on the same target is already in the sortie definition.

In the first loop over all targets in the sortie, the differential value of each ASM (DVALA) is calculated. The smallest of these DVALAs is flagged. In the second loop over all targets in the omit list, a launch point is located and DVALA is calculated for each target. The largest of these DVALAs is flagged. The values of these two targets, MINDA and MAXDA, are returned to SORTOPT in common /EVAL/.

As in the other evaluating routines, JH is the index of the target under consideration in the SORTYTGT arrays, I1 is its position in the hit list, and JA is the SORTYTGT index of its corresponding fly point. JX is the SORTYTGT index of a target in the omit list, and IA is the position of target JA in the fly list.

To check for the same target already being in the sortie, the original INDEXNO must be obtained from the RAIDSTRK data (common /3/). MYPOTGT(JX) contains the RAIDSTRK index of target JX in the omit list.

As in EVALB, JAF is the SORTYTGT index for the route point that the new ASM should follow in the sortie. This is usually the same launch point chosen for the ASM, but when the ASM is its own launch point. JAF becomes the preceding point and the target keeps its original position in the hit list.

Subroutine EVALOA is illustrated in figure 128.

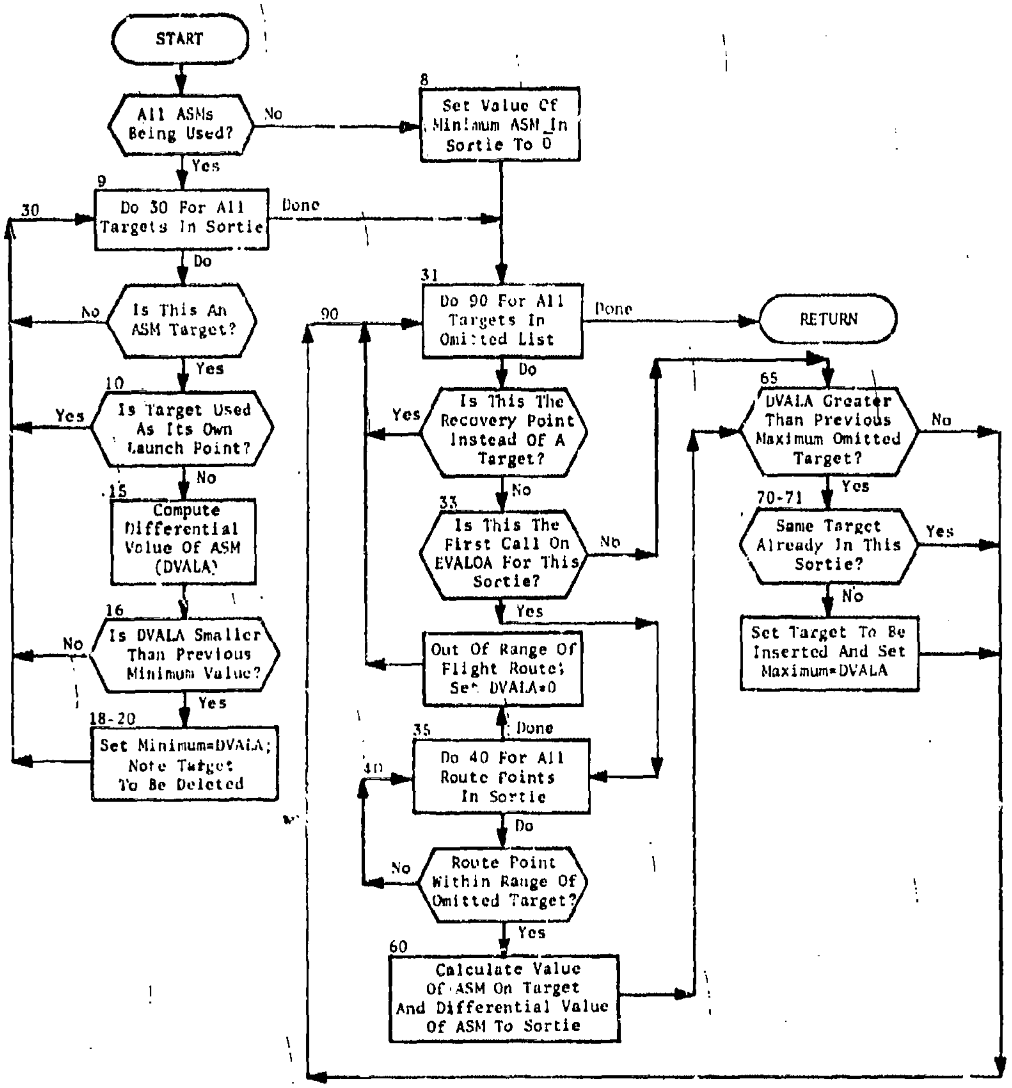


Fig. 128. Subroutine EVALOA

SUBROUTINE EVALOB

PURPOSE:	To evaluate the advantage of substituting a target from the omit list for a current bomb target.
ENTRY POINTS:	EVALOB
FORMAL PARAMETERS:	None
COMMON BLOCKS:	3, CORRIDOR, CURSORTY, DEBUG, EVAL, INDEX, PCALL, PRINTOPT, PRNTF, SORTYTGT, VAL
SUBROUTINES CALLED:	DIFF, PRINTIT, PRNTF
CALLED BY:	SORTOPT

Method

EVALOB estimates the value of strikes in the omit list as potential targets for bombs. It is called by SORTOPT to find an additional target or to find an omitted target that is more profitable for a bomb than the least valuable in the sortie. The main outputs from this routine are placed in common EVAL and consist of:

- MAXOB The maximum potential marginal value for an omitted strike as a target for a bomb
- JADDB The index to the above target
- JAF The route point in the sortie which it should follow

The subroutine processes in turn each target in the omit list. Each potential target is tried, first in a position in the flight route just before the first target in the list with a higher value of RHO. The distance added to the sortie is then evaluated. The target is then tried in a position on the other side of its nearest neighbor (nearest in value of RHO). If this position produces a lower value for the distance added, this position is accepted instead of the original position.

The marginal contribution of the bomb in the preferred position is then computed. The method parallels the calculation of the marginal value of bombs in EVALB. The effect of the extra attrition on following targets is evaluated. Then the effect on low-altitude range is estimated using (VALDIST * DISTAD). These quantities are added to get the total benefit, VALO, of not flying to this new route point. The value of the target times the probability of surviving to reach it is then computed to get the benefit of adding the target. Finally VALO is subtracted from the benefit to get the net marginal value of adding the target, DVALB.

The index for the target with the highest DVALB is then recorded as JADDB and the route point it should follow is recorded as JAF. Of course, any strike on a target already in the sortie is excluded from consideration, to avoid duplicate strikes on the same target by the same bomber.

Subroutine EVALOB is illustrated in figure 129. Sheet 3 of the figure includes notes to facilitate interpretation of the flowchart.

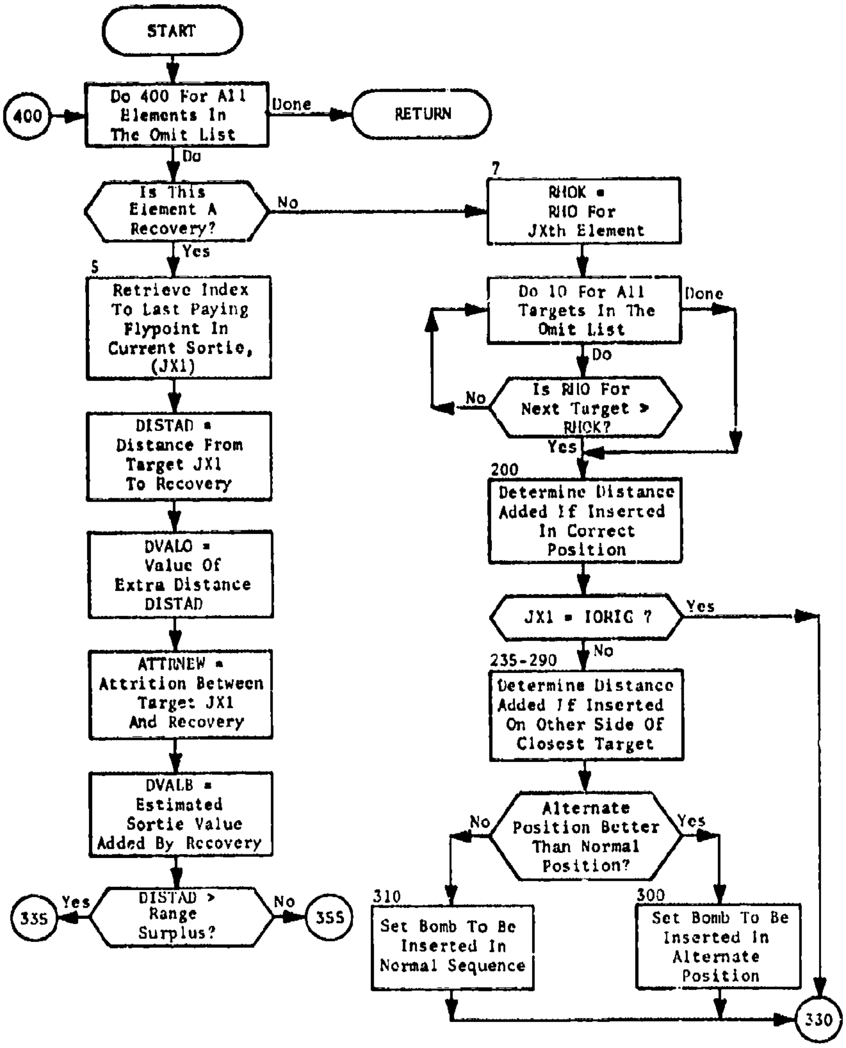


Fig. 129. Subroutine EVALOB
(Sheet 1 of 3)

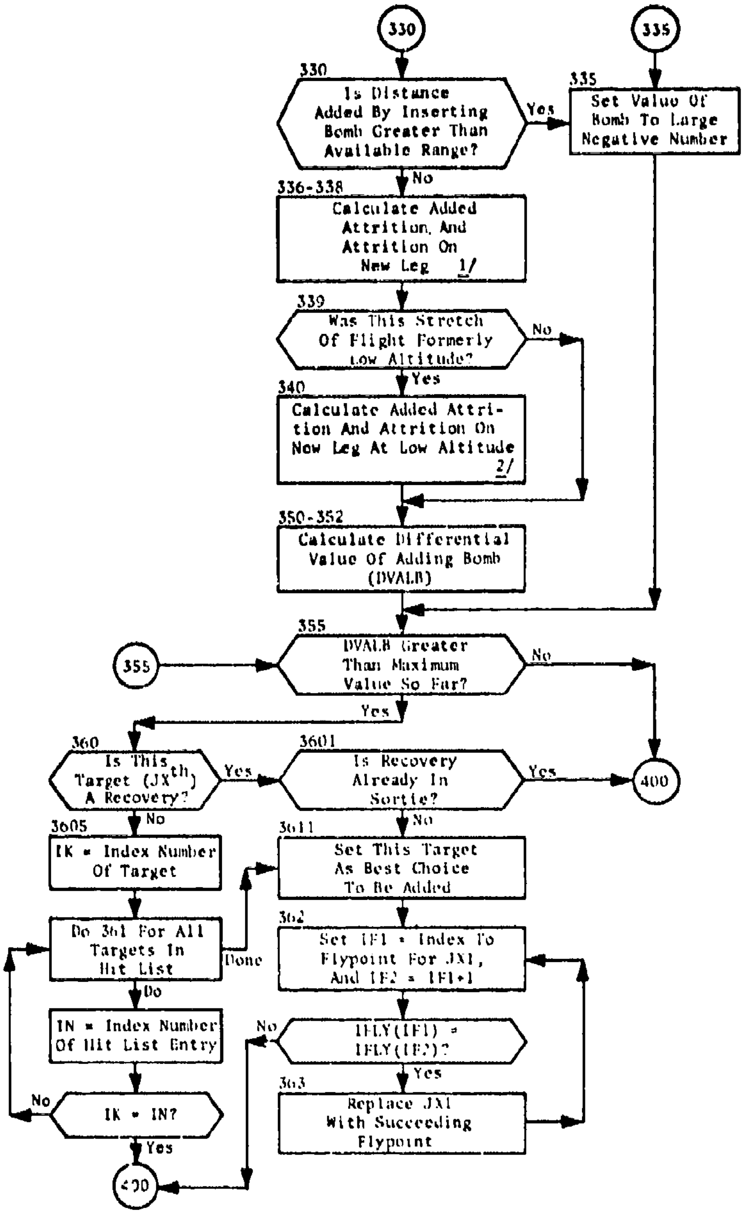


Fig. 129. (cont.)
(Sheet 2 of 3)

Notes:

1. Before calculating DVALB, a check is made to see if all of the warheads on the sortie are ASMs. (It is possible for EVALOB to be called if an ASM is dropped in the process of sortie optimizing. In this case EVALOB picks up a bomb without deleting one, and then EVALB is called to switch a bomb to an ASM.) If all warheads are ASMs, then local attrition is ignored in computing DVALB.

2. If the entire JX2 leg was low altitude before, the assumption is made that when target JX is inserted, the additional distance flown will be at low altitude and the new attrition calculated is reduced by the factor ATTRHILO.

   Finally, if all warheads are ASMs, VALA is added into the final computation of DVALB.

Fig. 129. (cont.)
(Sheet 3 of 3)

SUBROUTINE FLTPLAN

PURPOSE: To set up the flight profile and evaluate the sortie plan.

ENTRY POINTS: FLTPLAN, FINFLT

FORMAL PARAMETERS: None

COMMON BLOCKS: 3, CORRIDOR, CURRAID, CURSORTY, DEBUG, EVAL, GRPTYPE, INDEX, INITOPT, OUTSORT, PCALL, PRINTOPT, PRINTF, RAIDSHR, RUNCHECK, SORTYTGT

SUBROUTINES CALLED: DIFF, PRINTIT, PRNTF

Method

FLTPLAN is used by SORTOPT to provide an estimate of VALSORTY for any given sortie definition. It makes use of the marginal target value RVAL supplied by the allocator, together with attrition parameters for the corridor, to estimate the value of a sortie. The routine is called by OUTSRT using the entry point FINFLT.

The value of a sortie as computed by FLTPLAN is given by:

$$VALSORTY = \sum SURV(I) * V(I)$$

where the summation is over all flight points including recovery. SURV(I) is the estimated probability of the bomber surviving to reach the flight point I, and V(I) is the estimated value of reaching that point.

The value V(I) attached to the targets, I, depends on whether it is to be attacked by a bomb or an ASM.

(1) If I is a target for a bomb then:

$$V(I) = RVAL(tgt)$$

(2) If I is a target for an ASM then:

$$V(I) = RVAL(tgt) * [1.0 + TIMEPREM(tgt)]$$

In the second relation, TIMEPREM is a bonus factor that is given for using an ASM on certain classes of targets. At present TIMEPREM is nonzero only for air defense targets. This bonus is intended to reflect the advantage of destroying these targets before the aircraft and others in the same flight have to pass the target.

(3) If I is a recovery point then we define:

$$V(I) = .5 * \sum RVAL(tgt)$$

In the third equation the summation is over all targets in the sortie, which implies that the value of recovery is equal to one-half the value of all targets in the mission. The computation of SURV(I) for the formula is based on a simple exponential attrition law.

If the integrated attrition probability on each individual leg to a point J is given by ATLEG(J), then the survival probability for the bomber to the point I will be given by:

$$SURV(I) = EXPF \left\{ - \sum_{J=i}^{J=I} ATLEG(J) \right\}$$

The attrition ATLEG(J) includes both area and local attrition for the leg. Figure 130 illustrates the attrition rates used by FLTPLAN.

The area attrition for each leg is computed by integrating the assumed area attrition rate over the length of each leg. After the first target, this assumed area attrition rate per nautical mile is a constant, equal to the data base variable ATTRCORR supplied for the corridor. Prior to the first target, the assumed attrition rate falls off exponentially toward the limiting value ATTRSUPP which is also a data base variable for the corridor. Thus the assumed attrition rate between the origin and the first target is given by

$$Rate = ATTRSUPP + (ATTRCORR - ATTRSUPP) * EXPF(-X/DEFRANGE)$$

where X is the distance in nautical miles prior to the first target. Attrition rates (ATTRLEG) may also be specified for the precorridor legs leading in to the corridor.

The local attrition ATTRLOC (see TGT2 in figure 130) obtained directly from ALOCOUT is estimated directly from the data base variable TARDEF (a TYPE INTEGER variable) associated with each target as follows:

ATTRLOC = .1 * TARDEFHI

Naturally this local attrition is operative in FLTPLAN only when the route point having local attrition is itself a target for a bomb. It does not apply if the local attrition is associated with an ASM target and the sortie definition shows the ASM as launched from some other route point. Moreover, even if the sortie definition shows the ASM target as the ASM launch point, any local attrition associated with the target is ignored. This is done because it is presumed that the actual launch point (to be defined in PLNTPLAN) will not require the aircraft to penetrate the local defenses. In FLTPLAN the local attrition is applied entirely to the incoming leg to the target.

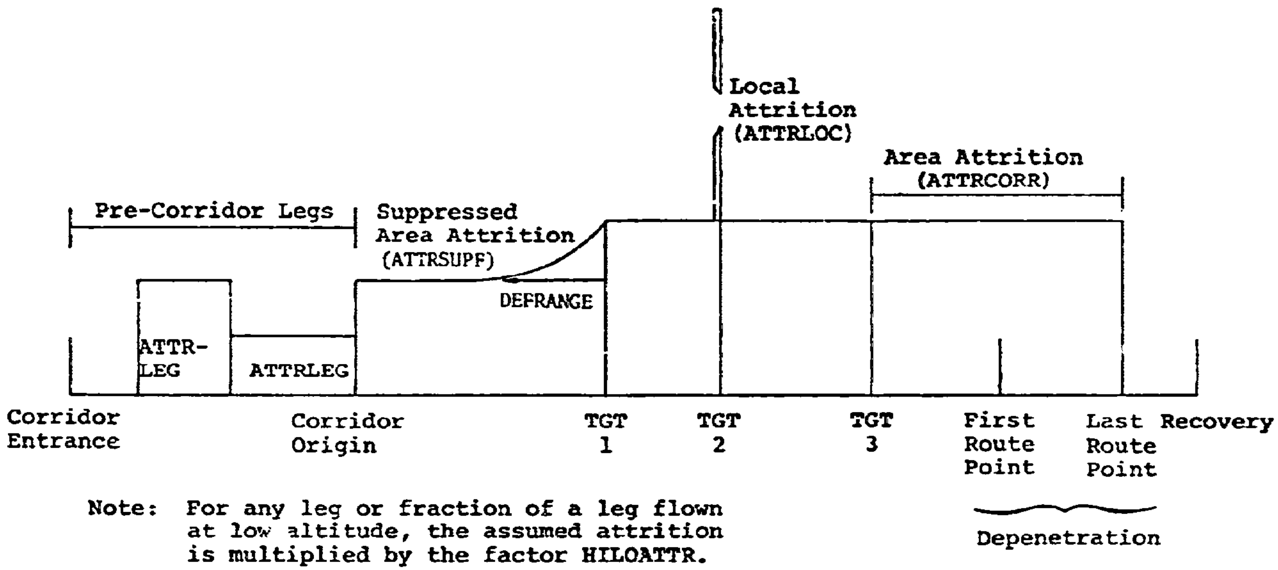
FLTPLAN assumes that on any leg or fraction of a leg flown at low altitude the attrition rates will be reduced by the factor HILOATTR. In order to estimate the expected value of the sortie, therefore, an estimate must be made of how the available low-altitude range should be applied.

Notice that a change in the assumed attrition rate for any leg or part of a leg will change the integrated attrition for the leg ATLEG(J). This in turn will change the probability of survival to any point I(SURV(I)) which is required to evaluate VALSORTY.

FLTPLAN therefore begins by summing the total distance for the sortie as specified. This distance is subtracted from the aircraft range to give the surplus range RNGSURP available for the mission. Using the conversion factor, RANGEDEC, this surplus range is used to estimate the available low-altitude distance, AVAILOW, for the mission. Finally, AVAILOW is allocated to the various legs in a manner intended to maximize the value of the sortie, VALSORTY.

During this allocation of available low-altitude range the following alternatives are provided:

1. Allocate low-altitude range to that remaining precorridor leg that has the highest attrition.
2. Extend the low-altitude flight from the first target one more leg toward the depenetration point (where the attrition is assumed to end).
3. Extend the low altitude a little further in front of the first target toward the corridor origin.



Note: For any leg or fraction of a leg flown at low altitude, the assumed attrition is multiplied by the factor HILOATTR.

Fig. 130. Illustration of Attrition Rates Assumed by FLTPLAN

Choices among these alternatives are made on the basis of which one will produce the largest rate of increase in VALSORTY per nautical mile of low-altitude range required. Figure 131, in two parts, illustrates the general flow of the subroutine. (The off-page connectors "A" and "B" in the diagram are used only by an alternate entry point for the subroutine, which is discussed later.) A more detailed diagram of the subroutine follows (figure 132); part III of this figure consists of notes to facilitate interpretation of this detailed flowchart.

To illustrate how the priorities for this allocation work out mathematically, we note that the cumulative survival probability SURV to route point, i, can be represented as a product of the survival probabilities, S_j, for each leg, j, up to and including the ith. Thus we can rewrite the equation for VALSORTY as follows:

$$V = \sum_{i=1}^{i=n} \left\{ \prod_{j=1}^{j=i} S_j \right\} V_i$$

where V is the value of the sortie and V_i is the value of successfully reaching the ith route point. (This is referred to as the value [illegible], or VALDONE in the program.)

We also note that

$$S_j = e^{-\alpha_j}$$

where α_j is the total attrition on the jth leg. Obviously α_j is a function of L_j, the low-altitude distance allocated to the jth leg.

Differentiating V with respect to L_k, the low altitude allocated to some specific leg k, we obtain

462-546 O - 72 - 17

$$\frac{\partial V}{\partial L_k} = \frac{\partial V}{\partial S_k} \frac{\partial S_k}{\partial \alpha_k} \frac{\partial \alpha_k}{\partial L_k}$$

while

$$\frac{\partial V}{\partial S_k} = \sum_{i=k}^{i=n} \frac{1}{S_k} \left\{ \prod_{j=1}^{j=i} S_j \right\} v_i$$

$$\frac{\partial S_k}{\partial \alpha_k} = -e^{-\alpha_k} = -S_k \quad .$$

Thus

$$\frac{\partial V}{\partial L_k} = - \left[\sum_{i=k}^{i=n} \left\{ \prod_{j=1}^{j=i} S_j \right\} v_i \right] \frac{\partial \alpha_k}{\partial L_k}$$

and separating out the common factors S_j for j=i, k, and noting that

$$\prod_{i=1}^{i=k} S_i = SURV(k)$$

we obtain

$$\frac{\partial V}{\partial L_k} = -SURV(k) \left[\sum_{i=k}^{i=n} \left\{ \prod_{j=k+1}^{j=i} S_j \right\} v_i \right] \frac{\partial \alpha_k}{\partial L_k} \quad .$$

The term in the square bracket is the estimated value of the remainder of the mission, assuming that the aircraft arrives successfully at the point k. (This is called VALON(k) in the program.) Since α_k is the total attrition for the kth leg, the quantity $\partial\alpha_k/\partial L_k$ is simply the difference between high-altitude and low-altitude attrition rates per nautical mile. Moreover, since we are assuming a constant ratio, HILOATTR, between high-altitude and low-altitude attrition rates, this quantity is proportional to the attrition rate. Therefore, we can write:

$$\frac{\partial V}{\partial L_k} = -SURV(k) * VALON(k) * Attrition\ Rate\ (k) * CONSTANT\ .$$

Thus the leg where additional low-altitude range will do the most good can be selected by comparing the product of the first three factors in the above expression for $\partial V/\partial L_k$.

This is the technique used in determining whether the next increment of low-altitude range is to go into the precorridor legs, the leg to the first target, or in extending the low-altitude flight to additional legs or fractions thereof beyond the first target*.

The attrition rate used in this decision process for legs beyond the first target is simply ATLEG(k)/DISTLEG(k); thus the effective attrition rate also reflects any local attrition associated with the kth route point.

The assumed position-dependent attrition rate per nautical mile is used on the leg to target one so that low-altitude range is added to this leg only as far ahead of the target as is justified by the assumed attrition rate.

The attrition rate used in the precorridor legs is the constant value specified in the data base.

It is also worth noting that regardless of which leg, k, receives the final allocation of low altitude, this allocation will correspond to

---

* Actually the values of SURV used in the subroutine during the allocation of the low-altitude flight are all divided by the value of SURV to the first target. This speeds up the operation of the routine since changes in the survival probability in the precorridor legs or on the way to the first targets, as allocations are made to these legs, do not affect the value of SURV which must be used in later legs.

some value for the quantity $\partial V/\partial L_k$. This value, of course, is the marginal value of additional low-altitude range. It can be converted (using the conversion factor RANGEDEC) to obtain a marginal value of additional range or the marginal value of saving distance in the sortie definition. This marginal value of distance, known as VALDIST, is computed by FLTPLAN and used by the EVAL routines to estimate the value of the distance saved in alternative sortie definitions.

The above allocation procedure produces a rigorously optimum allocation of the low-altitude range to the sortie so long as there is no local attrition. However, where local attrition is present at specific targets late in the sortie, a theoretically optimum allocation might allocate limited low-altitude range explicitly for each such target. If this were permitted, it could lead to sorties which unrealistically go low for each defended target and fly high between such targets. To avoid this difficulty the requirement has been imposed that, after passing the corridor origin, a flight is allowed to go low only once.

Moreover, for simplicity of computation during the development of the sortie definitions, the flight is required to go low before the first target, if it is going to fly low at all. Obviously if there is local attrition at a target toward the end of the mission but not at the first target, it might be better to stay high past the first target and save the low-altitude capability to be used in the vicinity of later defended targets. While this possibility is ignored (for computational speed) during the development of the sortie definitions, after the sortie definitions are complete a final check is made. If such a change would increase the estimated value of VALSORTY, the change is incorporated in the final version of the flight plan.

To perform this operation, FLTPLAN is called using the alternate entry point, FINFLT, (see Part II of figure 131). If there are no defended targets beyond where the flight goes high, FINFLT simply returns without changing the sortie. Otherwise FINFLT tries extending the low-altitude capability prior to the first target, and the excess is allocated as before between the leg to the first target and the precorridor legs. If there is no such excess, the point where the aircraft first goes low is set as soon after the first target as possible. The resulting value of VALSORTY is then computed. If the sortie value is increased over that previously obtained, the revised sortie is used. If not, the prior version is retained. This process is repeated until a version of the sortie is tested in which the low-altitude flight is extended to the last defended target. FINFLT then exits with that version of the sortie which produced the best value of VALSORTY.

There is a possibility that when FLTPLAN is originally called for a given sortie, the total range may be inadequate to execute the sortie as defined even if the entire mission were carried out at high altitude. In this case, FLTPLAN exits without assigning low altitude to any of the legs. Moreover, VALSORTY is computed so that it receives no contribution from any route point beyond the maximum range of the aircraft. In this case, later operations by SORTOPT usually result in the omission of some targets that cannot be reached or the elimination of recovery, so that a revised sortie definition is developed which constitutes a feasible sortie.

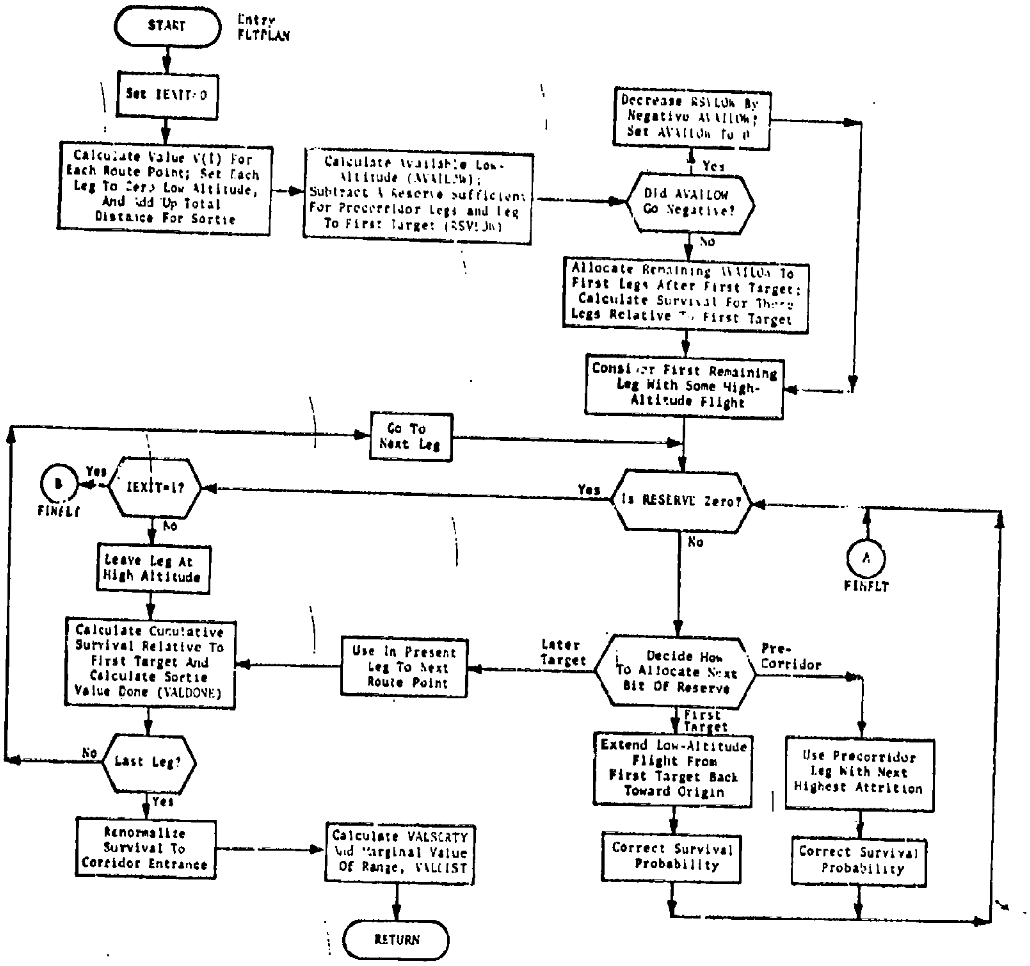


Fig. 131. Subroutine FLTPLAN (Macro Flowchart)
Part I: Entry FLTPLAN

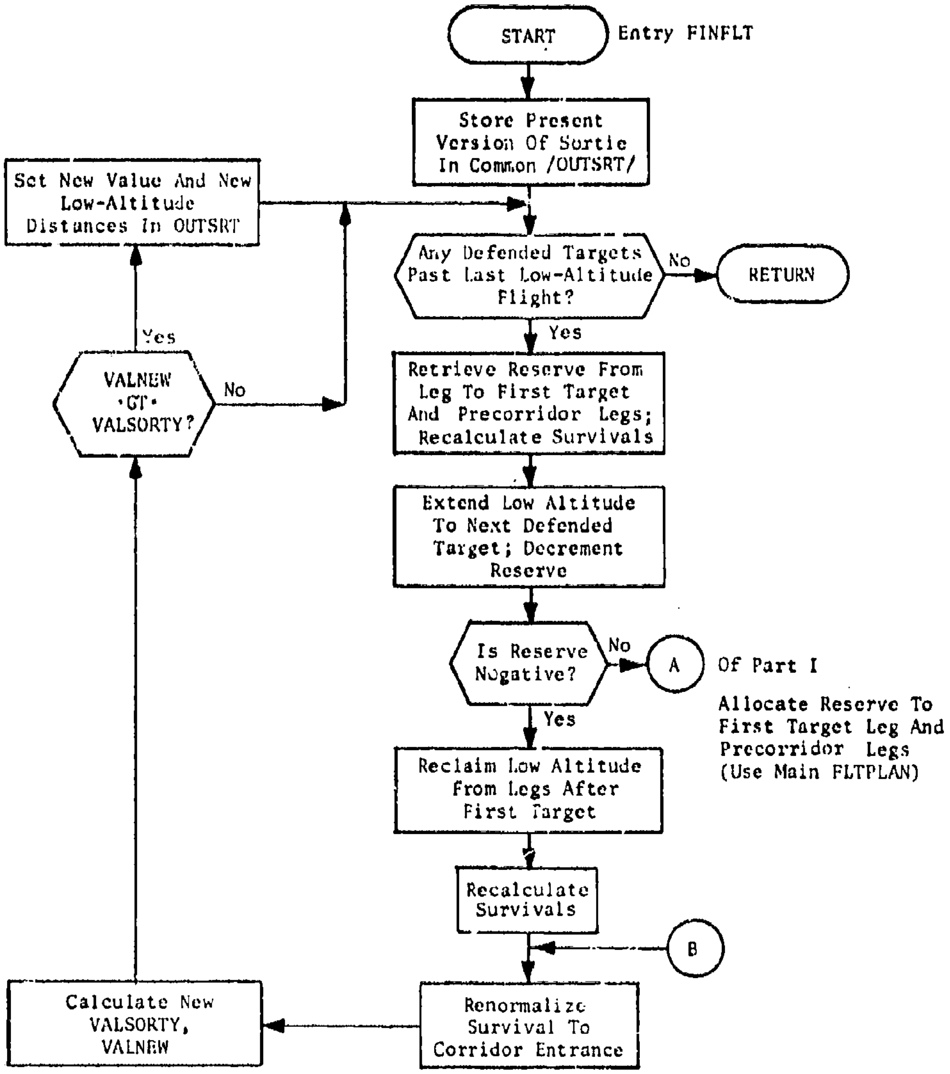


Fig. 131. (cont.)
Part II: Entry FINFLT

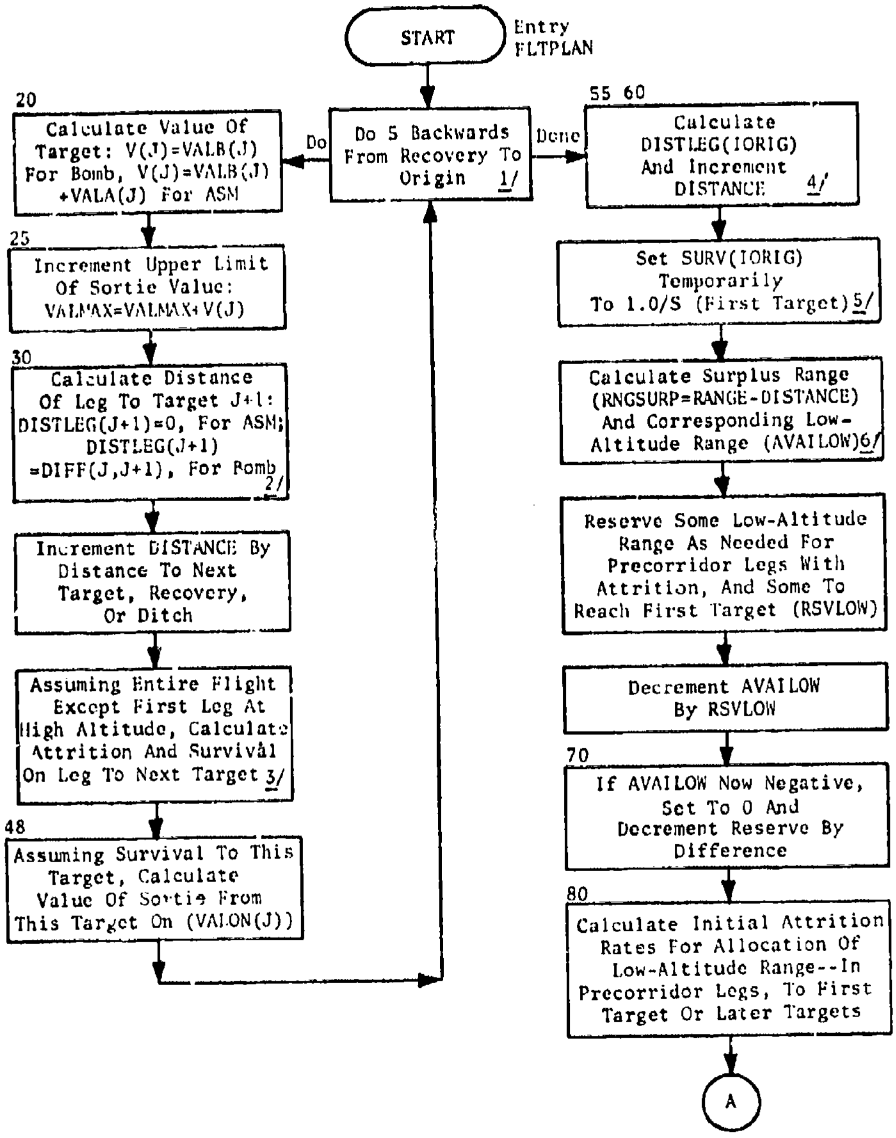


Fig. 132. Subroutine FLTPLAN (Detailed Flow)
Part I: Entry FLTPLAN
(Sheet 1 of 5)

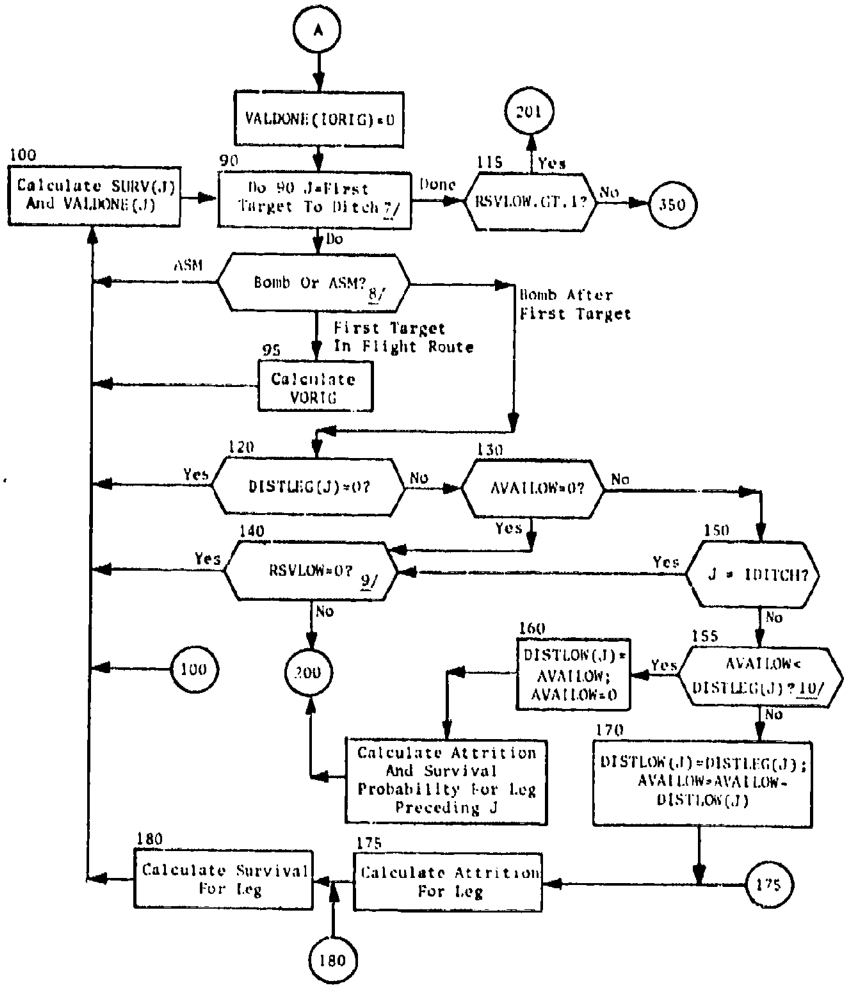


Fig. 132. (cont.)
Part I: (cont.)
(Sheet 2 of 5)

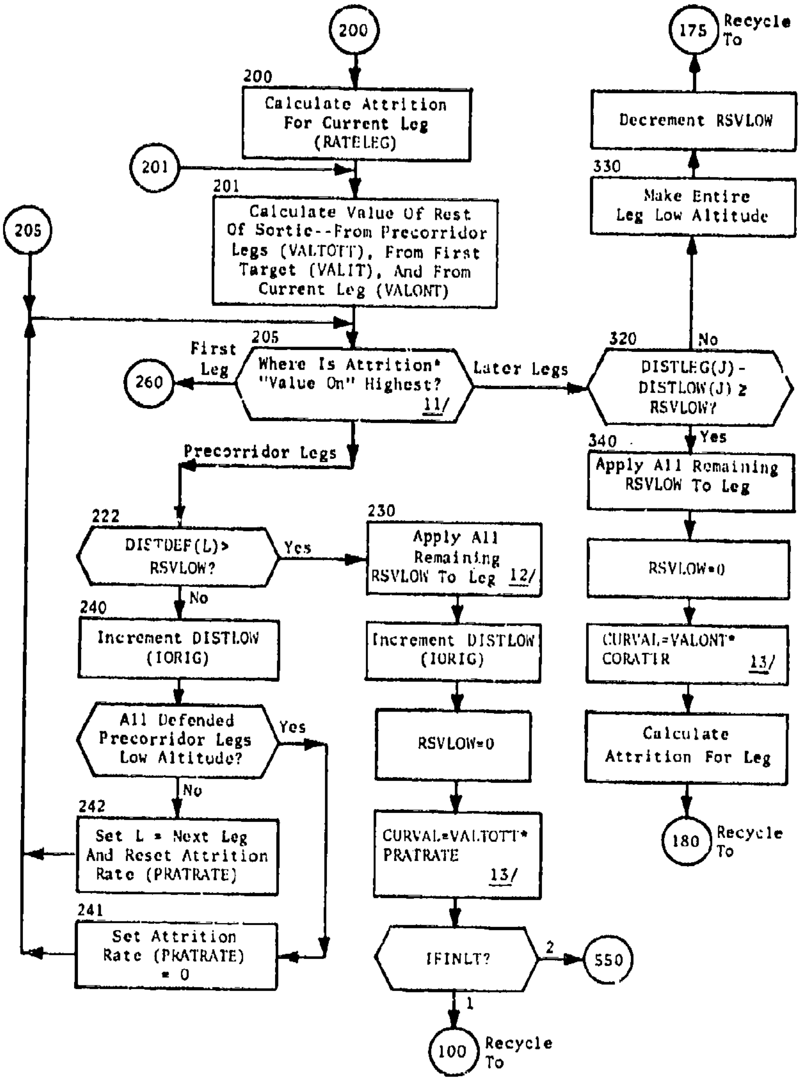


Fig. 132. (cont.)
Part I: (cont.)
(Sheet 3 of 5)

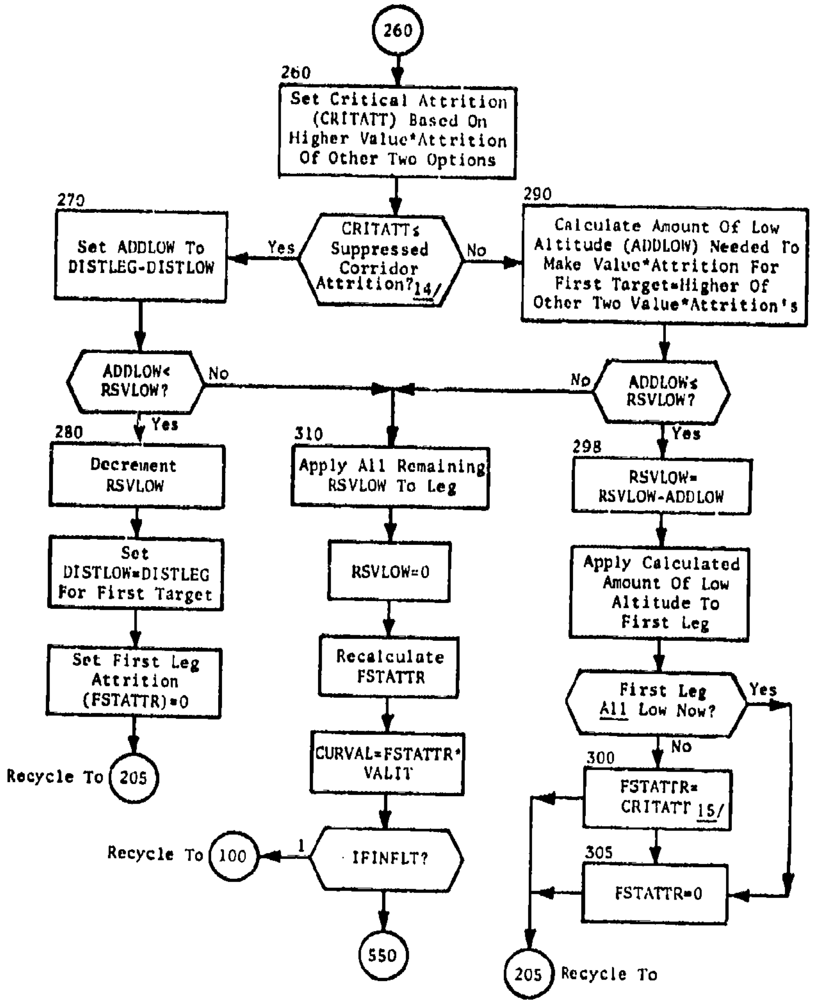


Fig. 132. (cont.)
Part I: (cont.)
(Sheet 4 of 5)

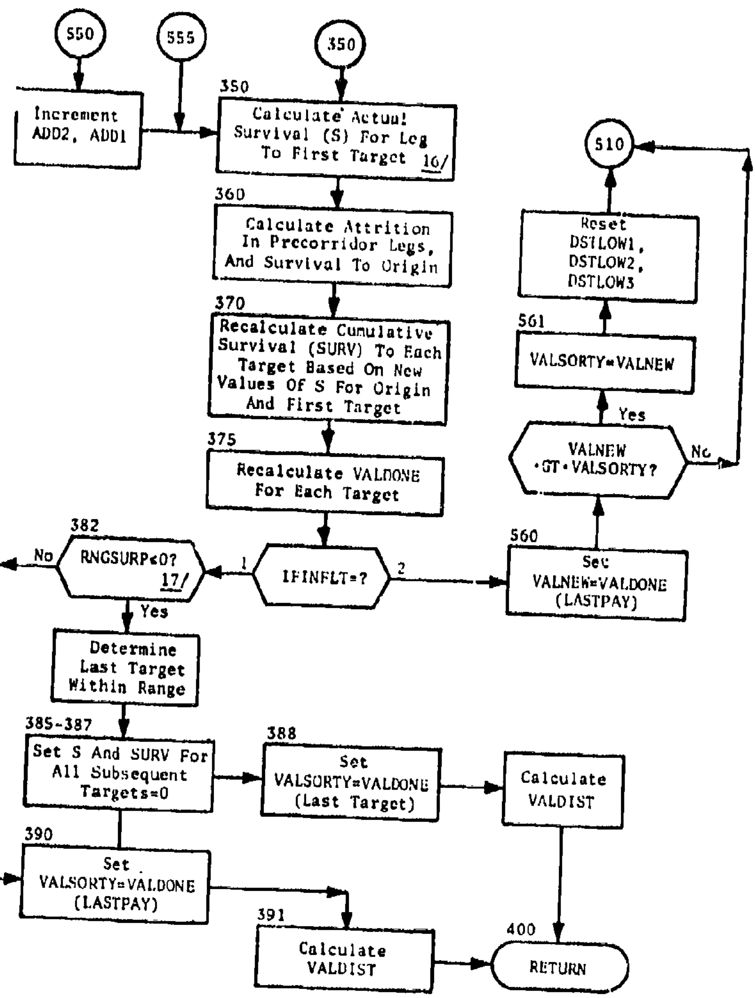


Fig. 132. (cont.)
Part I: (cont.)
(Sheet 5 of 5)

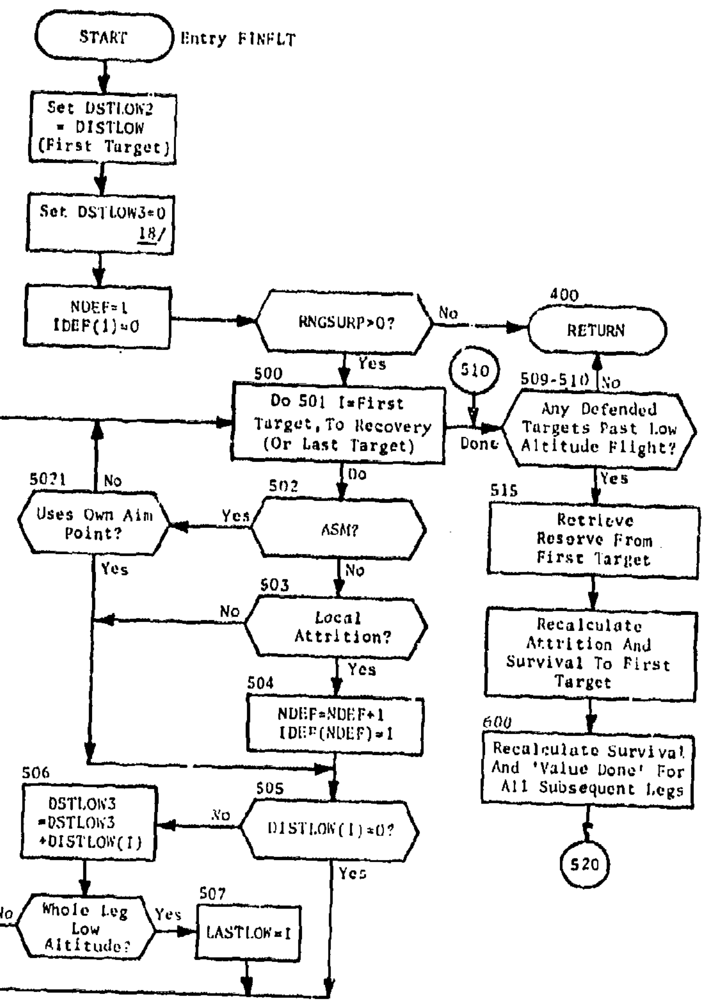


Fig. 132. (cont.)
Part II: Entry FINFLT
(Sheet 1 of 2)

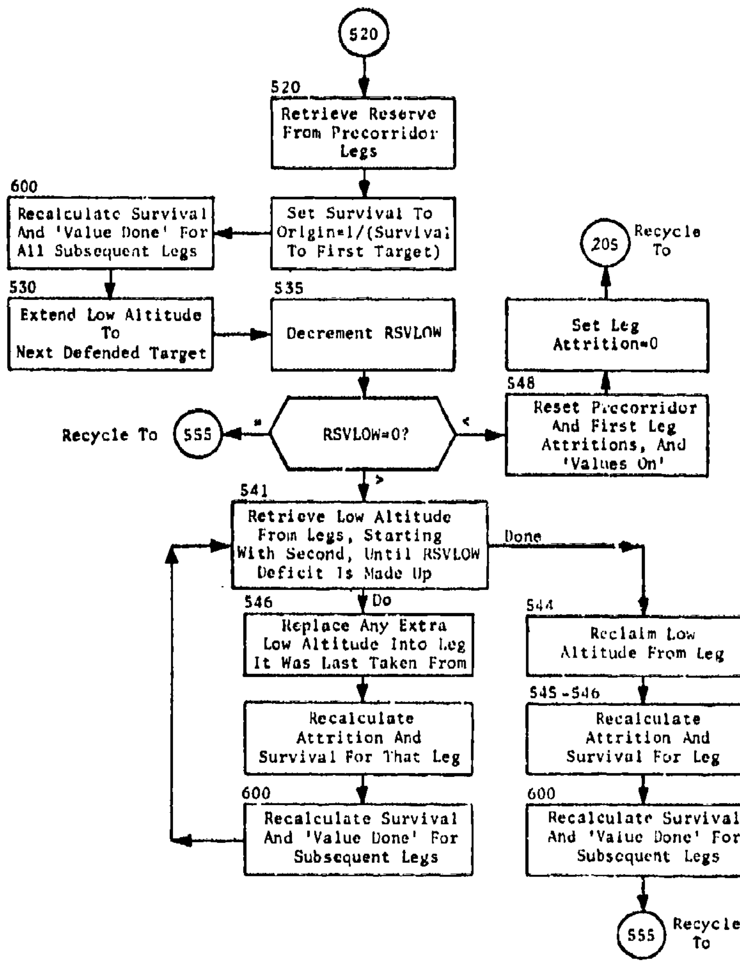


Fig. 132. (cont.)
Part II: (cont.)
(Sheet 2 of 2)

Notes:

1. The implied do loop here actually carries the indices of two consecutive targets, since distance and cumulative values are being computed. JH and JH2 are first set to the last two points in the hit list which will either be recovery, or last target and ditch. If the recovery point is in the hit list, it indicates that recovery is planned. In this event, the distance associated with the recovery point is the distance from the last target to the depenetration point (i.e., the end of the depenetration corridor, or the point past which there is no expected attrition); and the distance associated with "ditch" (which in this case implies "land") is the distance from the depenetration point to the recovery point, or end of flight. There is no attrition associated with this last leg, and no low altitude is ever allocated to it. No survival or value parameters are calculated which have meaning beyond the depenetration point.

   If recovery has been omitted, the ditch point indicates that the vehicle will land as soon as it drops its last bomb. In this case, the distance of the "ditch" leg is 0. In the distance table (D(I,J) in common /SORTYTGT/), the distance from any target to ditch is 0 and the distance from recovery to ditch is DISTREC(IT) - DISTOUT(IT), where IT is the last target in the route.

   The contents of JH and JH2 are the indices of the two consecutive targets or route points in the SORTYTGT arrays. These indices are set negative to represent ASMs. JY and JY2 contain the indices of the corresponding fly points (which may differ from the targets if ASMs are being used).

   Since JH2 is always = 3 on the first cycle of this loop it is not tested for sign, but if JH is negative (indicating an ASM target), it is set positive so that it may be used as an index. (See the section Program Conventions for description of POSTALOC.)

2. At this point, since JH2 will never be negative, we test JY.EQ.JY. If they are equal, we know that JH2 is an ASM target, and DISTANCE(JH2) and S(JH2) are calculated as such (i.e., DISTANCE = 0 and S = 1.0). If JY and JY2 are not equal, either JH2 is a bomb target or it is an ASM target out of range of the rest of the flight route. In either case, DISTANCE (JH2) is calculated as the distance between JY and JY2. On the first cycle through this loop, when JH2 = IRECOVER we also set DISTLEG(IDITCH) equal to the distance from the depenetration point associated with target JH to its corresponding recovery point.

Fig. 132. (cont.)
Part III: Notes
(Sheet 1 of 4)

3. When JY = IORIG, we are dealing with the first-target case, for which attrition is calculated differently from the subsequent legs. IONE and JHONE are saved at this point, as the position and SORTYTGT index of the first target in the sortie.

4. DISTLEG(IORIG) represents the entire distance from the base (or refuel point) to the corridor origin. When buddy refueling or no refueling is used, this is the sum of DISTINK which is the corridor length, and DSTB the distance from the base to the corridor entry point. If a regular refueling area is used, DSTB is zero and DISTINK is the distance from the refuel point to the corridor entry, plus the corridor length. These variables are pre-set for the corridor in INITOPT.

5. The values of SURV used during the allocation of the low-altitude flight are all divided by the value of SURV to the first target. This speeds up the operation of the routine, since changes in the survival probability in the precorridor legs or on the way to the first target will not affect the value of SURV which must be used in later legs.

6. If there is any surplus range, the amount of low-altitude flight to be allocated is calculated by dividing the range surplus (RNGSURP) by the incremental (high-altitude) range used per mile by flying at low altitude.

7. Here again JH and JH2 are indices for a pair of consecutive targets in the hit list. We begin with JH at the origin and JH2 equal to the next point in the hit list.

8. I1, the position of JH2, is compared to IONE, the position of the first bomb target. If it is less than IONE, it must be an ASM. In this case, SURV and VALDONE are calculated immediately and it returns to the beginning of the loop. When I1 is equal to IONE, then all ASMs launched from the origin have been processed and, since the value of these is included in the value of the sortie accomplished upon reaching the origin, VORIG is now calculated, as well as SURV and VALDONE for the first bomb target. When I1 is greater than IONE, the allocation of any available low altitude is begun. We do not move to the next leg until either all available low altitude has been distributed among the three alternatives: precorridor legs, first target leg, or the current JH2 leg, or until the JH2 leg is all low-altitude and we need to move to the next leg as the third alternative for low altitude. If DISTLEG = 0 for I1 greater than IONE, JH2 is again an ASM target and as before SURV and VALDONE are calculated and I1 is incremented.

Fig. 132. (cont.)
Part III: (cont.)
(Sheet 2 of 4)

9. If AVAILOW and RSVLOW are both zero, there is no low altitude to distribute and SURV and VALDONE are calculated for JH2, and all subsequent points in the hit list up to IDITCH.

10. If AVAILOW is positive and if there is enough for the JH2 leg, we fill the leg, decrement AVAILOW, recompute the attrition and survival for the leg based on its low-altitude flight, and continue as before. If AVAILOW is not adequate for the entire leg, we assign as much as there is to the leg and go to the section where the RSVLOW is distributed among the three alternative areas. (Entry E on the flowchart.)

11. When all AVAILOW is used, a decision-making process must be gone through in order to allocate each stretch of low altitude. Values and attrition rates are associated with each of the three alternative areas as follows: VALTOTT is the value of the sortie from the precorridor legs on; PRATRATE contains the attrition rate for the current precorridor leg. VALIT is the value of the sortie from the first target on, and the attrition rate calculated for this leg is stored in FSTATTR. VALONT is the value of the sortie from the current leg to the end of the sortie, and RATELEG is the attrition calculated for the leg. The decision for the allocation of low altitude is made on the basis of the products of these values and attritions.

12. This is done by setting the whole leg to low altitude, decrementing RSVLOW, and then checking to see it RSVLOW is negative. If so, the low-altitude distance DISTLOW(IORIG) is corrected, and RSVLOW is set to zero.

13. CURVAL is set to the current maximum product of attrition and value, and is used to calculate a "value per unit distance" (VALDIST) of low-altitude flight, which is used by the evaluating routine in estimating the effect of route changes on the total value of the sortie.

14. If CRITATT is less than or equal to the corridor attrition rate with defenses suppressed, then the entire first leg is assumed to have a higher attrition rate than any other area (i.e., all other areas with attrition have already been allocated low altitude), so the entire leg is set to low altitude without further calcuations.

15. Actually FSTATTR is set to .999 x CRITATT so that low altitude will next be allocated to the higher of the other two areas, rather than coming back to the first target leg.

Fig. 132. (cont.)
Part III: (cont.)
(Sheet 3 of 4)

462-048 O - 72 - 18

16. Now that all low altitude flight is allocated, we compute the actual survival probability for the leg to the first target, and recalculate the cumulative survival probabilities to later legs. (See note 5.)

17. If RNGSURP < 0, there was not enough range to fly the entire sortie at high altitude. In this event, the sortie plan is not changed by FLTPLAN, but only the targets actually reached are considered in computing VALSORTY.

18. ADD1, ADD2, and ADD3 are used by FINFLT to record changes to DISTLOW1, DSTLOW2, and DSTLOW3. They are actually added to the original distances only if, by doing so, the value of the sortie will be improved.

Fig. 132. (cont.)
Part III: (cont.)
(Sheet 4 of 4)

SUBROUTINE FLTROUTE

PURPOSE:	Prepares for and controls the assignment of targets to sorties, specifying the launch bases and determining which side of the corridor each flight of vehicles is to fly down.
ENTRY POINTS:	FLTROUTE. FLTPASS
FORMAL PARAMETERS:	None
COMMON BLOCKS:	2, 3, ARAYSIZE, CURRAID, DEBUG, FLTPASS, GRPDATA IRESRCH, KEYS, MASTER, NEXTFLT, PCALL, PRINTOPT, RAIDSHR, TGTASGN
SUBROUTINES CALLED:	IGET, NEXTFLT, PRINTIT, PRNTF, TGTASGN, TIMEME
CALLED BY:	GENRAID

Method

For each successive flight assigned to the corridor, FLTROUTE assigns its sorties to the right or left of the corridor. This subroutine is responsible for the initial generation of the skeletal description of all sorties in a raid (or sub-raid, if there are more than 100 vehicles in the raid). It stores this description in common block /CURRAID/ (Current Raid). For each sortie in the raid (or sub-raid) the subroutine records the base of origin, the number of strikes assigned, a list of indices to the targets assigned, and the vehicle index for the particular vehicle on the base.

Figure 133 illustrates the operation of subroutine FLTROUTE. Sheets 3 and 4 of this figure consist of notes to facilitate interpretation of this flowchart. It first calculates the number of vehicles for the corridor NVEH by determining, base by base, the number of vehicles required to transport at least the desired number of weapons for the corridor. So that PRERAID can keep its records correctly, the number of weapons for the corridor is then reset to the actual number of warheads on these vehicles.

If the number of bases involved in the raid is less than four, FLTROUTE sets a flag which causes NEXTFLT to deliver flights of only half a base load at a time. This makes it possible to maintain better time coordination between sorties assigned to opposite sides of the corridor.

If the number of vehicles would exceed 100, provision is made to process them in several passes of 100 vehicles or less each.

Proceeding to the second sheet of figure 133, all sorties in CURRAID are initialized to zero weapons assigned. The sorties for next flight are then assigned; the assignment is from the top or the bottom of the list of unassigned sorties, depending on which side of the corridor gives the higher absolute value of PHI. At the same time a flag (ISIDE = 1 or 2) is set so that TGTASGN will know in which order to assign targets.

Finally, the base of origin for the flight is recorded for all sorties assigned to the flight, and a vehicle index is assigned for each sortie. Then TGTASGN is called to actually assign specific targets for each sortie in the flight.

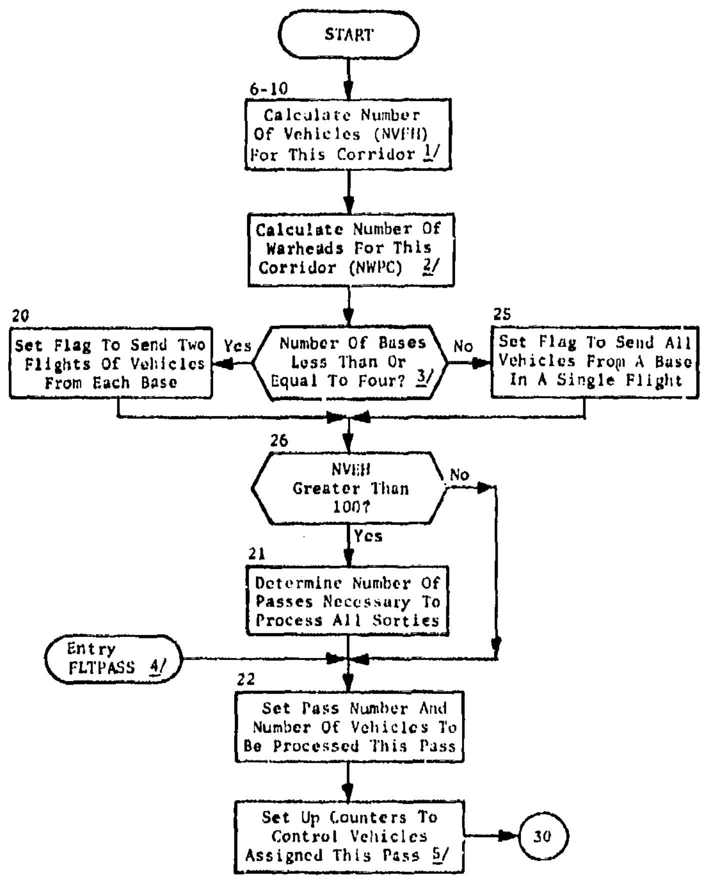


Fig. 133. Subroutine FLTROUTE
(Sheet 1 of 4)

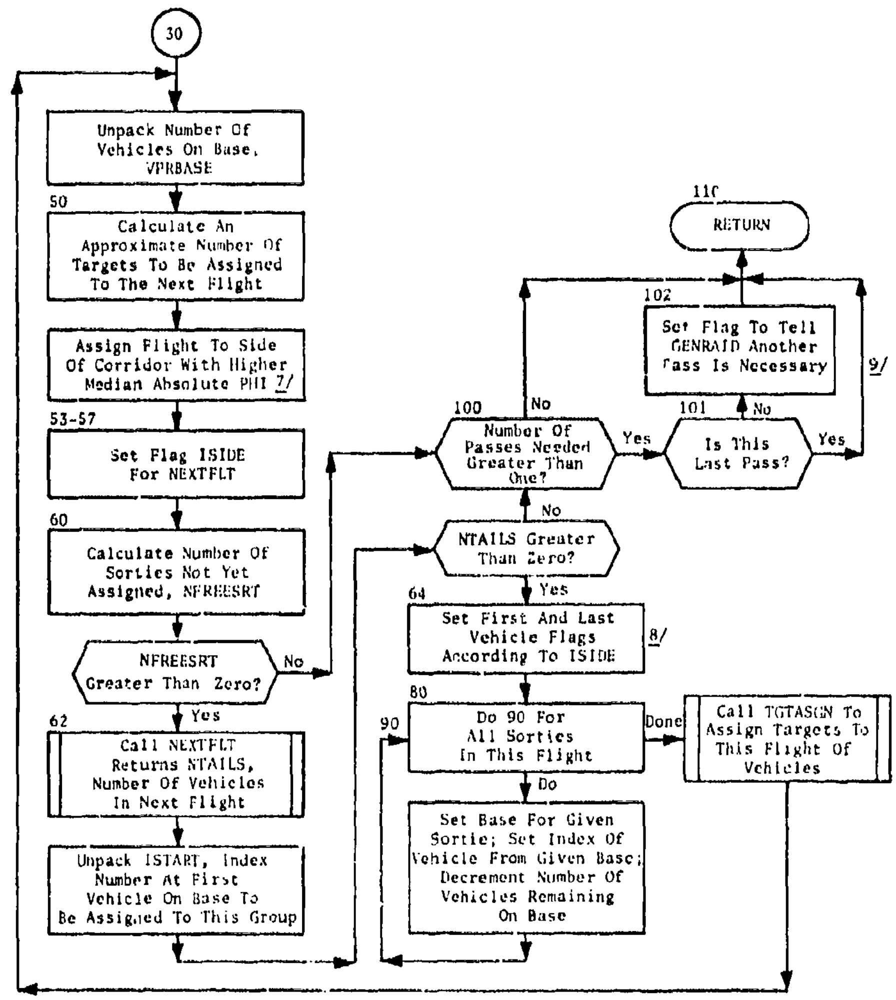


Fig. 133. (cont.)
(Sheet 2 of 4)

Notes

1. Given the total number of weapons for a given corridor, and the number of weapons that can be carried on each vehicle, FLTROUTE computes the number of vehicles to be sent down the corridor (NVEH).

2. If the last vehicle ends up not carrying its maximum number of weapons, it is set to do so and NWPC is increased accordingly. Thus the first corridor, which is the one with the most strikes assigned, may end up with slightly more than its "share" of weapons, which was the original NWPC computed by PRERAID. This has the effect of leaving more targets per weapon to choose from for the sorties in the more sparsely targeted corridors. The average number of targets per warhead (TGTSWHD) is also calculated at this point.

3. It is desired to send approximately half the vehicles down either side. If there are more than four launch bases, it will suffice to merely alternate bases from one side to the other. If there are four or fewer bases, half the vehicles from each base are sent down either side of the corridor to provide an even distribution.

4. Since FLTROUTE and the subsequent routines are equipped to process only 100 sorties at a time, it is sometimes necessary to make more than one pass to complete the processing. Entry point FLTPASS is used for the second and subsequent calls on FLTROUTE within the same corridor.

5. The sorties are processed working from both ends of the list, in order to alternate from the left to the right side of the corridor. Thus, if there is difficulty in making the final assignments, these sorties will fall in the middle of the list and there will be a better chance of finding suitable targets by interchanging with the sorties on either side. The sorties from the beginning of the list are sent down the right side of the corridor and those from the end are sent down the left side. ILFTRSRT and IRTSRT are set to indicate the first sortie not yet assigned from either end of the list. NFREESRT indicates the number of vehicles not yet assigned.

6. The target-to-sortie assignments are made by "flights" which consist of either part or all of the bombers from a given base. NEXTFLT returns the size of the flight (NTAILS) and its launch base.

Fig. 133. (cont.)
(Sheet 3 of 4)

7. IFSTGT and ILASTGT are markers in the target list indicating the first and last unassigned targets in the list. The median value of PHI is taken, from the number of targets to be assigned, at both ends of the list of unassigned targets. The side with the higher absolute value of PHI is designated for the next flight. (Thus, if one side of the corridor has more targets than the other, the depth of penetration is still kept fairly even on either side.)

8. Before the target assignments for the individual sorties are made, IFSTVEH and LSTVEH are set to the first and last index numbers of the vehicles in this flight, and ILFTSRT or IRTSRT is decremented or incremented as the case may be.

9. If another pass is necessary to finish target-to-sortie assignments for the corridor, NTAILS is set to 1000 as a flag for GENRAID. In the event that NTAILS was left as a negative number, -1000 is added to its former value.

Fig. 133. (cont.)
(Sheet 4 of 4)

SUBROUTINE GENRAID

PURPOSE: Prepares for and controls the processing of all sorties through a given penetration corridor.

ENTRY POINTS: GENRAID

FORMAL PARAMETERS: None

COMMON BLOCKS: 2, 3, 4, ARAYSIZE, BEGIN, CORRCHAR, CORRIDOR, DEBUG, DPENREF, FILES, FIXALL, FIXRANGE, GRPDATA, IFTPRNT, INDEX, INPUTFL, ISKIPTO, ITP, NEXTFLT, PCALL, PRINTOPT, RAIDSHR

SUBROUTINES CALLED: CENTROID, CORRPARM, DISTF, FLTPASS, FLTROUTE, NOCORR, OPTRAID, ORDER, PRINTIT, PRNTF, RDARRAY, REORDER, SETFLAG

CALLED BY: PRERAID

Method

GENRAID is the controlling subroutine for the generation of a raid in a corridor. It reads in the specific list of strikes for the corridor from the ALOCGRP or TMPALOC file and places the available bases and the strikes in proper order so that the assignment of strikes to vehicles can be accomplished by subroutine FLTROUTE. When the initial assignment is complete it calls subroutine OPTRAID to optimize all resulting sorties.

Figure 134 illustrates the operation of GENRAID.

First, GENRAID reads the target data for strikes assigned to the corridor to fill common /3/. It moves the corridor characteristics for the corridor JCORR, named in common /3/, to common /CORRIDOR/ in an unindexed form for easier reference. For each corridor where strikes have been assigned, GENRAID places the bases for the group in order of increasing distance from the corridor entrance. (If weapons from the same group have been previously assigned, bases with remaining aircraft may be mixed with bases from which all aircraft have been assigned; to avoid this, routines which process bases always check numbers of remaining aircraft on each base.) The result, however, is that bases close to each corridor are always processed first (so long as aircraft remain on the bases). In the case of the tactical bombers (JCORR = 1 or 2 indicates tactical bombers which use no penetration corridor) or bombers assigned to naval targets, subroutine NOCORR is called to initialize; then CORRPARM is called to order the bases in the same way that the targets will be ordered.

GENRAID next calls CORRPARM to assign values of PHI and RHO to each target and the targets are sorted on the values of PHI.

FLTROUTE is then called to assign the strikes to aircraft beginning with the nearest bases and proceeding until all strikes in the corridor have been assigned. Finally OPTRAID is called to optimize the sorties in the raid.

However, due to computer memory limitations, FLTROUTE and OPTRAID cannot handle more than 100 sorties in a single raid. While such large raids are unusual, they can occur. Therefore, a spill provision is included. If FLTROUTE is called for a larger raid, it will return when 100 sorties have been set up. Thus, after OPTRAID, a test is included to be sure all sorties for the corridor have been processed. If not, FLTROUTE is called to continue processing sorties for the corridor and OPTRAID is again called to optimize the sorties. The entrance point for such a continuation is called FLTPASS.

The following points should be noted with regard to subroutine GENRAID:

1. When a penetration corridor is to be used, the launch bases are ordered so that sorties are processed beginning with the nearest base and working back to the farthest. In the case of tactical bombers, the bases are assigned values of RHO and PHI relative to a line between the centroid of the bases and the centroid of the targets. The bases are then arranged in order of PHI.

2. The targets are ordered so that those closest to the corridor origin are hit first. (See Raid Generation in POSTALOC of this manual for further discussion of this.)

3. The current array dimensions limit POSTALOC to processing no more than 100 sorties at once. If more than 100 sorties from a weapon group are to go through the same corridor, it must be done in more than one pass. FLTPASS is a second entry point in FLTROUTE, and is used for the second and subsequent passes. NTAILS, a variable usually used to tell FLTROUTE the number of vehicles in the next flight, is set to 1,000 by FLTROUTE to indicate to GENRAID that another pass is necessary. If NTAILS is negative at the end of FLTROUTE, indicating an error in count of vehicles, it is set to -1000+NTAILS if another pass is called for.

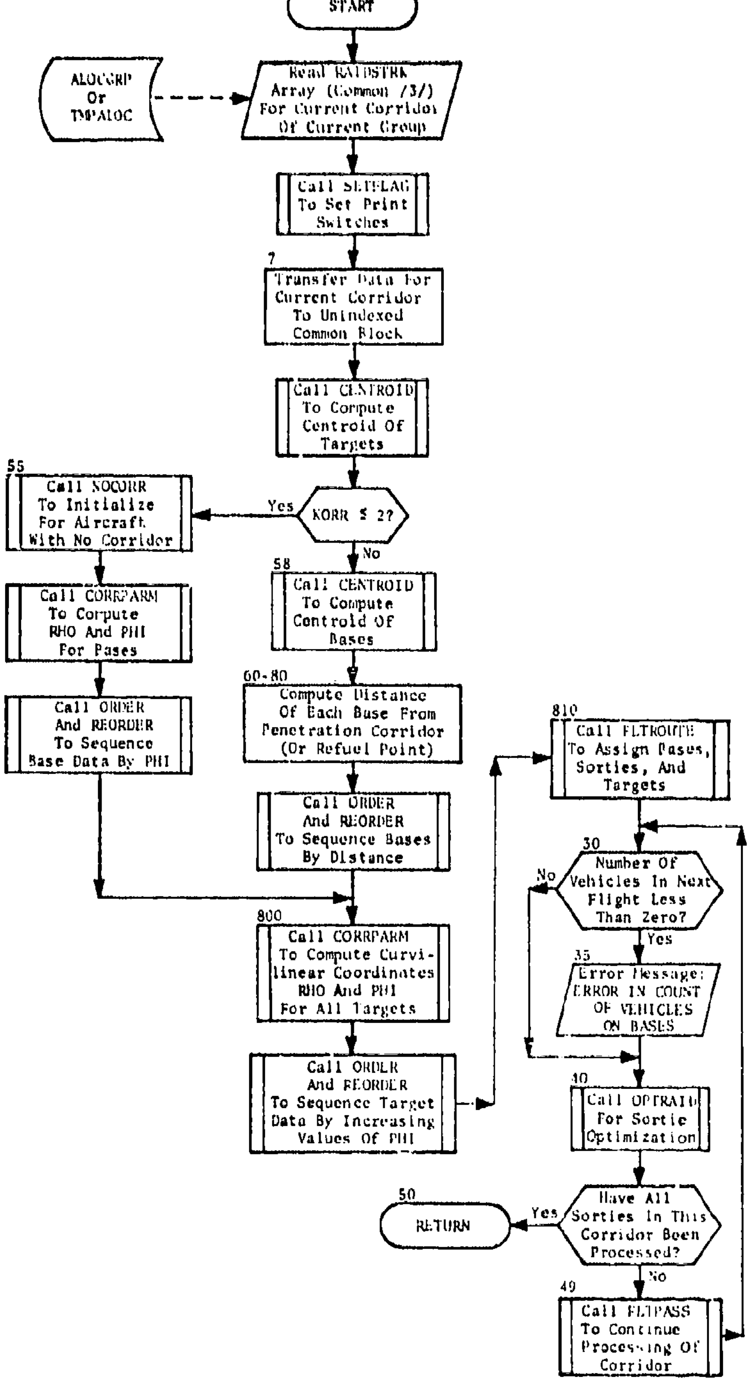


Fig. 134. Subroutine GENRAID

SUBROUTINE GETGROUP

PURPOSE: To read the BASFILE, one weapon group at a time, and call PRERAID or MISASGN to process the bomber or missile group.

ENTRY POINTS: GETGROUP

FORMAL PARAMETERS: None

COMMON BLOCKS: 1, ARAYSIZE, ASMTABLE, CONTROL, CORRCHAR, DEBUG, DPENREF, FILABEL, FILES, GRPDATA, GRPTYPE, IDUMP, IFTNO, IFTPRNT, INPUTFL, ISKIPTO, ITP, MASTER, MYIDENT, MYLABEL, NOPRINT, PAYLOAD, PCALL, PLANTYPE, PRINTOPT, STRKSUM, TWORD, VAL

SUBROUTINES CALLED: ABORT, DEACTIV, INITAPE, MISASGN, OUTSRT, PRERAID, PRINTIT, PRNTF, RDARRAY, RDWORD, SETREAD, SETWRITE, TERMTAPE

CALLED BY: POSTALOC

Method

GETGROUP calls the filehandler initializing routine INITAPE; calls SETREAD or SETWRITE for each input or output file to be used by the program; and reads the BASFILE to fill common blocks /MASTER/, /FILES/, /CORRCHAR/, /ASMTABLE/, /PAYLOAD/, /DPENREF/, and /PLANTYPE/.

Then from the BASFILE, the first weapon group data are read into common /GRPDATA/, and the weapon type for the first group are read into common /GRPTYPE/.

ICLASS, a variable in /GRPTYPE/, is then tested to see whether the group is missile or bomber. If it is a missile group, MISASGN is called; if it is a bomber, PRERAID is called.

When all groups have been processed, IGROUP is set to 201 as a flag, and OUTSRT is called to write a record on the output file STRKFILE. The filehandler terminator TERMTAPE is then called for all files, and the subroutine returns.

Subroutine GETGROUP is illustrated in figure 135.

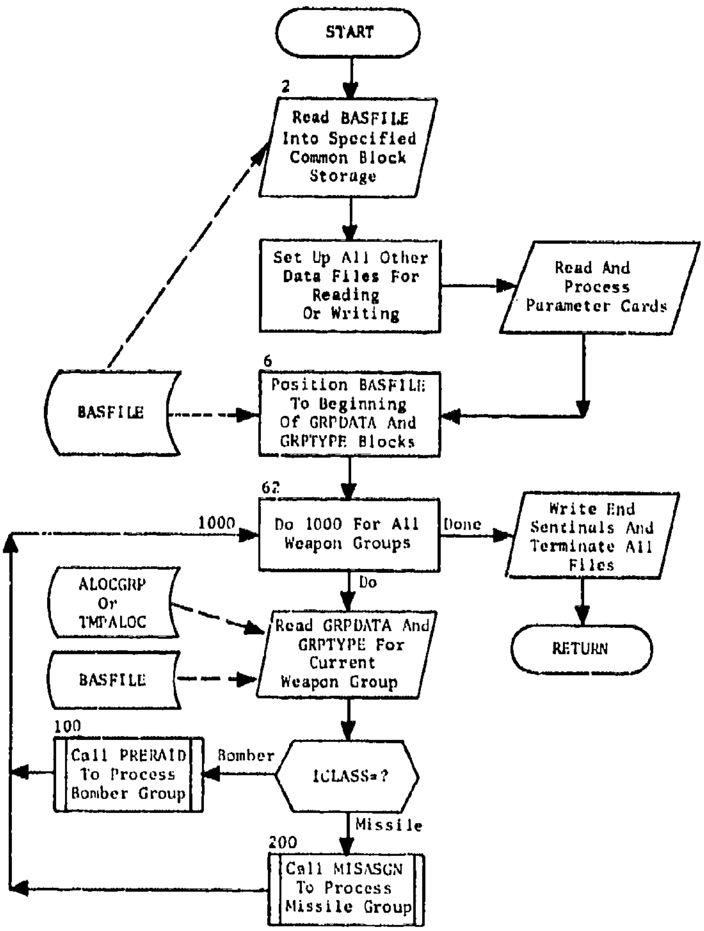


Fig. 135. Subroutine GETGROUP

SUBROUTINE GETSORT

<u>PURPOSE</u>:	To bring into the potential target arrays the targets currently assigned to the sortie in the JTGT array.
<u>ENTRY POINTS</u>:	GETSORT
<u>FORMAL PARAMETERS</u>:	None
<u>COMMON BLOCKS</u>:	ASMTABLE, CHGPLAN, CORRIDOR, CURRAID, CURSORTY, DEBUG, DPENREF, FIXRANGE, FLTPASS, GRPDATA, GRPTYPE, INDEX, INITOPT, PAYLOAD, PCALL, PRINTOPT, RAIDSHR, SORTYTGT, VAL, 3
<u>SUBROUTINES CALLED</u>:	CHGPLAN, DIFF, INPOTGT, OUTPOTGT, PRINTIT, PRNTF
<u>CALLED BY</u>:	OPTRAID

Method

GETSORT searches for any unassigned targets in the portion of the target list with which it is currently working. If it finds any, they are placed in the LOSTTGT array. It then locates the least valuable targets in the omit list, and drops the lower half of the list by calling subroutine OUTPOTGT for each target to be dropped. The targets listed in the JTGT array, having been placed there by TGTASGN or after the first optimization pass, are brought into the SORTYTGT arrays by calling INPOTGT for each one. If there is extra space in the SORTYTGT arrays, some of the unassigned targets may be brought in from the LOSTTGT array at this time.

Subroutine GETSORT is illustrated in figure 136; sheets 3 and 4 of the figure are notes to facilitate interpretation of the flowchart.

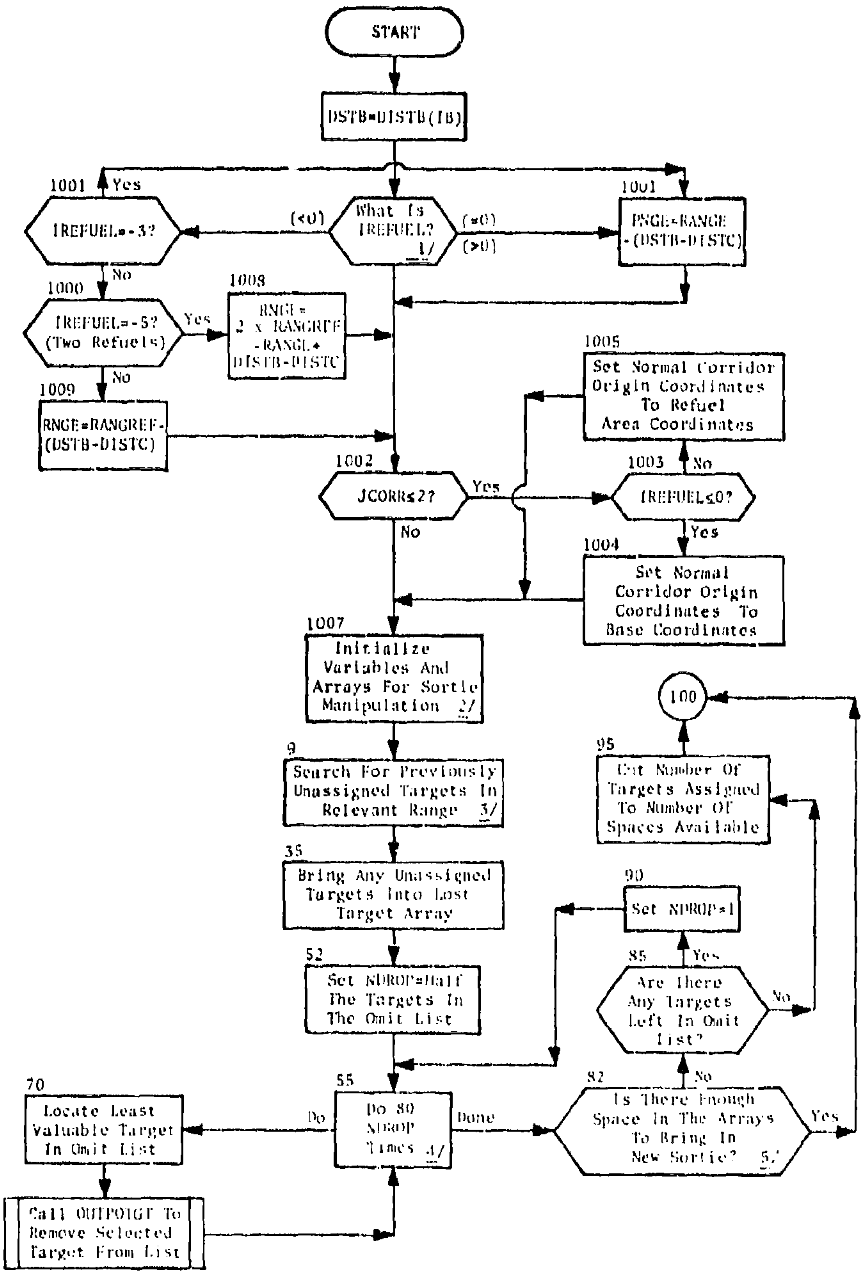


Fig. 136. Subroutine GETSORT
(Sheet 1 of 4)

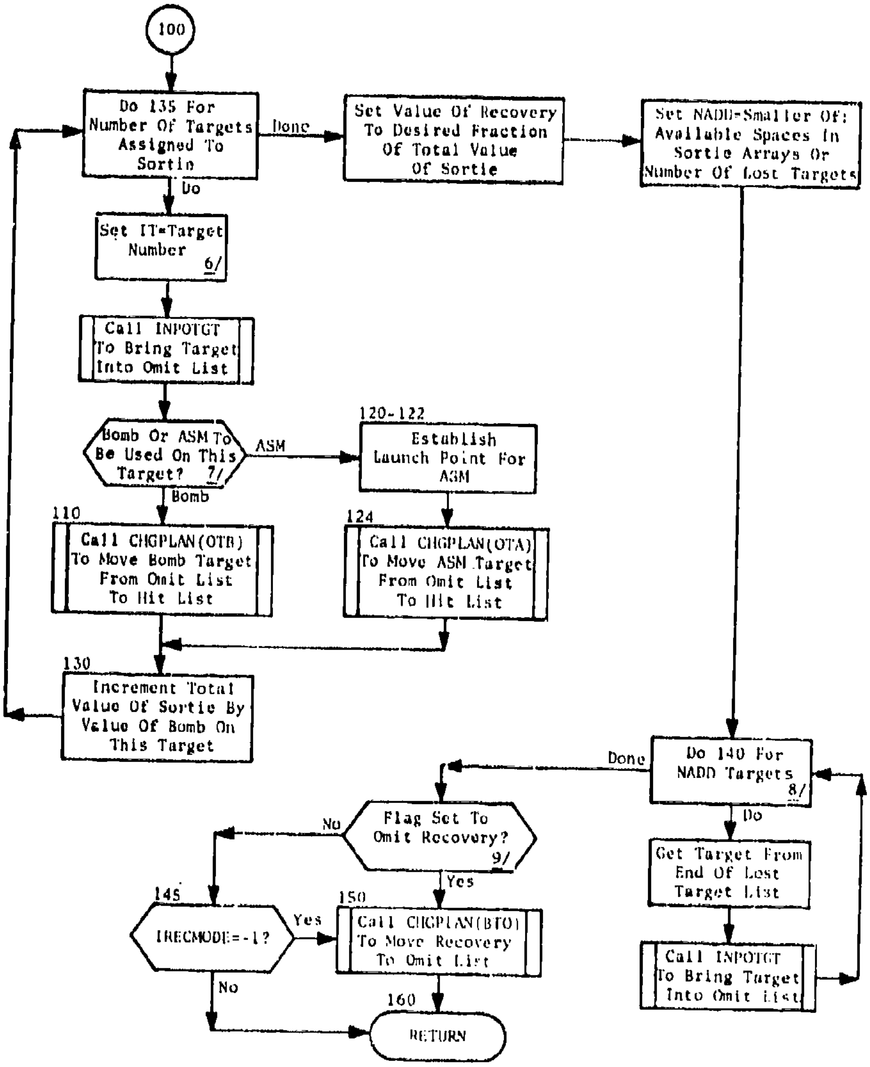


Fig. 136. (cont.)
(Sheet 2 of 4)

Notes

1. The problem of having insufficient range to reach a single target arises occasionally, because the allocator uses the centroid of the bases when it is allocating targets to a group. To avoid this problem temporarily, the range which is set for each sortie, RNGE, is adjusted to compensate for the difference between DSTB and DISTC, where DSTB is the distance of that sortie's base from the corridor entry point, and DISTC is the distance of the centroid from the entry point. (The problem does not exist when area refueling is done.)

2. In addition to DSTB, the following unindexed variables are extracted from the indexed arrays to facilitate frequent reference during single-sortie processing:

NWHDS:	Number of warheads on this vehicle
NASMS:	Number of warheads which are ASMs
RNGASM:	Range of this ASM type.

Also, the following variables in the sortie description (common /CURSORTY/) are initialized:

NUMHIT	=0	Number of targets being hit with bombs or ASMs
NUMBOMB	=0	Number of targets being hit with bombs
NUMASM	=0	Number of targets being hit with ASMs
LASTTGT	=1	Position in hit list of last target
LASTPAY	=2	Position in hit list of last "paying" route point
NHIT	=3	Number of route points in hit list

IHIT(1)=IFLY(1)=IORIG=1

IHIT(2)=IFLY(2)=IRECOVER=2

IHIT(3)=IFLY(3)=IDITCH=3

LKHITMT(IORIG)=1

LKHITMT(IRECOVER)=2

LKHITMT(IDITCH)=3

Fig. 136. (cont.)
(Sheet 3 of 4)

3. ITNEW is set to the RAIDSTRK index of the first target in JTGT array for the current sortie. ITOLD contains the index of the first target from the last sortie. All of the targets in the RAIDSTRK array between and including these two are checked to see if any have not been assigned; i.e., MYASGN (IT)=0. If so, these targets are placed in the LOSTTGT array, NLOSTTGT is incremented, and MYASGN (IT) is set to 1. Notice that if MYASGN (IT)= -1, the target has not been assigned because it is in the group to be processed on the next FLTPASS call; thus it is not considered as a lost target.

4. To avoid keeping targets in the omit list that are completely out of range, the targets are scanned and the half that had the lowest evaluations, with respect to the last sortie, are dropped. This provides additional space in the potential target (SORTYTGT) arrays.

5. If there is not enough space in the potential target arrays to bring in all the targets assigned to the sortie in the JTGT array, some more targets must be dropped from the omit list, or the number of targets assigned must be arbitrarily cut to the space available.

6. Each target currently stored in the JTGT array of common /CURRAID/ for the current sortie is brought into the SORTYTGT arrays by calling INPOTGT.

7. If the target in the JTGT array is negative, it is set to be hit with an ASM. (This will occur only on the second pass.)

8. If there are spaces left in the SORTYTGT arrays, and if there are lost targets, these targets are brought in by calling INPOTGT until either space or targets run out.

9. If recovery is omitted for a sortie on the first processing pass, NASGN is set negative so that GETSORT can omit it on the second pass, since the recovery point is not shown in the JTGT array. Also, if IRECMODE is -1, the bombers are unmanned "air-breathing missiles" which are not intended to recover. In this case, GETSORT removes recovery from the hit list on both passes.

Fig. 136. (cont.)
(Sheet 4 of 4)

SUBROUTINE INITOPT

PURPOSE: To initialize or clear the variables in commons /SORTYTGT/, CURSORTY/, and /INITOPT/, in preparation for the optimization of all sorties in the raid.

ENTRY POINTS: INITOPT

FORMAL PARAMETERS: None

COMMON BLOCKS: ARAYSIZE, CORRIDOR, CURSORTY, DEBUG, DPENREF, FIXRANGE, GRPDATA, GRPTYPE, INDEX, INITOPT, PCALL, PRINTOPT, SORTYTGT, 3

SUBROUTINES CALLED: DISTF, ORDER, PRINTIT

CALLED BY: OPTRAID

Method

A variable, MAXPT, is set first to indicate the maximum number of targets to be allowed in the potential target arrays in common /SORTYTGT/ (elsewhere referred to as "SORTYTGT arrays"). This number is the sum of: three, representing the number of basic route points (origin, recovery, and ditch or land; NSPARE, the number of extra targets to be considered as alternatives (NSPARE=4); and NWHDS, the maximum number of warheads carried on any one vehicle from any of the bases in the group. Thus if the number of warheads is 4, MAXPT is 11. (MAXPT may not exceed the dimensions of the arrays, MAXPA, currently set at 25.)

The first MAXPT cells of the SORTYTGT arrays are then either cleared or preset. The following variables in commons /SORTYTGT/ and /CURSORTY/ are initialized to something other than 0:

NHIT = 3	Representing the three basic route points: origin, recovery, and ditch or land

NAVAIL = MAXPT -3	Representing the remaining available spaces in the SORTYTGT arrays for potential targets
DVALA(J), DVALB(J) = 1E200	These variables are often used in inequality tests and may assume legitimate negative values
D(J1, J2) = -10	Where J1 and J2 are targets, since some distances are actually computed as 0
D(IORIG, IRECOVER) = 20000	So that if any sortie ever considers flying directly from origin to recovery without hitting any targets, it will be discarded as a bad idea
RHOJ(IORIG) = 0 RHOJ(IRECOVER) = 1E200 RHOJ(IDITCH) = 2E200	To assure proper positioning when the sortie hit list is ordered by RHO
IAVAIL(1,....,NAVAIL)=MAXPT,....,4	So that IAVAIL(NAVAIL) always contains the index of the next available SORTYTGT index.

Subroutine INITOPT is illustrated in figure 137.

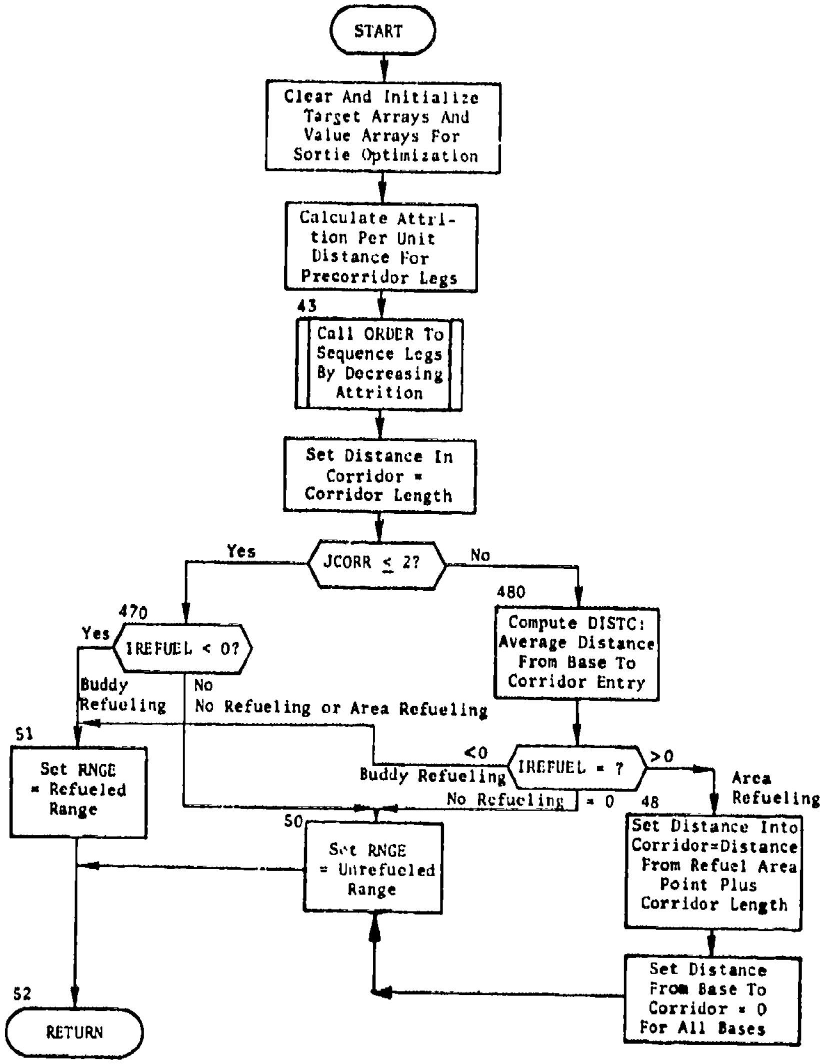


Fig. 137. Subroutine INITOPT

SUBROUTINE INPOTGT

PURPOSE:	To bring a target into the next available cell of the potential target arrays and record in omit list.
ENTRY POINTS:	INPOTGT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	CORRIDOR, 2, CURRAID, CURSORTY, DEBUG, FIXALL INDEX, PCALL, 3, PRINTOPT, SORTYTGT
SUBROUTINES CALLED:	DISTF, PRINTIT
CALLED BY:	GETSORT, REFABORT

Method

INPOTGT receives from GETSORT the RAIDSTRK index IT in common /INDEX/ for the target to be brought into the SORTYTGT list. It enters the index and transfers the target values, local attrition, and RHO coordinate from the complete list of targets (common /3/) to the smaller, more selective list for the current sortie (common /SORTYTGT/).

Subroutine INPOTGT is illustrated in figure 138, which includes notes to facilitate interpretation of the flowchart.

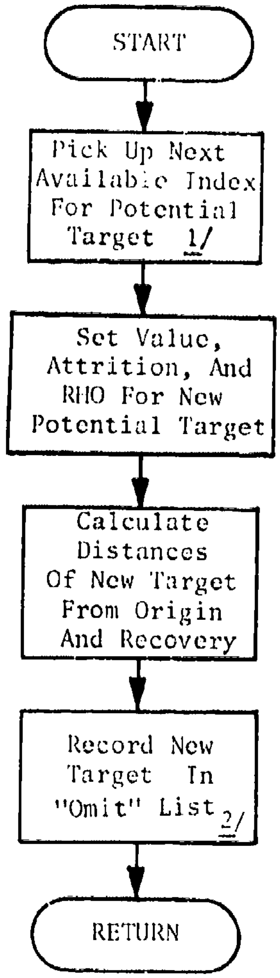


Fig. 138. Subroutine INPOTGT (Sheet 1 of 2)

Notes:

1. Since the indices of the available cells in the SORTYTGT arrays are stored in reverse order in array IAVAIL, and NAVAIL contains the number of cells currently available, IAVAIL (NAVAIL) will at any time contain the index of the next available cell, in this case JX.

   Before bringing in a new target, the assignment status (MYASGN) of the target previously occupying the space must be reset to 1. DUMPSRT has already recorded the targets assigned to the previous sortie in the JTGT array and has entered the SORTYTGT indices of those targets in the AVAIL list, but has not changed their status in the MYASGN list. A check must be made, however, to be sure that the target was a "legitimate" SORTYTGT entry (i.e., MYASGN=2), and not one that was brought in temporarily by REFABORT, which may not have even been reached in the normal target assignment (i.e., MYASGN=-1).

   To save time by avoiding frequent referencing of variables in common, a local variable is often set to the common variable at the beginning of a subroutine. Thus ITX is set equal to IT.

   The RAIDSTRK index ITX is stored in MYPOTGT(JX), and MYASGN(ITX) is set to 2. The distance array for the new target is cleared to -10.0.

2. LKHITMT(JX) gives the position of MYPOTGT(JX) in the hit or omit list. To indicate that the target is in the omit list rather than the hit list, the omit list index is made negative.

Fig. 138. (cont.)
(Sheet 2 of 2)

SUBROUTINE MISASGN

PURPOSE:	To generate and write missile events for each missile weapon group and its assigned targets.
ENTRY POINTS:	MISASGN
FORMAL PARAMETERS:	None
COMMON BLOCKS:	3, FILES, GRPDATA, GRPTYPE, IFTPRNT, INPUTFL, ISKIPTO, ITP, MASTER, MISPRNT, PAYLOAD, PCALL, PLANTYPE
SUBROUTINES CALLED:	ABORT, DISTF, PRINTIT, POSIT, RDARRAY, WRARRAY
CALLED BY:	GETGROUP

Method

Prior to calling MISASGN, the weapon group data must be stored in commons /GRPDATA/ and /GRPTYPE/. The target data are read from the ALOCGRP or TMPALOC file and stored in common /3/. From these data the missile events are generated in the format shown in the STRKFILE record format and written on the STRKFILE.

A maximum of 18 missiles can be assigned to a single event which, in most cases, is sufficient for planting the launches for one squadron. However, if a squadron contains more than 18 weapons (or re-entry vehicles in the case of MIRV groups), the number of events required to output all the weapons is computed. This number times the number of squadrons in the group gives the total number of events to be generated for the group. (For missiles carrying MIRV warheads, a maximum of 18 independent re-entry vehicles is allowed for each event.) This subroutine generates events separately for each squadron (or base) which belongs to the weapon group. This computation, however, is not performed at the start of the subroutine, but rather during processing. Since missile groups with a MIRV capability have a variable number of strikes per booster, the number of missiles in each event can be determined only dynamically.

The strikes are input into this subroutine in order of decreasing value. For weapon groups with a MIRV payload, the strikes are ordered by decreasing value of the total assignment to each booster. In order that each event to be output to PLNTPLAN will contain a mix of values for its strikes, the strikes are not assigned to events in simple serial order. Rather, each event is assigned one of the highest value strikes before any event receives a second strike. This process continues so that each event contains a mix of values. Subroutine POSIT is used to determine which strikes should be assigned to each event. Since there is no room to store the data for all events simultaneously, POSIT must predetermine which strikes should be placed in the event which is currently being processed. The inputs to POSIT are the current pointer to the strike array and the "skip" index. This latter variable informs POSIT of the number of strikes that it should skip before selecting a strike (or set of strikes for a MIRV weapon) to be added to the current event. It is this "skip" mechanism which distributes equal value missions to the different events; using POSIT, MISASGN selects a strike to start an event, skips a number of high value strikes, selects another for inclusion in the event, and so on. (Thus, the first event may be composed of strikes number 1, 11, 21, 31,....in the input list. The second event has strikes 2, 12, 22,...) The number of strikes to be skipped is computed as a function of the number of squadrons in the group. POSIT returns to MISASGN the starting index (in the strike array in common /3/) and the number of re-entry vehicles in the strike. Since POSIT considers a negative index number to designate the first strike assigned to booster, non-MIRV weapon groups have all negative index numbers to show one strike per booster. Program FOOTPRNT has previously negated the correct index numbers for MIRV weapon groups.

As MISASGN receives information on the target (or targets) to be added to the current event, it checks to see if there is room in the event for these targets. If the total number of targets would exceed 18, then the current event is output on the STRKFILE, the output array is cleared, and a new event is initiated with the targets that would not fit previously. MISASGN continues in this fashion until all strikes have been assigned to events.

Before assigning the strikes to each vehicle, the subroutine computes the number of vehicles in the group and the number of vehicle assignments (i.e., number of negative index numbers) for the group. If the number of vehicle assignments is less than the number of vehicles, the number of vehicles for which a plan will be processed is decreased until it matches the number of assignments. The vehicles are removed as equally as possible from each squadron (i.e., base). If the number of vehicle assignments exceeds the number of vehicles, the subroutine determines if the vehicles are carrying a MIRV payload. If so, then program FOOTPRNT has erred in generating the footprint assignments. MISASGN prints an error message to this effect and proceeds. The result will be the omission of some target sets from the final plan. If the group does not have a MIRV payload, MISASGN removes the least valuable assigned targets until the number of targets equals the number of vehicles. However, the subroutine will not omit any targets assigned by the fixed assignment capability of program ALOC, unless there are more fixed assignments for this group than there are vehicles. In that case, some fixed targets would be omitted.

The flow of operations in MISASGN is shown in figure 139. The flowchart consists of three parts. Part I illustrates the basic processing accomplished by MISASGN. Part II shows the operations performed to balance the number of weapon assignments with the available delivery vehicles. Part III shows the processing performed to construct the output missile events.

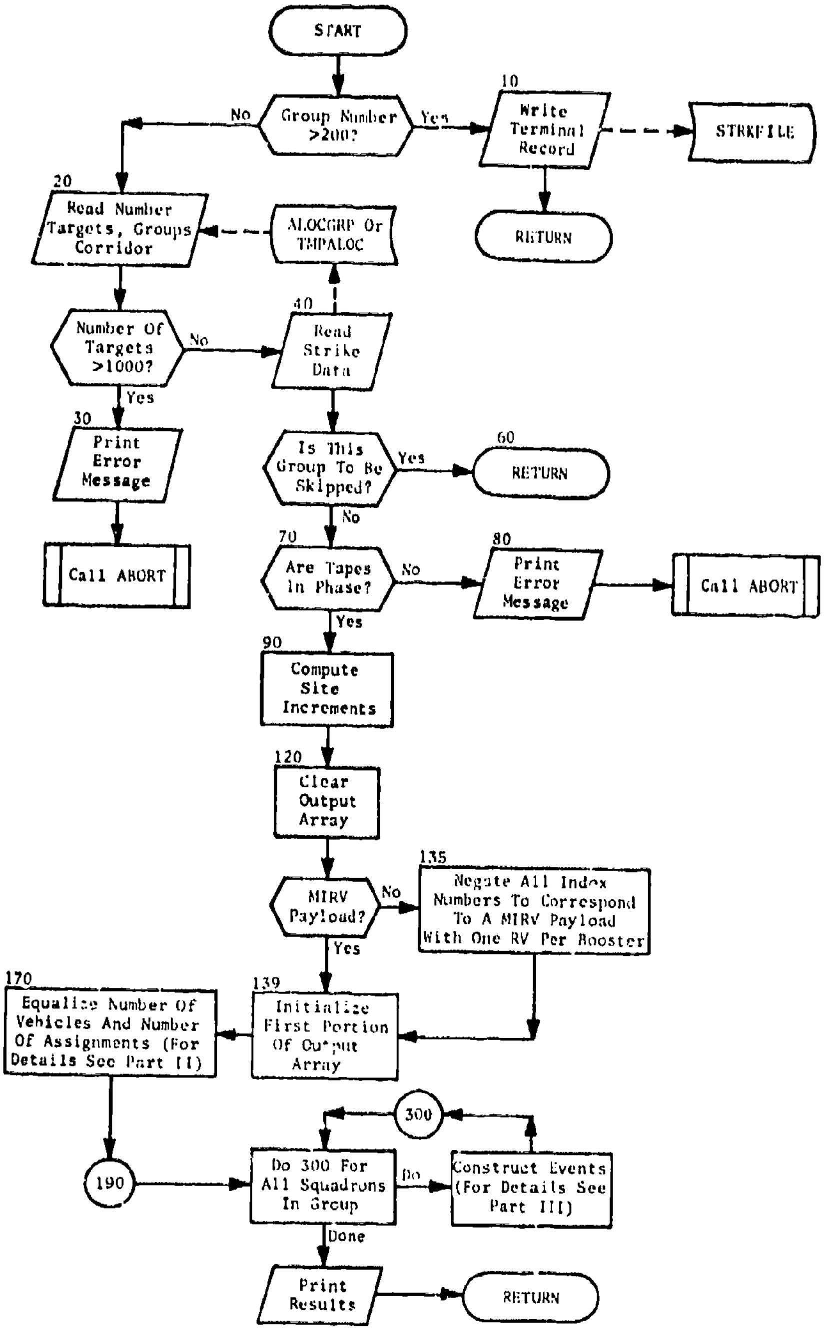


Fig. 139. Subroutine MISASGN
Part I: Basic Processing

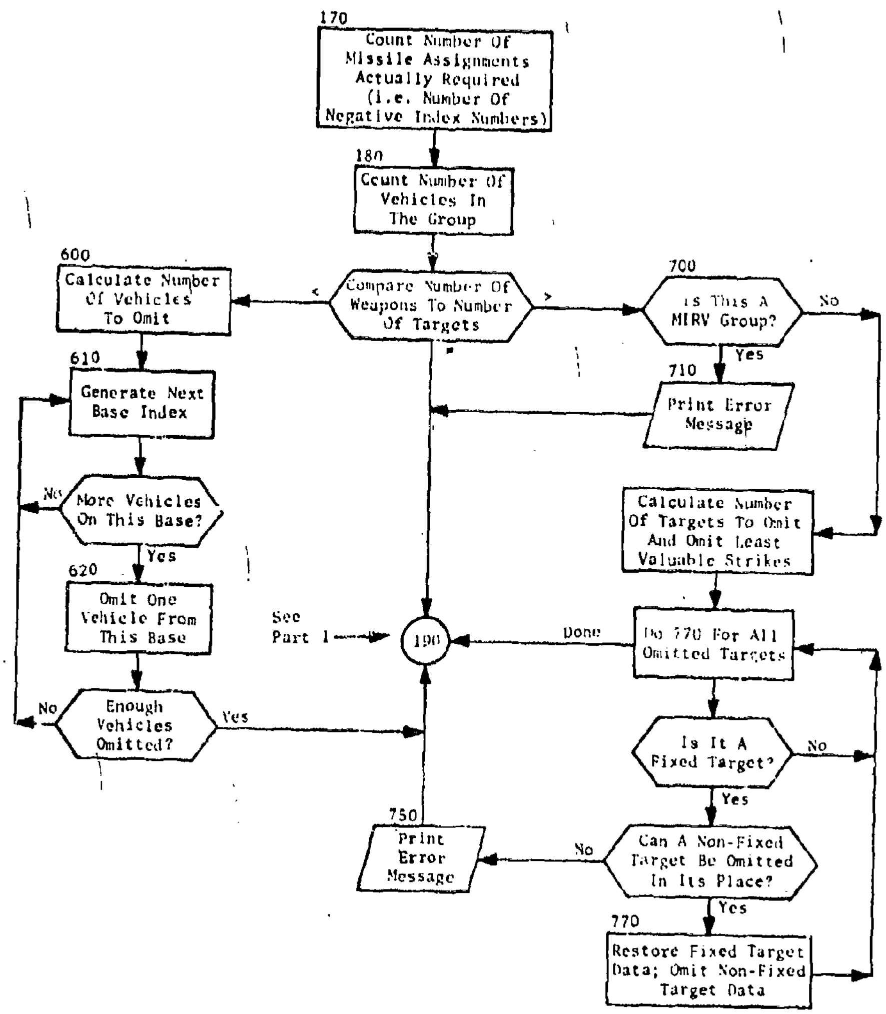


Fig. 139. (cont.)
Part II: Balancing Weapon Assignments and Delivery Vehicles

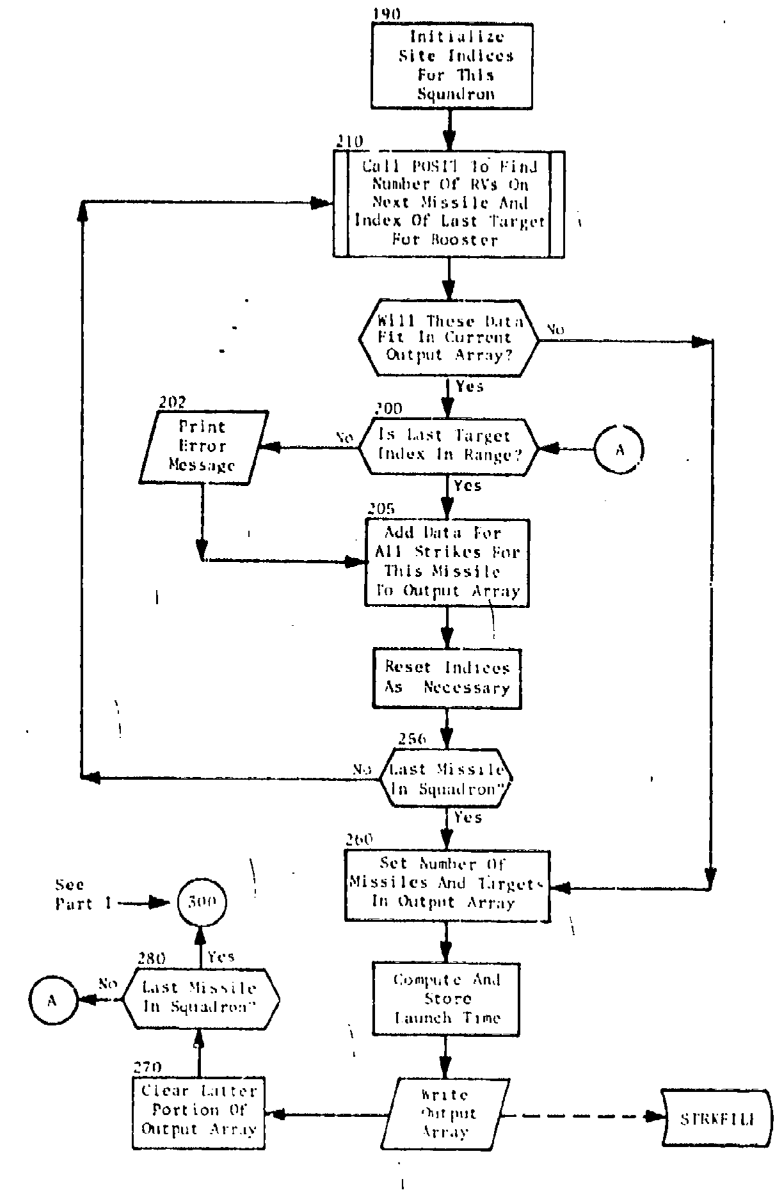


Fig. 139. (cont.)
Part III: Event Processing

SUBROUTINE NEXTFLT

PURPOSE:	To determine the launch base for the next flight and the number of vehicles in the flight.
ENTRY POINT:	NEXTFLT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	DEBUG, GRPDATA, KEYS, NEXTFLT, PCALL, PRINTOPT, RAIDSHR
SUBROUTINES CALLED:	IGET, PRINTIT
CALLED BY:	FLTROUTE
CALLING PARAMETERS:	JCORR, ISIDE

Method

NEXTFLT selects the number of vehicles NTAILS to be processed in the next flight and designates the base index of the vehicles. It assigns all vehicles from one base before processing the next base. If there is to be one flight from the base, all vehicles remaining on the base are assigned to that flight. If there are four or fewer bases, two flights will be sent. In this case if nearly all the original vehicles remain on the base, half of them are assigned to the flight; if only half remain, all of them are assigned to the flight. Since the order of the bases is changed from one corridor to the next, NEXTFLT must be prepared to encounter new bases that are already completely or partly depleted of aircraft.

NEXTFLT uses the NRVEH (number of remaining vehicles) array to determine the launch base of the next flight. It receives the variable SPLIT from FLTROUTE, which equals 1 if the base is to split into two flights, and equals 0 if the entire base is to fly down the same side of the corridor. Thus, by letting CRITN = 1.7 - SPLIT, if NRVEH is less than CRITN * VPRBASE, all vehicles remaining on the base are assigned to the next flight; otherwise, half the vehicles remaining are assigned.

If a penetration corridor is being used, that is, JCORR is not equal to 1 or 2, the bases are processed in turn, going from the beginning of the list to the end. In the case of tactical bombers (JCORR = 1) or naval bombers (JCORR = 2), the bases are processed in the same way as the targets, that is from both ends of the list working toward the middle, depending on which side of the corridor, ISIDE, the flight is taking.

Subroutine NEXTFLT is illustrated in figure 140.

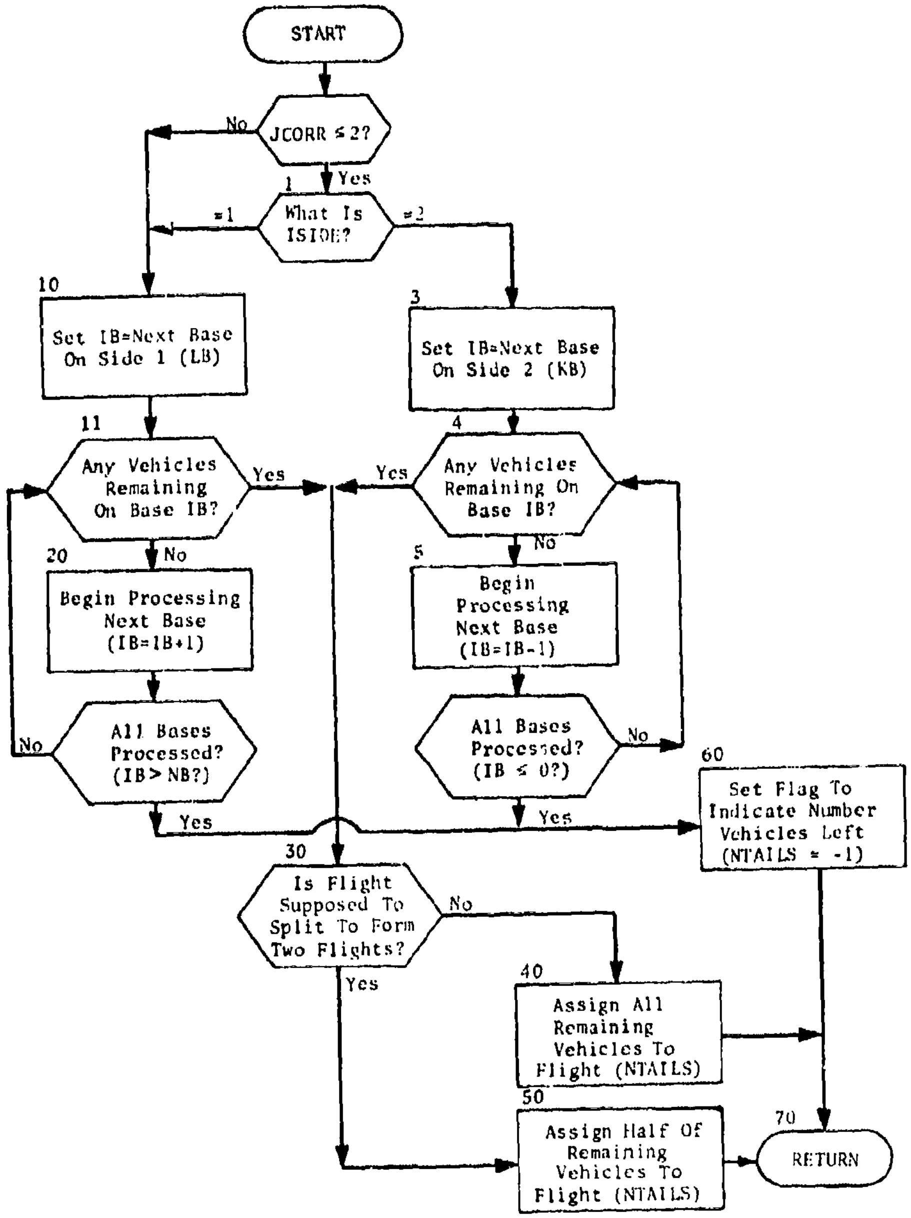


Fig. 140. Subroutine NEXTFLT

462-546 O - 72 - 20

SUBROUTINE NOCORR

PURPOSE: Sets variables in commons /CORRIDOR/ and /RAIDSHR/ in order to process tactical bombers or bombers assigned to naval targets.

ENTRY POINTS: NOCORR

FORMAL PARAMETERS: None

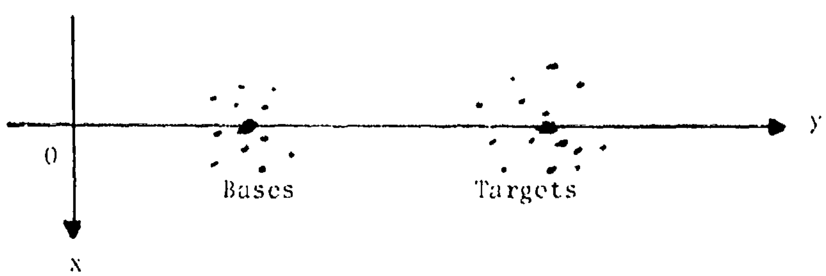
COMMON BLOCKS: CORRIDOR, FIXRANGE, GRPDATA, 3, POLITE, RAIDSHR

SUBROUTINES CALLED: CENTROID, DISTF, INTERP

CALLED BY: GENRAID

Method

Subroutine NOCORR defines an axis to be used in the coordinate system calculated by subroutine CORRPARM. This axis is normally the penetration corridor direction arrow. In the case of tactical or naval bombers which do not use penetration corridors but fly directly from base to target, the y-axis of the coordinate system is defined to be the line running through the centroid of the bases in the weapon group and the centroid of the targets assigned to that group. In order to assign values of RHO and PHI to the bases as well as the targets, the origin of the system is defined to be a point projected on the other side of the base centroid at a distance from the base centroid which is equal to the distance between the target and base centroid, as shown below.



NOCORR sets up this axis very simply by calling subroutine CENTROID to compute the two centroids, and by calling subroutine INTERP to find the origin point. The coordinates of these points are filled into the common block /CORRIDOR/ which normally contains the corresponding penetration corridor coordinates. KORPWR is set to 1, which causes lines of constant ϕ to be straight lines radiating from the origin and lines of constant ρ to be concentric circles in the relevant range.

NOCORR also computes the distance of each base from the target centroid, and the distance between the two centroids.

Subroutine NOCORR is illustrated in figure 141.

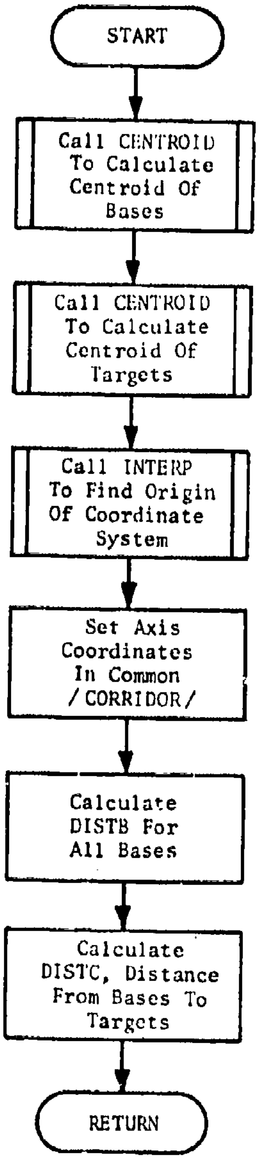


Fig. 141. Subroutine NOCORR

SUBROUTINE OPTRAID

PURPOSE:	To control the setting up, optimizing, storing, and writing on tape of the sortie plans, for each sortie in the raid.
ENTRY POINTS:	OPTRAID
FORMAL PARAMETERS:	None
COMMON BLOCKS:	DEBUG, FLTFASS, INDEX, PCALL, PRINTOPT, RAIDSHR
SUBROUTINES CALLED:	DUMPSRT, GETSORT, INITOPT, OUTSRT, PRINTIT, PRNTF, REFABORT, SETFLAG, SORTOPT, TIMEME
CALLED BY:	GENRAID

Method

OPTRAID controls the cycling of the optimization from one sortie to the next within a raid. It is called by GENRAID after FLTROUTE and TGTASGN have completed an initial assignment of strikes to sorties for all sorties in the raid (or subraid). For each sortie, OPTRAID makes calls on the appropriate data handling routines to set up the sortie for optimization, and it then calls SORTOPT to accomplish the optimization.

Before beginning to process the sorties in the raid, OPTRAID makes a call on INITOPT. The purpose of this call is to clear out the SORTYTGT arrays and initialize them for the new raid.

OPTRAID makes two optimization passes over the sorties, the second in reverse order. One reason for the double pass over the sorties is to provide a chance for valuable targets omitted by a sortie to be picked up by sorties on either side. On each pass, before calling SORTOPT, OPTRAID calls GETSORT. GETSORT reads the list of targets for the sortie as prepared by TGTASGN and inserts the appropriate targets into the SORTYTGT arrays. GETSORT also sets up the current definition of the sortie in the CURSORTY array.

SORTOPT is then called to optimize the sortie, and DUMPSRT is called to record the resulting sortie. Targets that remain in the sortie are cleared out of the SORTYTGT array and are recorded for the skeletal representation of the sortie in the CURRAID array. (The indices for

ASM targets are recorded here with a minus sign.) Similarly, if the resulting optimized sortie does not include recovery, this is also noted in the skeletal representation (by making the number assigned NASGN negative in CURRAID). Targets that are omitted from the sortie by SORTOPT are not cleared out by DUMPSRT, so that they remain in the IOMIT list for future consideration.

On the second pass, subroutine OUTSRT is called after SORTOPT. OUTSRT records the final form of the sortie, including specific distances at low altitude, on the output STRKFILE to be used by PLNTPLAN.

A call is then made on REFABORT. If the bomber does not refuel, REFABORT simply returns. However, if the bomber does refuel, REFABORT prepares an alternative sortie plan to be used in case of refueling abort. Basically, REFABORT sets up a revised sortie to be optimized, calls SORTOPT and OUTSRT, and then restores all arrays to their previous content. To set up the alternative sortie, REFABORT reduces the range used by SORTOPT for the sortie. It also adds into the IOMIT list certain alternative targets with relatively low values of RHO and values of PHI close to those of targets in the original plan. Since these alternative targets are probably already scheduled to be hit by the adjacent sorties, their value is reduced by 50% so that they will be less attractive than the targets now assigned to the sortie. Consequently, these alternative targets are usually used by SORTOPT in developing the alternate plan only if it is impossible to reach the ones originally assigned. When the alternate plan is complete, OUTSRT is called to record it on the output file for PLNTPLAN. REFABORT then sets all variables back as it found them and exits.

Subroutine OPTRAID is illustrated in figure 142.

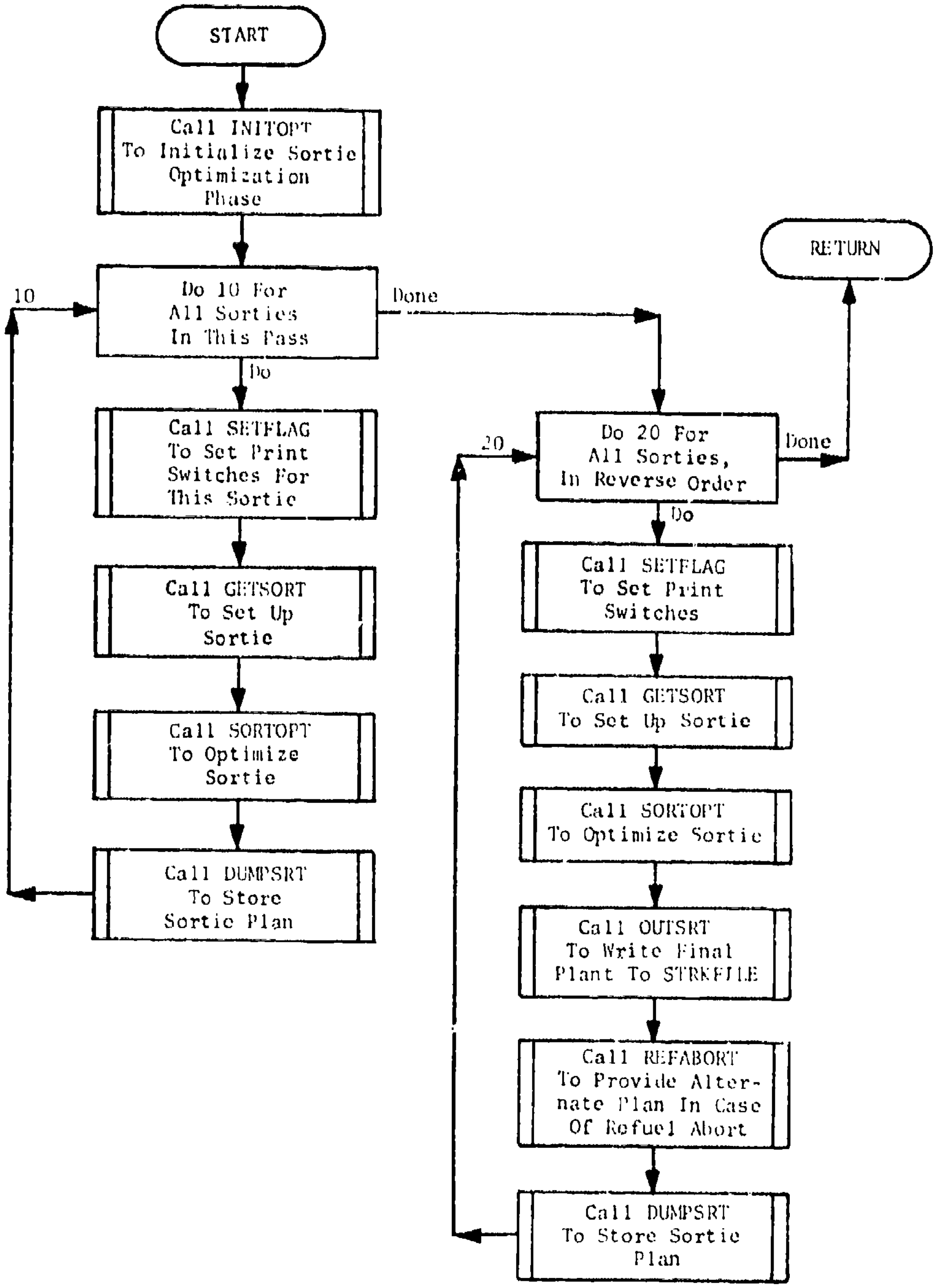


Fig. 142. Subroutine OPTRAID

SUBROUTINE OUTPOTGT

PURPOSE: To remove a potential target from the omit list.

ENTRY POINTS: OUTPOTGT

FORMAL PARAMETERS: None

COMMON BLOCKS: CHGPLAN, CURRAID, CURSORTY, DEBUG, PRINTOPT, SORTYTGT

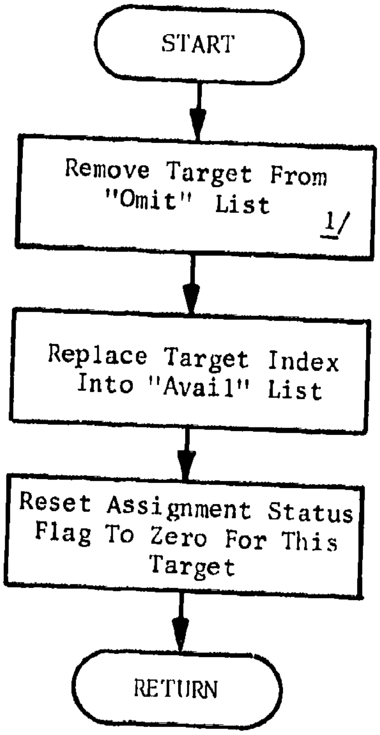
SUBROUTINES CALLED: None

CALLED BY: GETSORT

Method

OUTPOTGT receives from GETSORT the SORTYTGT index JDO in common /CHGPLAN/ for the target to be removed from the omit list. It moves the index from the omit list to the AVAIL list, resets the values of LKHITMT for the targets affected by the move, and flags the target as available.

Subroutine OUTPOTGT is illustrated in figure 143.



Note:
1. 10M is set to the position of the target JX in the omit list (=-LKHITMT(JX)), where JX has been set equal to JDO.

Fig. 143. Subroutine OUTPOTGT

SUBROUTINE OUTSRT

PURPOSE:	To write the final bomber sortie plan onto the output STRKFILE for input to PLNTPLAN.
ENTRY POINTS:	OUTSRT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	3, CORRIDOR, CURRAID, CURSORTY, DEBUG, FILES, FLTPASS, GRPDATA, GRPTYPE, IDUMP, IFTNO, IFTPRNT, INDEX, INITOPT, INPUTFL, ITP, OUTSRT, PCALL, PLANTYPE, PRINTOPT, PRNTF, RAIDSHR, SORTYTGT
SUBROUTINES CALLED:	ABORT, FINFLT, PRINTIT, WRARRAY
CALLED BY:	GETGROUP, OPTRAID, REFABORT

Method

All of the data to be included in the output record are assembled in common /OUTSRT/ to be written out as a block onto the STRKFILE. A call is made on FINFLT, which is a special entry point in FLTPLAN, to produce a final sortie plan. This final plan differs from previous plans in that, if there are defended targets past the point at which low altitude ran out, FINFLT considers flying low up to each of these defended targets (and therefore flying high earlier in the route), checking at each leg to see if the value of the sortie is increased. When the final plan is obtained, it is included in common /OUTSRT/ as a series of happenings with associated latitudes, longitudes, and object numbers. WRARRAY is then called to write the record on STRKFILE.

Subroutine OUTSRT is illustrated in figure 144; sheet 2 of the figure consists of notes to facilitate interpretation of the flowchart.

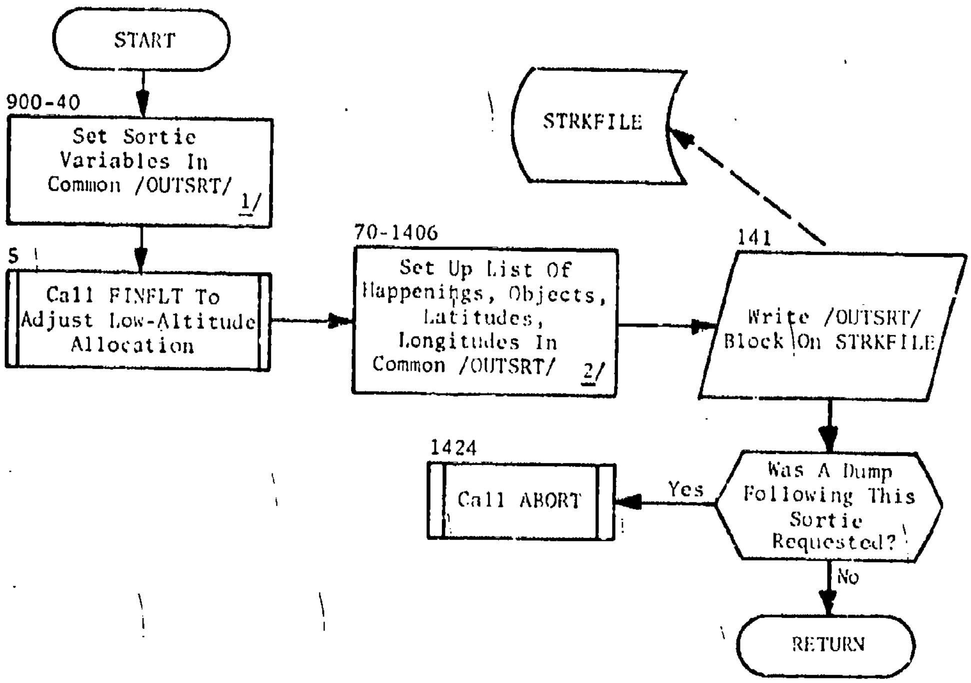


Fig. 144. Subroutine OUTSRT
(Sheet 1 of 2)

Notes

1. IOUTSRT runs from 1 to NVEH in the output records for the sorties from a given weapon group through a given corridor. This index must be calculated from ISRT, IFPASS, and NVEHPASS, since OUTSRT is called on the second optimizing pass for each "FLTROUTE pass," and ISRT is running in descending order.

   MYGROUP is set equal to IGROUP. IGROUP is used as a flag (set equal to 201) when all groups have been processed and thus causes a sentinel record with MYGROUP = 201 to be written out.

   JDPEN is the depenetration corridor associated with the last target in the fly list. If the last target is hit with an ASM shot from the corridor origin, JDPEN is the depenetration corridor associated with that target (i.e., the last target in the hit list).

   DELAY = 0 if the sortie is on alert, and is equal to the difference between alert delay and nonalert delay, if the sortie is nonalert.

2. The flight from base to corridor origin is called a dogleg. The latitude and longitude are for the corridor origin, and the object is the corridor index number. An ASM strike is called AIM ASM. A bomb strike is called DROPBOMB. In either case, latitude, longitude, and object index number are those of the target.

   Depenetration is called DEPEN, and the latitude, longitude, and depenetration corridor indices are given.

   If the bomber is an "air-breathing missile," instead of DEPEN, we write DIVEMISL. If recovery is omitted, we write LAND. In either case, the latitude, longitude, and object number are irrelevant and set to 0.

   DLAT and DLONG are used to represent deviations of the desired ground zero from the target latitude and longitude given for a complex target. In all other cases, these are zero.

   If for any reason a bomber does not have range to reach even one target, RNGSURP will be negative in the final plan. In this case, DSTLOW1 is set equal to RNGSURP as an error flag to the user and to program PLNTPLAN.

Fig. 144. (cont.)
(Sheet 2 of 2)

SUBROUTINE POSIT

PURPOSE:	To select the strikes to be added to the current event in MISASGN.
ENTRY POINTS:	POSIT
FORMAL PARAMETERS:	ITGT, ISKIP, JTGT, NRVS
COMMON BLOCKS:	3
SUBROUTINES CALLED:	None
CALLED BY:	MISASGN

Method

MISASGN calls this subroutine to select the set of strikes to add to the current event. In order that each event will have a spread of values in its strikes, this subroutine must not only find the targets that belong to a single missile (for MIRV weapons) but also return these data in an order that distributes the value evenly. The strikes are placed in the data array in common /3/ in order of decreasing value.

The input from MISASGN is:

ITGT - An index to the INDEXNO array which specifies the target at which the search is to begin

ISKIP - An integer which specifies the number of missile plans the subroutine is to skip before returning to MISASGN.

The output from POSIT IS:

JTGT - An index to the INDEXNO array which specifies the first target in the missile assignment

NRVS - The total number of targets assigned to the missile.

This subroutine recognizes the first target assigned to a missile by a negative index number in the INDEXNO array. For weapon systems without MIRV capability, all index numbers are negative and NRVS is always set equal to one.

Subroutine POSIT is illustrated in figure 145.

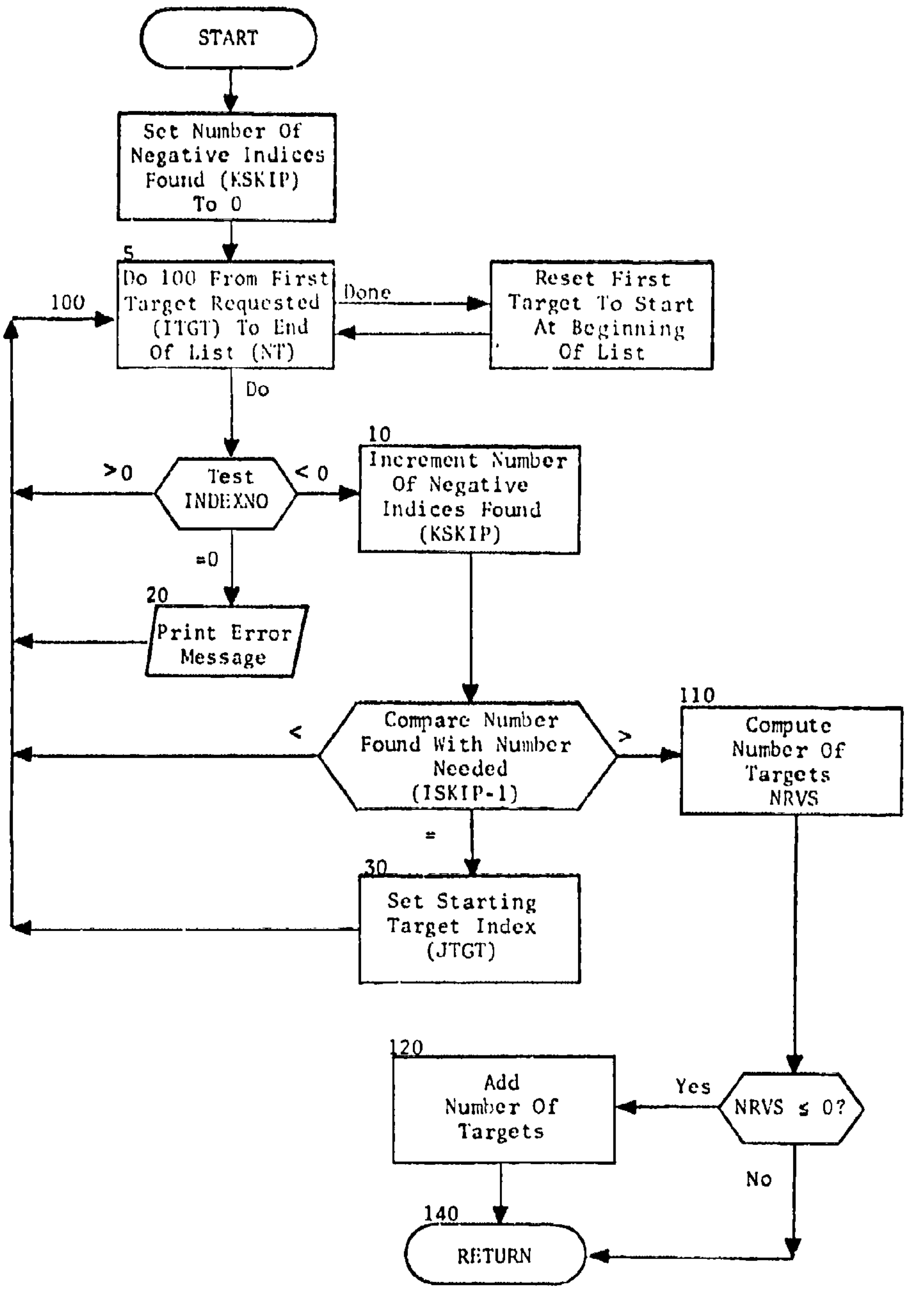


Fig. 145. Subroutine POSIT

SUBROUTINE PRERAID

PURPOSE:	To control the processing for an entire bomber group and determine the distribution of sorties by penetration corridor.
ENTRY POINTS:	PRERAID
FORMAL PARAMETERS:	None
COMMON BLOCKS:	ARAYSIZE, CORRCHAR, DEBUG, DPENREF, FILES, GRPDATA, GRPTYPE, IFTPRNT, INPUTFL, ITP, KEYS, MASTER, PAYLOAD, PCALL, PRINTOPT, RAIDSHR, STRKSUM
SUBROUTINES CALLED:	GENRAID, IGET, KEYMAKE, PRINTIT, PRNTF, RDARRAY
CALLED BY:	GETGROUP

Method

PRERAID controls the entire sortie generation processing for each bomber group. It reads the header array STRKSUM from the ALOCGRP or TEMPALOC file to obtain summary information on how the strikes assigned to the group are divided among the various corridors. Then for each corridor, it designates the weapon resources to be used and calls GENRAID to divide up the job in that corridor.

After reading the input file, PRERAID calculates the following variables in common /RAIDSHR/ to be used in later processing:

VPRBASE	- The average number of vehicles per base
NRVEH(IB)	- The number of vehicles in group on IBth base which are still unassigned
NASMPV(IB)	- The number of ASMs per vehicle on the IBth base
NWPV(IB)	- The number of warheads per vehicle on the IBth base
NB	- The number of bases in the group
NWPC	- The total number of warheads to enter by the corridor.

It also accumulates the number of warheads in this group (NWPG).

The preferred number of warheads to assign to the next corridor (neglecting the problem of integral number of vehicles) is calculated using the total number of strikes remaining and the total number of warheads remaining.

GENRAID is then called to develop a raid in the corridor using vehicles as required to carry at least the warheads specified.

It should be noted that the number of targets assigned to a given weapon group by the allocator always slightly exceeds the number of weapons in the group, to allow for some flexibility in setting up the sorties. The distribution of weapons through the various penetration corridors should be in the same proportions as the whole set of targets assigned. It is on this basis that the number of weapons for each corridor is calculated.

Subroutine PRERAID is illustrated in figure 146.

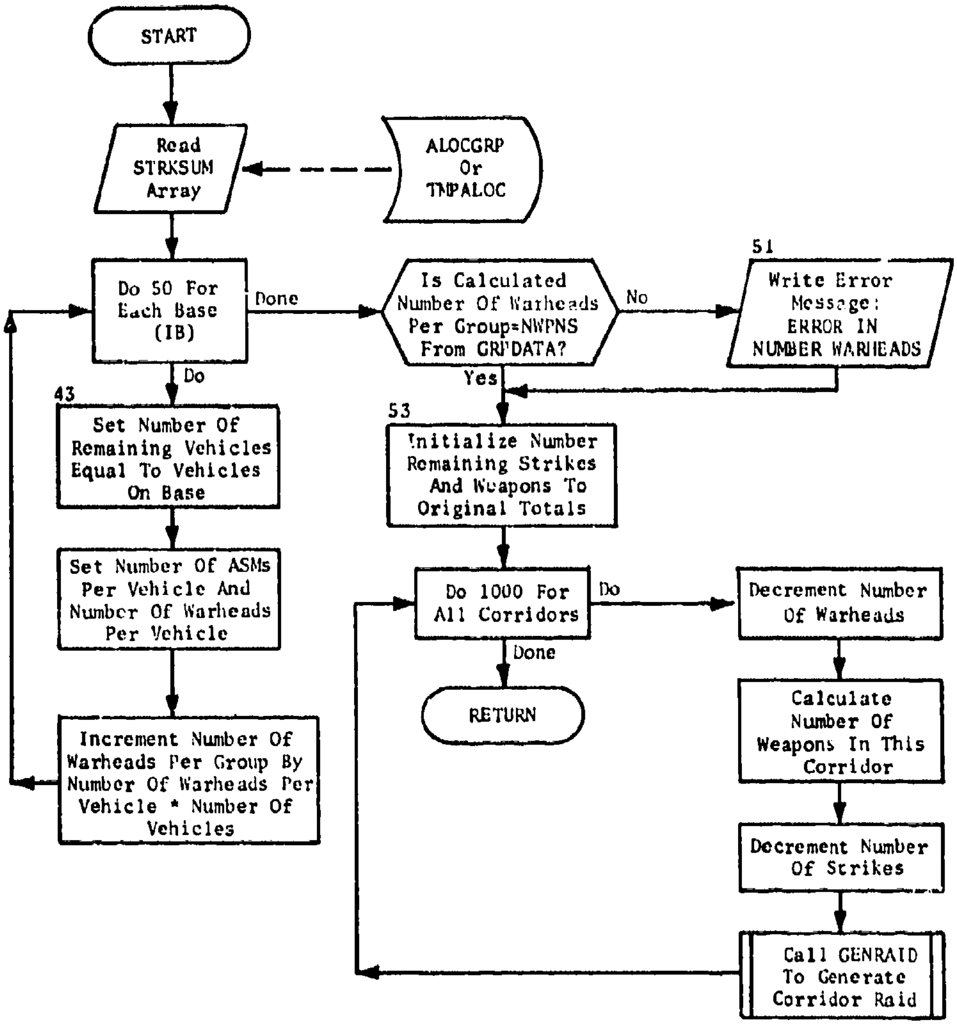


Fig. 146. Subroutine PRERAID

462-546 O - 72 - 21

SUBROUTINE PRINTIT

PURPOSE: To write formatted prints of the contents of various common blocks.

ENTRY POINTS: PRINTIT

FORMAL PARAMETERS: ICOMMON

COMMON BLOCKS: 2, 3, ARAYSIZE, CHGPLAN, CORRCHAR, CORRIDOR, CURRAID, CURSORTY, DEBUG, DPENREF, EVAL, FLAG, FLTPASS, GRPDATA, GRPTYPE, INDEX, INITOPT, MASTER, MISPRNT, NEXTFLT, OUTSRT, PAYLOAD, PCALL, PRINTOPT, RAIDSHR, SORTYTGT, STRKSUM, TGTASGN

SUBROUTINES CALLED: TIMEME

CALLED BY: CHGPLAN, CORRPARM, DUMPSRT, EVALB, EVALOA, EVALOB, FLTPLAN, FLTROUTE, GENRAID, GETGROUP, GETSORT, INITOPT, INPOTGT, MISASGN, NEXTFLT, OPTRAID, OUTSRT, PRERAID, REFABORT, SORTOPT, TGTASGN

Method

PRINTIT is called with one of the variables in common /PRINTOPT/, which are mnemonically indicative of the common block being requested. These variables have been set to positive integers so that they may be used in a computed GO TO to the proper portion of the program.

PRINTIT first checks the ICALLth cell of IFLAG to see if the print is active. (See description of subroutine SETFLAG.) If not, the program returns. If it is active, a one-line print is then written giving the value of ICALL and the calling subroutine. The computed GO TO then sends the program to the desired print.

Subroutine PRINTIT is illustrated in figure 147.

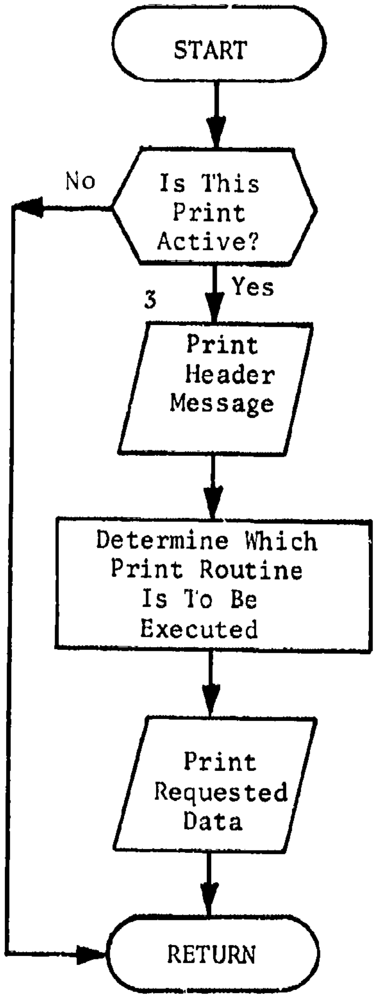


Fig. 147. Subroutine PRINTIT

FUNCTION PRNTF

PURPOSE:	To write formatted prints of local or indexed variables, called often within a do loop in a subroutine where the index is one of the calling parameters.
ENTRY POINTS:	PRNTF
FORMAL PARAMETERS:	IP, JP, IDENT
COMMON BLOCKS:	3, ARAYSIZE, CORRIDOR, CURRAID, CURSORTY, DATA, DEBUG, EVAL, FLAG, FLTPASS, GRPDATA, GRPTYPE, INDEX, INITOPT, PCALL, PRNTF, RAIDSHR, RUNCHECK, SKIP, SORTYTGT
SUBROUTINES CALLED:	TIMEME
CALLED BY:	EVALB, EVALOA, EVALOB, FLTPLAN, FLTROUTE, GENRAID, GETGROUP, GETSORT, OPTRAID, POSTALOC, PRERAID, TGTASGN

Method

This second print routine was written in order to print indexed variables being computed within a do loop, controlling on one or two separate indices IP and JP and identifying the stage of processing by a Hollerith identifier in IDENT. It also will print certain temporary values which were originally local variables but have now been placed in common /PRNTF/. PRNTF first tests the ICALLth cell of IFLAG to see if the print is active. If not, it returns. Since the print may be a one-line print called within a do loop, a test is made on IDENT so that the header will be printed only on the first call in a loop. Before the header for any print, one line is printed giving the value of ICALL and the calling subroutine.

The following values of IP will produce different types of prints: any positive integer, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12. For the first type, where IP is positive, IP is the do loop index and JP is the SORTYTGT index of the target for which the variables are being printed. For each of the other types, where IP is negative, the sign is reversed and IP is used in a computed GO TO the part of the program writing the desired type of print. JP may or may not be a significant index for these prints. (See the User's Manual for examples of all prints.)

Function PRNTF is illustrated in figure 148.

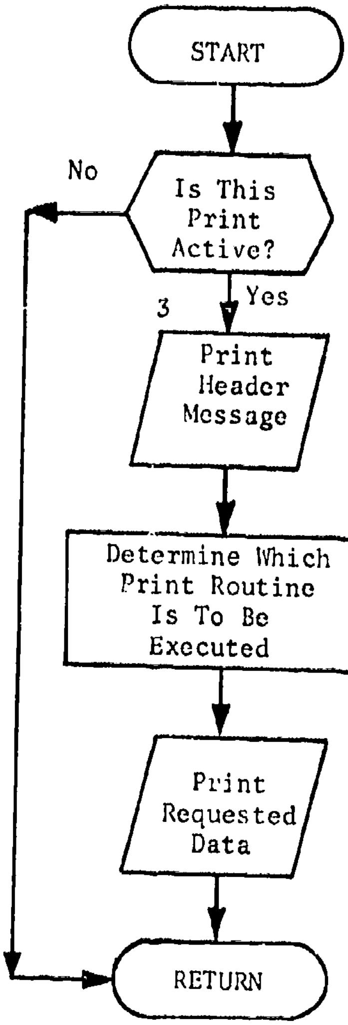


Fig. 148. Function PRNTF

SUBROUTINE REFABORT

PURPOSE: To prepare an alternate bomber plan requiring less range, to be used in the event of a refuel abort.

ENTRY POINTS: REFABORT

FORMAL PARAMETERS: None

COMMON BLOCKS: 2, 3, CURRAID, CURSORTY, GRPDATA, GRPTYPE, DEBUG, IDUMP, INITOPT, INDEX, PCALL, PRINTOPT, SORTYTGT, VAL

SUBROUTINES CALLED: INPOTGT, OUTSRT, PRINTIT, SORTOPT

CALLED BY: OPTRAID

Method

REFABORT stores the regular plan defined in common /CURSORTY/ in a temporary location for later retrieval. It then searches for closer targets, which may be already assigned to other sorties, and sets up an alternate plan which utilizes all weapons within the limitation of unrefueled range. When the new set of potential targets has been obtained, SORTOPT is called to optimize the sortie plan, and OUTSRT is called to write the output record for the alternate plan on the STRKFILE.

Subroutine REFABORT is illustrated in figure 149; sheet 3 of the figure consists of notes to facilitate interpretation of the flowchart.

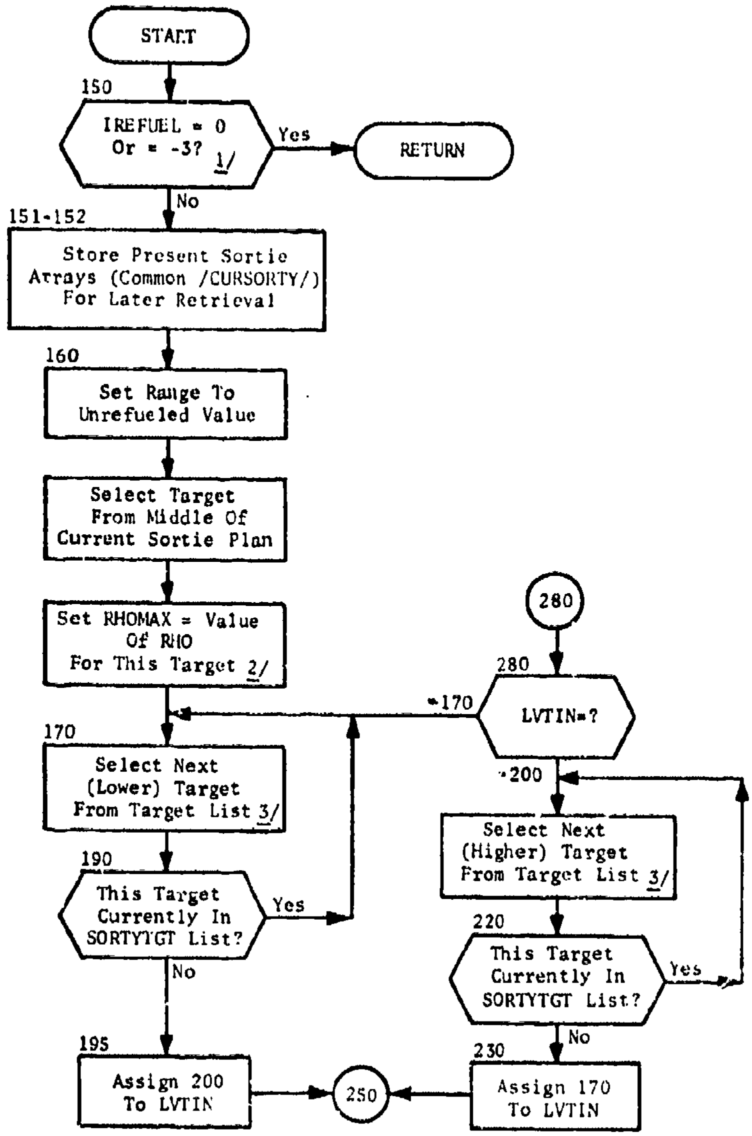


Fig. 149. Subroutine REFABORT
(Sheet 1 of 3)

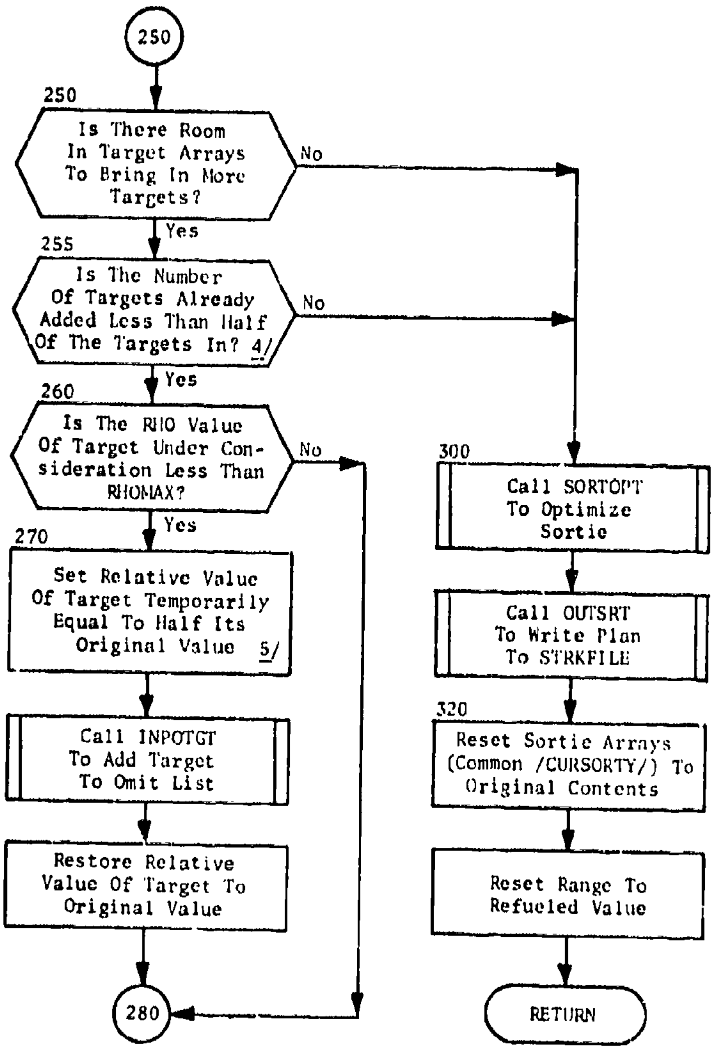


Fig. 149. (cont.)
(Sheet 2 of 3)

Notes:

1. IREFUEL may assume the following values (when input to POSTALOC):

-5	For program-assigned area refueling (two refuels)
-4	For program-assigned area refueling (one refuel)
-3	For air-breathing missiles
-1 or -2	For "buddy refueling"
0	For no refueling
1,...,NREFUEL	For area refueling, where the value is the index of the refueling area.

2. The new set of potential targets should be near the original set but closer to the corridor origin. Therefore, the targets with the closest values of PHI are examined, and those whose values of RHO are less than or equal to the middle target on the original list are selected.

3. Markers ITHI and ITLO are kept to designate the range of targets that has been searched. If ITHI exceeds NT (the total number of targets), or if ITLO is less than one, the corresponding flag ITOOHI or ITOOLO is set. As long as these limits are not exceeded, the search alternates from targets preceding to targets succeeding the "middle" target, in the PHI-sorted RAIDSTRK array.

4. The number of new targets brought in should not exceed NHIT/2 (where NHIT is the number of targets, plus origin, recovery, and ditch) for the original sortie plan.

5. RVAL is reduced for the new targets since they presumably are to be hit by some other sortie. Also, since INPOTGT will tamper with MYASGN, its original value for the new target is stored before INPOTGT is called. Afterwards, both RVAL and MYASGN are reset to their original values.

Fig. 149. (cont.)
(Sheet 3 of 3)

SUBROUTINE SETFLAG

PURPOSE:	To read print request control cards from standard input data, and set print switches as requested.
ENTRY POINTS:	SETFLAG
FORMAL PARAMETERS:	None
COMMON BLOCKS:	DATA, FLAG, GRPDATA, INDEX, PCALL
SUBROUTINES CALLED:	None
CALLED BY:	POSTALOC, GENRAID, OPTRAID, GETGROUP

Method

The first time it is called, SETFLAG reads all the data cards into common /DATA/. A card with 99999 in the first field, or the 60th card, will terminate the read. The data cards consist of six format fields (I10) as follows:

Field	Contents
1	Value of ICALL at desired print request (print request number)
2	First sortie for which print is desired
3	Last sortie for which print is desired
4	OPTRAID pass (1 or 2) for which print is desired
5	Penetration corridor in which print is desired
6	Weapon group in which print is desired

A zero in any field implies no restriction at that level; i.e., a card with 36, 0, 0, 1, 3, 0 will turn the print at ICALL = 36 on for all sorties on the first pass of the third corridor in all groups.

SETFLAG is called by GENRAID whenever a new corridor is processed, and is called by OPTRAID whenever a new sortie is processed. Thus on each call, it tests the current sortie, pass, corridor, and group against each print request in common /DATA/ and, if the conditions are right, it activates the print by setting to one the ICALLth cell of the IFLAG array in common /FLAG/.

Subroutine SETFLAG is illustrated in figure 150.

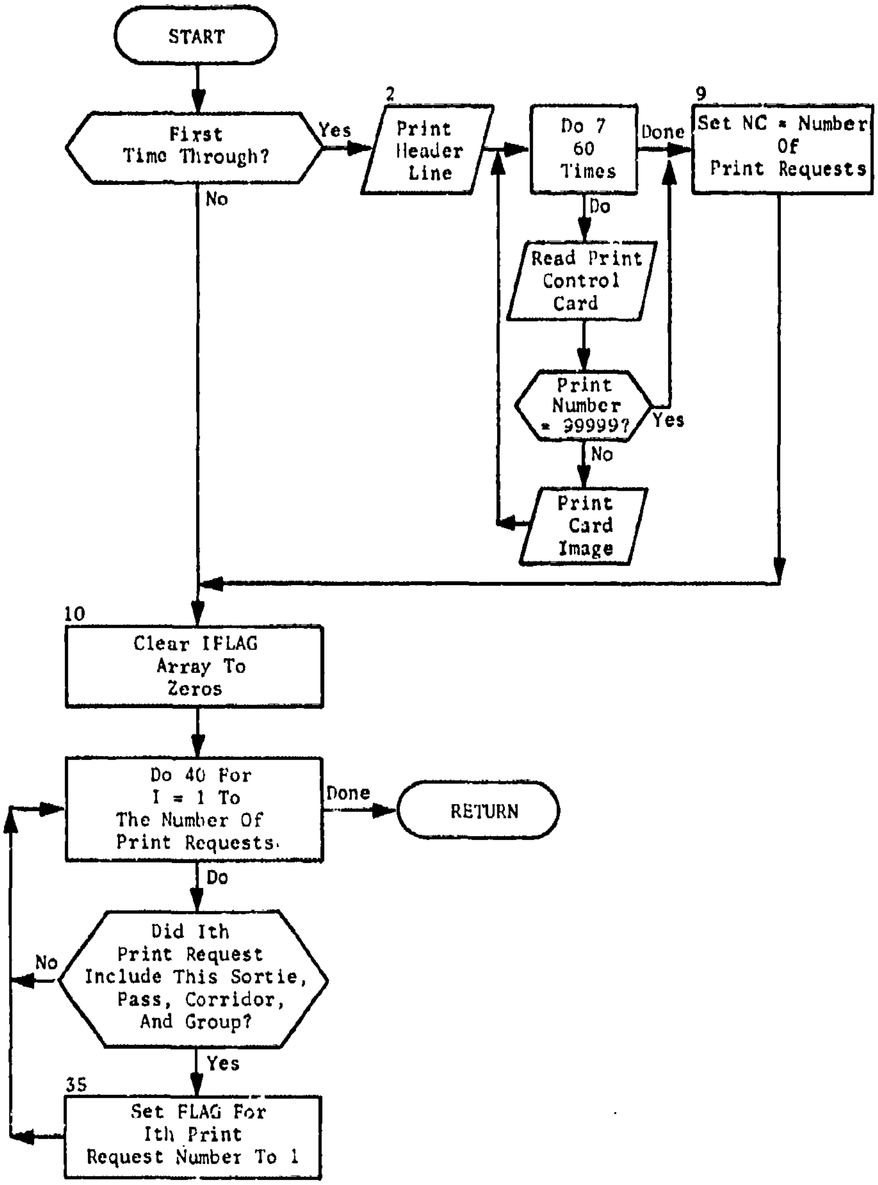


Fig. 150. Subroutine SETFLAG

SUBROUTINE SORTOPT

PURPOSE:	To control the optimization of each sortie.
ENTRY POINTS:	SORTOPT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	CHGPLAN, CONTROL, CURSORTY, DEBUG, EVAL, FIXALL, GRPTYPE, INDEX, PCALL, PRINTOPT, SORTYTGT, VAL
SUBROUTINES CALLED:	ABORT, CHGPLAN, EVALB, EVALOA, EVALOB, FLTPLAN, PRINTIT
CALLED BY:	OPTRAID, REFABORT

Method

SORTOPT controls the actual optimization of each individual sortie. When SORTOPT is called, all the relevant targets for the sortie to be optimized are in the SORTYTGT arrays, and an initial definition of the sortie which defines the sequence of targets to be attacked is contained in common /CURSORTY/. SORTOPT modifies this sortie definition in /CURSORTY/ to produce a feasible sortie of the highest possible value.

SORTOPT first calls EVALB to evaluate each target in the current sortie definition in common /CURSORTY/ as set up by GETSORT. If there are ASMs on the bomber, they will be placed on the targets chosen by EVALB as the best potential ASM targets. CHGPLAN is called to effect these changes from bombs to ASMs. If there are too many targets in the plan, the least valuable ones are omitted. (Each call on EVALB returns the index of the least valuable target as JDELB in common /EVAL/.) SORTOPT then calls EVALB to see whether the sortie could be improved by deleting any target from the plan. EVALOA and EVALOB are called to consider trading any targets in the hit list for one of the omitted ones to be hit by an ASM or a bomb, respectively, in order to improve the sortie value. Any omission or switch in the sortie definition (the "hit" list in common /CURSORTY/) is effected by calling CHGPLAN. FLTPLAN is called after each change to recompute the distances involved and to recalculate the value of the sortie to be sure that the change was an improvement.

Figure 151, in four parts, illustrates the operation of SORTOPT. Part I modifies the specified number of strikes so that they will correspond to the available warheads. First, until all available ASMs are used, the strikes (initially assigned for bombs) are converted to ASMs--using those for which the change promises the largest improvement in VALSORTY. Then if there remains an excess of bomber strikes, the least valuable strikes are deleted until the total number of strikes is just equal to the number of warheads.

Part II checks the profitability of recovery of all strikes assigned. Any strikes (or recovery) with a negative estimated payoff are omitted, beginning with the largest. After each strike is omitted, FLTPLAN is called to verify that the omission did indeed improve the payoff. If not, the omitted point is replaced in the mission, and control passes to Part III.

Part III of the flowchart deals with the possible substitution of omitted targets in place of targets now assigned to the ASMs. This portion carries out the substitution in the event that some of the omitted strikes appear to make better targets. Since addition or deletion of an ASM does not involve changing the flight path, the estimated changes in VALSORTY are assumed to be exact, and no check with FLTPLAN is made after the change.

Part IV deals with two considerations. First it checks the flight points to see if the length of the sortie could be reduced by reversing the order of any two flight points. If so, one of the two points is omitted with the expectation that it will be recovered and placed in the proper sequence when omitted targets are examined as potential targets for bombs. In the next phase, if some bombs are left without targets, EVALOB is called to find an omitted target that is estimated to increase VALSORTY if used as a target for a bomb in place of the least profitable target now assigned. The indicated changes, if any, are implemented, and FLTPLAN is called to check VALSORTY. If VALSORTY did not improve, the change is reversed.

At this point, the routine returns unless there still are insufficient ASMs assigned. If so, control is passed back to Part III for a final effort to locate a suitable ASM target in the Omit List.

Subroutine SORTOPT is illustrated in figure 151; part V of the figure consists of notes (referenced by number within the flow diagram) to explain the processes in more detail.

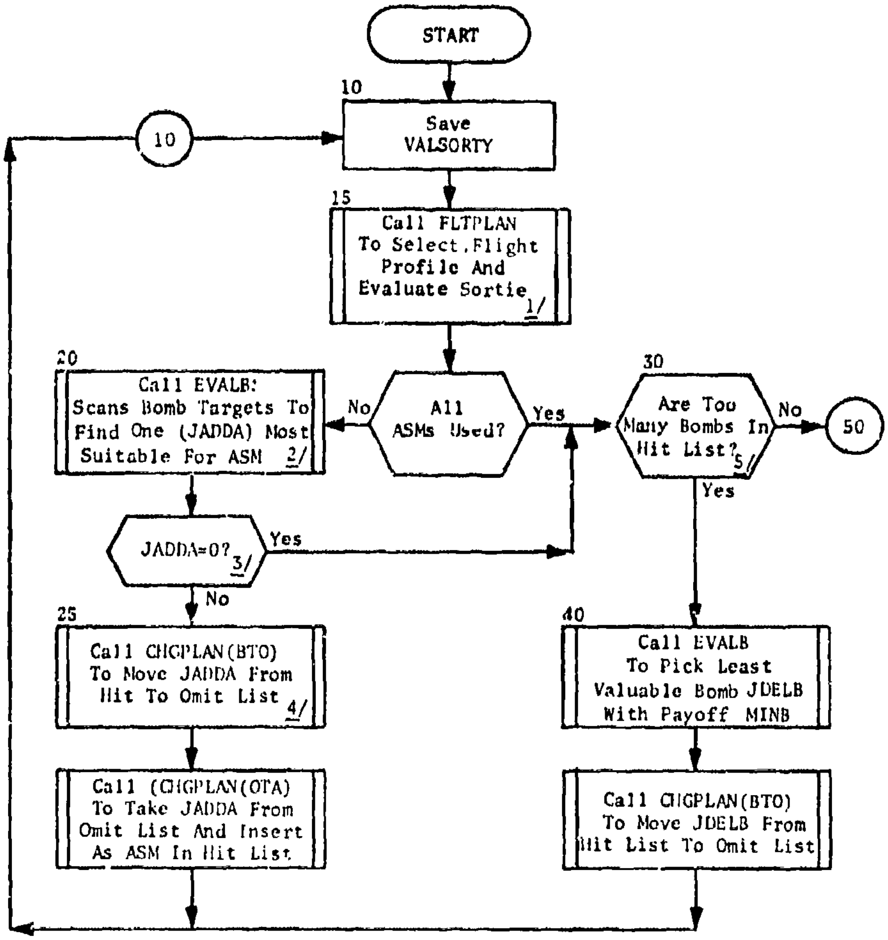


Fig. 151. Subroutine SORTOPT
Part I: Gets Strikes Set Up for Available Bombs and ASMs

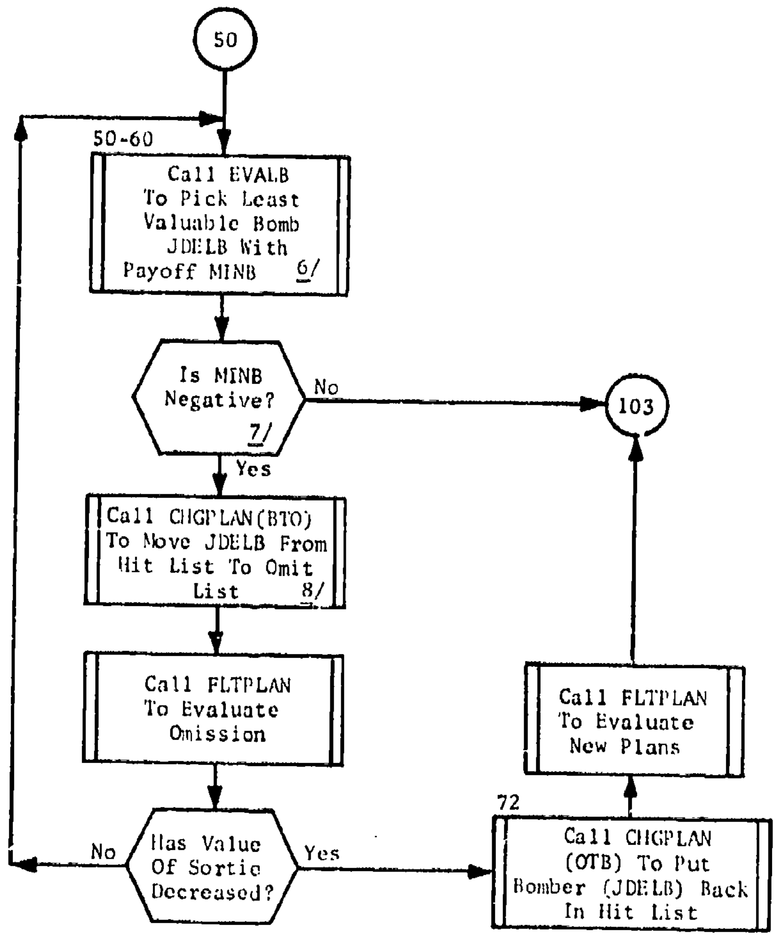


Fig. 151. (cont.)
Part II: Considers Omitting Bombs or Recovery to Improve Sortie

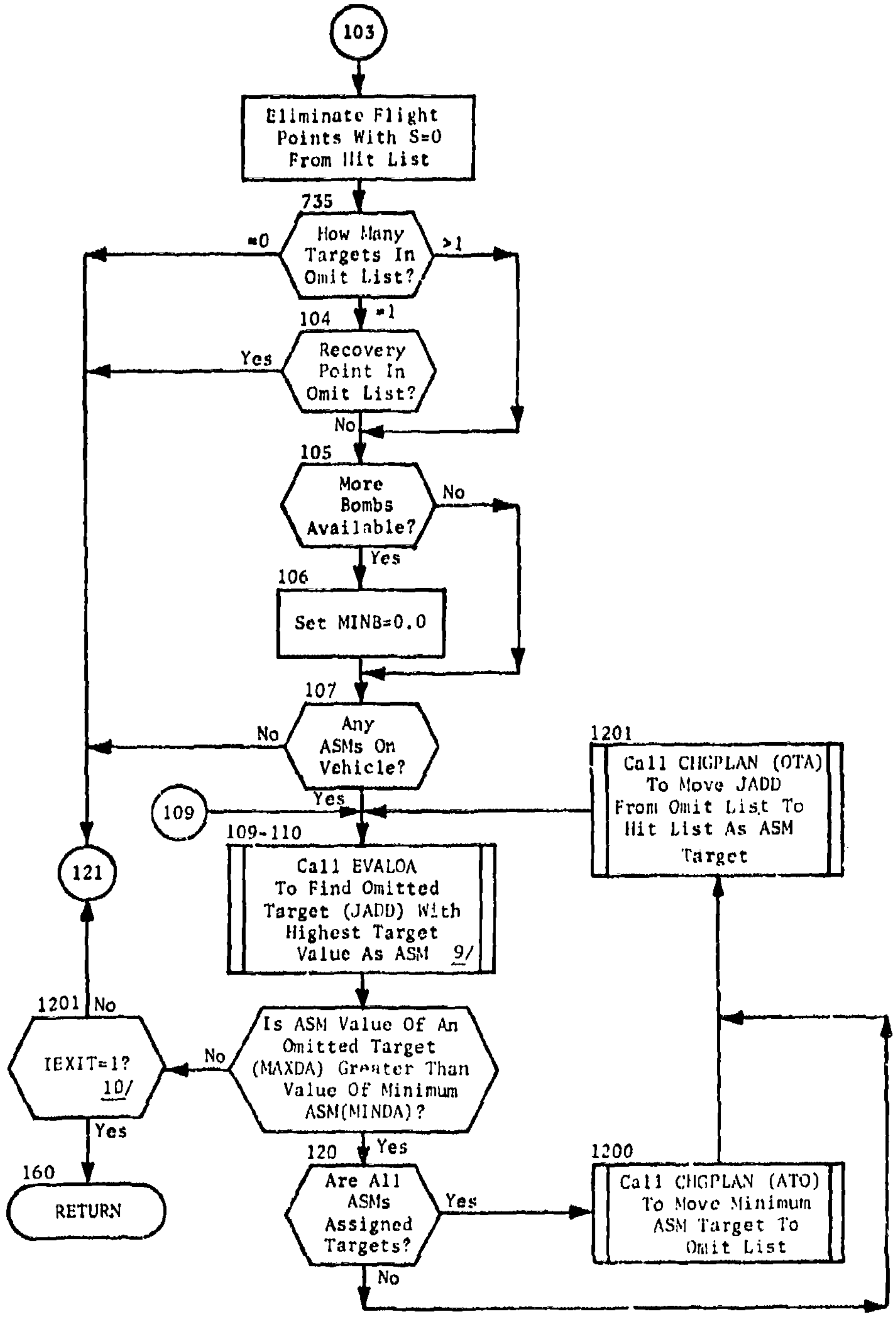


Fig. 151. (cont.)
Part III: Considers Substituting Targets in Omit List for Current ASM Targets

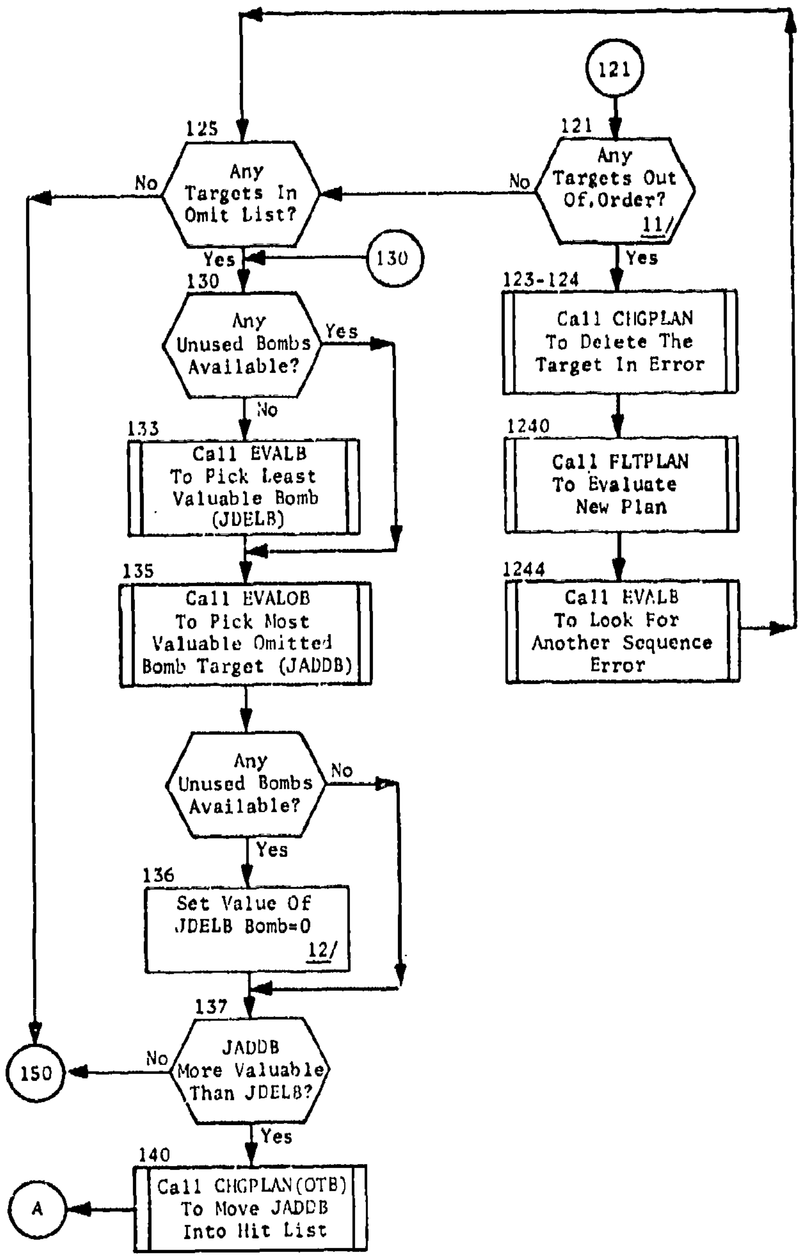


Fig. 151. (cont.)
Part IVA: Checks Target Order and Considers Substituting Omitted Targets for Current Bomb Targets
(Sheet 1 of 2)

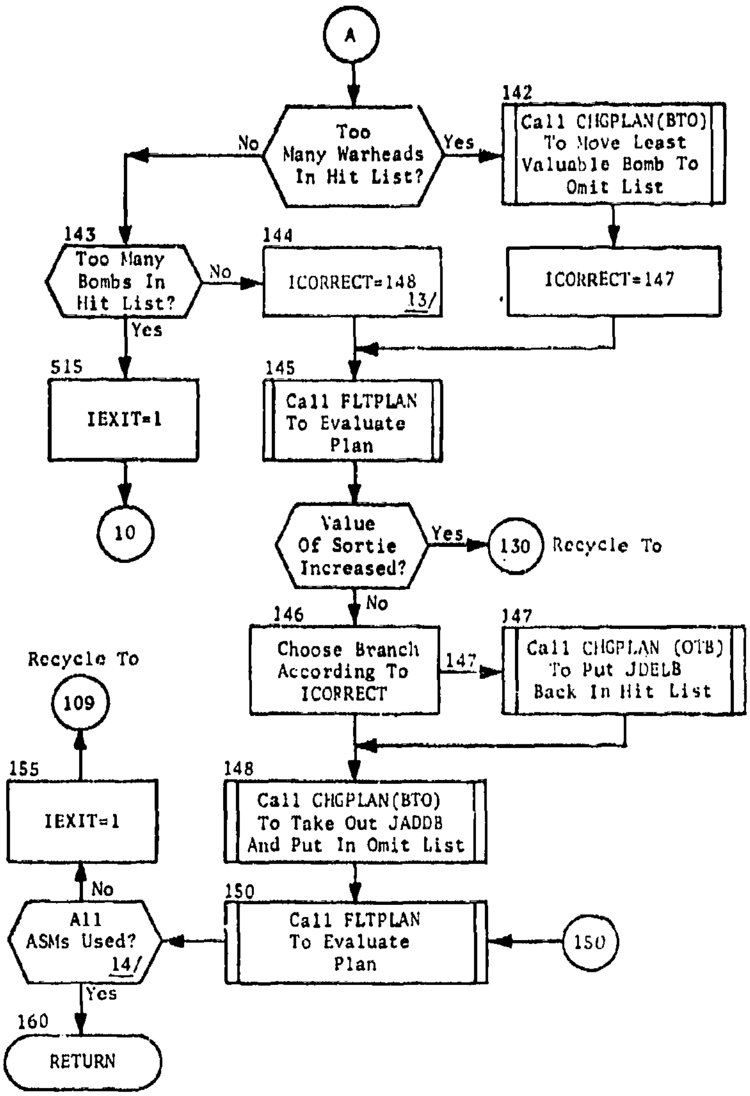


Fig. 151. (cont.)
Part IVB: Checks to Make Sure Any Substitution of Bombs Was an Improvement. If Not, Corrects Error (Sheet 2 or 2)

Notes:

1. Before calling FLTPLAN, variable OLDVAL is set equal to VALSORTY, the value of the sortie obtained on the last call on FLTPLAN. Each time a change is made, a check is made to see if the value of the sortie has increased. A variable EPSVAL is kept, which is equal to .001 times the value of the sortie. If a potential change will affect the value of the sortie by less than EPSVAL, the change is not made (to avoid looping because of rounding errors).

2. The variable ISCAN is set before calling EVALB to indicate whether or not EVALB should evaluate the sortie recovery. If it should, ISCAN is set to LASTPAY, which is the index of the last "paying" point in the hit list. This is the recovery point, or the last target if recovery has already been omitted. If EVALB is being called to evaluate only the bombs, ISCAN is set to LASTTGT which is always the last target in the hit list. At this point on the flowchart, EVALB is called to consider putting an ASM instead of a bomb on a target, so ISCAN = LASTTGT.

3. If EVALB cannot find a target suitable for an ASM (i.e., within range of any other route point), it returns JADDA = 0.

4. JDO and JAFT are variables in common /CHGPLAN/ which give CHGPLAN the index of the target to be moved, and the index of the target it is to follow in the hit list. In order to change from a bomb to an ASM on the same target, we call CHGPLAN twice, once to delete the bomb and once to insert the ASM.

5. If there were originally more targets assigned to the sortie than there were bombs available, the extras are not dropped until after the ASMs have been assigned. Then the least valuable remaining target is dropped from the list, to leave the right number of targets for the bombs.

6. In this phase of SORTOPT, we are looking for omissions which will increase the value of the sortie. ISCAN is set to LASTPAY so that recovery, as well as the targets, is considered for omission.

7. Again, MINE is actually required to be less than -EPSVAL, to be worth omitting. (See note 1.)

Fig. 151. (cont.)
Part V: Notes
(Sheet 1 of 2)

8. Before removing a target which seems to have a negative value, its position in the hit list is stored in JAFTXX, to facilitate reinserting it if its omission in fact decreases the value of the sortie.

9. Since EVALOA calculates the differential value to the sortie of an ASM on each target in the omit list, this need be done only once for the sortie. KALC is set to 0 the first time EVALOA is called, and then it is set to 1 to bypass the calculations on subsequent calls.

10. If the processing of Part IV of SORTOPT has already been done and this was a return pass to Part III, IEXIT equals 1 and the program exits at this point.

11. EVALB checks to see if it is possible to shorten the distance flown by interchanging the sequence of any two targets in the list. If so, JSEQERR is used to indicate to SORTOPT which target should be deleted. The target is not reinserted in its proper position immediately, but is left in the omit list. When all sequence errors have been "corrected" by deleting one of the targets out of order, processing continues. It is left to EVALOB to make the correct choice from the omit list to use up the extra bombs and insert the targets in their correct positions.

12. If there are extra bombs available at this point, we want to insert any target of value from the omit list, even if it is of less value than the others already in the list. MINB (the minimum value of a bomb in the hit list) is set to -EPSVAL so that when the test MAXOB.GT.MINB+EPSVAL is made, if the maximum omitted bomb (MAXOB) has any value at all, it will be picked up.

13. ICORRECT is set to indicate whether a bomb was only omitted, or whether one was omitted and one was inserted, so that if the change was a mistake it can be reversed.

14. If there is an ASM launched from the target being omitted, it is necessary to find a new launch point for the ASM. If there is not one within range, the ASM target is omitted as well as the bomb. Thus it is possible that at this state there may again be ASMs without targets, so SORTOPT recycles Part III to use up its ASMs. If this happens, IEXIT is set to 1 to indicate that processing is complete when the ASMs are assigned.

Fig. 151. (cont.)
Part V: (cont.)
(Sheet 2 of 2)

SUBROUTINE TGTASGN

PURPOSE:	To make the initial assignment of targets to all sorties in a flight.
ENTRY POINTS:	TGTASGN
FORMAL PARAMETERS:	None
COMMON BLOCKS:	ARAYSIZE, CURRAID, DEBUG, FIXALL, INDEX, IRESRCH PCALL, 3, PRINTOPT, RAIDSHR, 2, 4, TGTASGN
SUBROUTINES CALLED:	PRINTIT, PRNTF, TIMEME
CALLED BY:	FLTROUTE

Method

TGTASGN is called by FLTROUTE to carry out the actual assignment of strikes to each sortie in a flight. The first and last sortie for the flight are specified by FLTROUTE (IFSTVEH and LSTVEH in common /TGTASGN/) together with a flag (ISIDE) which notes whether targets are to be assigned from the left side of the corridor toward the middle or from the right side of the corridor toward the middle. For each side of the corridor, TGTASGN maintains a pointer to the first unassigned target so that the scanning of targets to be assigned can begin with this target.

Figure 152 illustrates the operation of TGTASGN. For each new sortie the "target limit" (i.e., the total number of targets that should be assigned up to and including the sortie) is increased by the number of warheads on the sortie (NWPV) multiplied by the average number of strikes per warhead for the corridor (TGTSPWHD). Since the allocator has assigned more strikes than it has weapons, the number of strikes per warhead is usually a little more than 1.0. Therefore (because no fractional strikes can be assigned) the number of strikes per sortie will vary and an occasional sortie will be assigned an extra strike in excess of the available warheads.

Specific strikes are then selected for the sortie, beginning with the first unassigned strike from either the top or bottom of the list depending on the value of ISIDE, until the target limit is exceeded.

Each strike selected is tested against any already assigned to the sortie to be sure that another strike on the same target is not included. If the same target is encountered, the strike is simply passed over. When an acceptable strike is encountered, it is inserted in the strike list for the sortie in such a position that the list remains in order of ascending values of RHO for the targets.

Subroutine TGTASGN is illustrated in figure 152; sheet 2 of the figure consists of notes to facilitate interpretation of the flowchart.

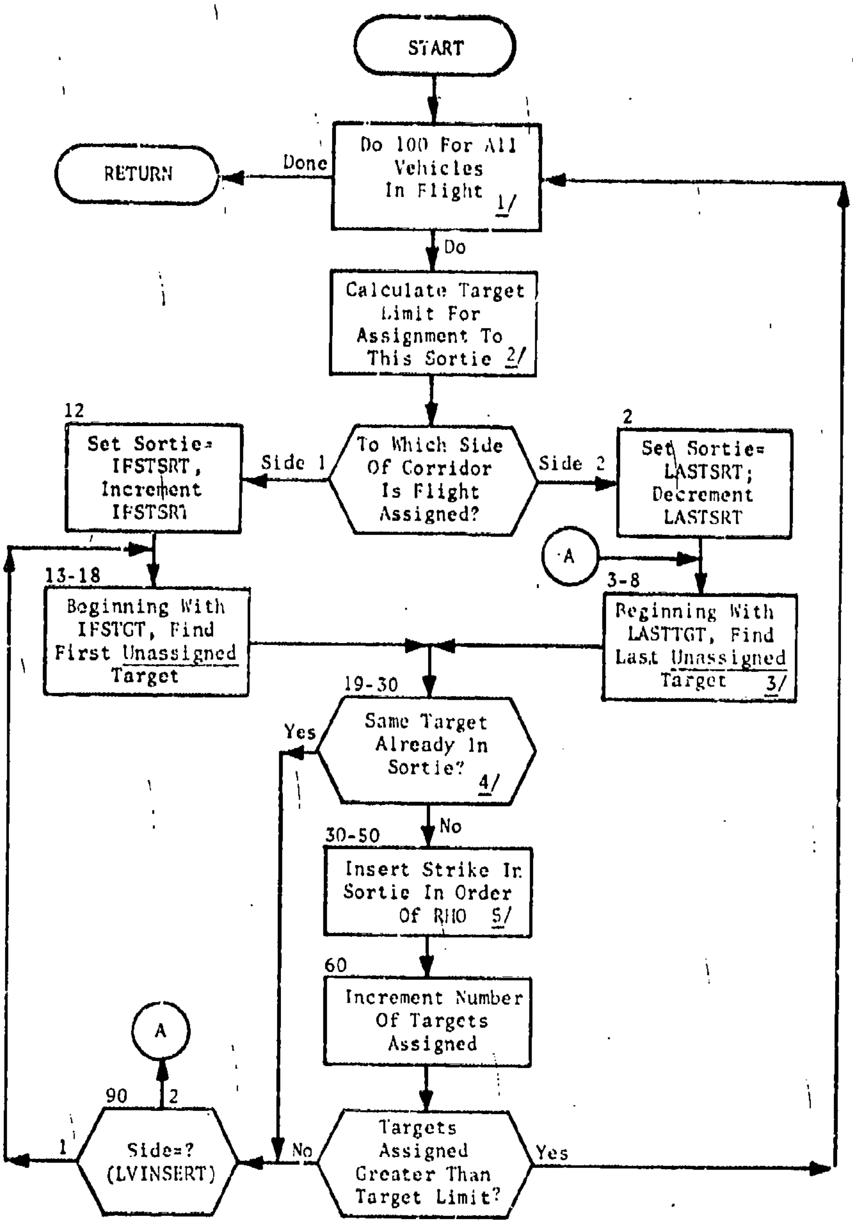


Fig. 152. Subroutine TGTASGN
(Sheet 1 of 2)

Notes:

1. TGTASGN receives from FLTROUTE the variables IFSTVEH and LSTVEH, indicating which sorties are to be processed. It initializes variables IFSTSRT and ILASTSRT to those values and uses them as internal markers.

2. TGTLIM, the target limit, is calculated as a floating point number. Therefore, since TGTSPWHD is usually slightly greater than 1.0, occasionally a sortie will be assigned more targets than it has warheads. These extra targets will be dropped in the sortie optimization and will be considered for substitution in other sorties.

3. When TGTASGN is entered, IFSTGT and ILASTGT are set at the first and unassigned target, respectively. MYASGN is initially set for all targets. As soon as TGTASGN "looks" at a ta t sets MYASGN to 0. If it assigns the target to a sort it sets MYASGN to 1. Thus the first unassigned target may be 0, if the target has been considered previously and rejected, or -1, if it is being looked at for the first time.

4. The allocation may assign more than one weapon from the same group to a target when this happens; it is not desirable to attempt more than one hit on the target by the same sortie. Therefore, TGTASGN must check each target already in the list each time it assigns a new target.

5. The list of targets for a given sortie are listed in order of RHO as an initial attempt at determining the flight route of the sortie. This order is subject to change by EVALB which does a sequence check during its evaluation.

Fig. 152. (cont.)
(Sheet 2 of 2

CHAPTER 8
PROGRAM PLNTPLAN

PURPOSE

PLNTPLAN processes the bomber and missile plans given it on the STRKFILE by program POSTALOC, and writes them with tanker plans to the EVENTAPE in a format required by the QUICK Simulator. In addition, a detailed plan is output via the PLANTAPE which reflects the plan in a form more suitable for hardcopy output. The PLANTAPE is used as input to programs INTRFACE and EVALALOC. Detailed prints of the final plans may also be obtained from PLNTPLAN. Each type of plan (bomber, missile, and tanker) is handled differently. Among the processing functions performed on the input bomber plans are: assigning refuel areas; calculating ASM launch points; determining where zone crossings, altitude changes, and decoy launch points should occur; and coordinating launch times according to user parameters.

For missiles, launch times are assigned based on user-supplied coordination parameters. Tanker plans are generated such that all bombers will be serviced as required. Finally, PLNTPLAN calculates the distances and times between all events of each plan.

INPUT FILES

Three files are input to program PLNTPLAN: the STRKFILE, BASFILE, and MSLTIME file.

The STRKFILE is the output of program POSTALOC, and contains the skeletal plans for each bomber and missile. Common block /OUTSRT/ contains the input bomber record; common /BLOCK/ contains the input record for missiles. The end-of-file signal is a dummy bomber record with a group number of 201.

For bombers which refuel, two records are provided; the first is the primary plan, the second describes the plan to be used in the event of a refuel abort. The input event list in common /OUTSRT/ includes all targets, whether bomb or ASM. It always begins with the corridor origin route leg, and ends with the input event DEPEN. LAND, or DIVEMISL (if an air-breathing missile), depending on whether the mission is successful or aborts. Table 37 lists the admissible input events by type, and indicates what information in the list is relevant for each type.

Table 37. List of Admissible Input Events by Type and Information Relevant to Each

TYPE OF HAPPENING	LAT., LONG	PLACE	WEAPON OFFSET LAT., LONG.
DOGLEG	Y	N	N
DROPBOMB	Y	Target Index	Y
AIM ASM	Y	Target Index	Y
USEDECOY	Y	N	N
DEPEN	N	Depenetration Corridor Index	N
LAND	N	N	N
DIVEMISL	N	N	N

Y = relevant

N = not relevant

Bomber sorties are identified uniquely by sortie number, group index, and corridor index together. Missiles are identified by index number.

The BASFILE, created by program PREPALOC, provides such information as corridor, defense zone, payload, weapon, and tanker data. It is from this file that PLNTPLAN retrieves the data required to fill the following common blocks:

/MASTER/

/FILES/

/CORRCHAR/

/ASMTABLE/

/PAYLOAD/

/DPENREF/

/PLANTYPE/

/WPNTYPEX/

/WPNGRPX/

/NAVAL/

/9/ (variables corresponding to /BOUNDARY/, /CHARTER/, and /HAPPEN/ in other QUICK programs)

/7/.

The MSLTIME file, created by program ALOC, contains a five-word record for each fixed missile (i.e., each missile-delivered weapon). It is read into array MSLFIL of common block TIMELINE. If there are no fixed missile assignments in the current plan, it consists simply of an end-of-file record.

OUTPUT FILES

The principal output of PLNTPLAN is the EVENTAPE, containing the bomber, missile, and tanker plans for use by the QUICK Simulator.

Bomber and tanker plans are written from common block /INDATA/, where the main part of the plan is contained in the History table. Each line lists an event by type, time, and place. The admissible events for the EVENTAPE record are those listed in table 38 (see exception noted for the last three events listed). The name for each event used in common /EVENT/ and in the output prints is also given in the table.

Events for each bomber and tanker record are listed in the History table in proper order. With each event the time given is the increment since the previous event, except for the first event--the Launch event. Here the "previous" event is taken to be the inception of the plan and the time increment is measured from it. An auxiliary table called the Warhead table is also included in the plan. This lists for each weapon the type of warhead, the weapon offset, if any, and height of burst. There is an entry in the Warhead table for every Local Attrition (drop bomb) or Launch ASM event.

Table 38. Bomber Events Recognized by PLNTPLAN

TYPE OF EVENT	EVENT TYPE	PLACE INDEX	EVENT NAMES USED IN /EVENTS/	EVENT NAMES USED IN OUTPUT PRINT
Launch	2	Base Index	LAUNB	LAUNCH B
Refuel	4	Refuel Index	LEREFUEL	REFUEL
Enter Sector	5	Sector	INSECTOR	INSECTOR
Local Attrition or Drop Bomb	8	Target Index	LOCLATTR	DROPBOMB
Launch ASM	14	ASM Type	LAUNASM	LAUN ASM
ASM Target	--	Target Index	--	ASM TGT
Launch Decoy	15	*	LAUNDCOY	LAUNDCOY
Change Altitude	17	1 for Go High, 0 for Go Low	LOHI	CHANGALT
Recover	16	Recovery Base Index	LANDHO	RECOVER
Abort	13	*	LABORT	ABORT
Enter Refuel	11	Refuel Area Index	LENTEREF	ENTERREF
Leave Refuel	12	*	LEAVEREF	LEAVEREF
Go High**	18	*	IGOHI	GO HIGH
Go Low**	19	*	IGOLOW	GO LOW
Dogleg**	20	*	LEGDOG	DOGLEG

*Place index not applicable

**Appears only in detailed version of plan on PLANTAPE

In the case where the bomber refuels, the History table contains two plans. The first is the "primary mission" to be executed after successful refueling. The second is an alternate plan, to be executed in the event that the last scheduled refueling is unsuccessful.

The "final" plan supplied to the Simulator does not contain references to specific geographic locations, i.e., no latitudes or longitudes are given. The geographic location of each event is available to PLNTPLAN and is used by it to compute distances and their associated time increments. This results in a "detailed" plan. PLNTPLAN then drops all references to geographic location, including dogleg events for the EVENTAPE.

Missile records are written from common /BLOCK/. In addition, records indicating a time-dependent destruction before launch event will precede the events for the first consecutively launched vehicle from a base of weapons having a time-dependent DBL probability.

The formats for all plan record types on the EVENTAPE are shown in tables 39 through 42. Each record is preceded by a one-word record length field. A zero length field indicates the end of plan records.

After the plan records, data on the recovery bases used by the bombers are placed on the EVENTAPE. These tables also are shown in table 43.

The PLANTAPE is optionally available to store the detailed plans for input to programs INTRFACE and EVALALOC. The information for the PLANTAPE is retrieved from the arrays of common blocks /INDATA/, /IOUTOLD/, /DINDATA/, and /DINDATA2/ for bombers and tankers. For missiles, it is written from variables within common /BLOCK/. The tape format for these records is shown in tables 44 through 46. The end of this first file (containing missile, bomber and tanker plans) is signalled by a dummy record containing 24 words of zeros, followed by the signal LAST in word 25. This, in turn, is followed by an END FILE mark.

The second file on the PLANTAPE contains the refuel area table. The first word is the number of refuel areas (NRF). Then follow NRF word pairs containing the latitude and longitude of each refuel area.

The PLANTAPE is written using standard FORTRAN unformatted WRITE statements rather than the filehandler subroutines.

Table 39. Format of EVENTAPE Records
(Bomber Plan Record)

WORD	DESCRIPTION
1	Side (BLUE = 1, RED = 2)
2	Launch base index
3	Vehicle index
4-5	Reserved
6	Naval kill probability (PKNAV)
7	NAVAL if bomber has time-dependent DBL probability
8	Reserved
9	Bomber type index
10	Target class (BOMBER = 2)
11	Launch region
12	Alert status (ALERT = 1, NONALERT = 2)
13-15	Reserved
16	Current altitude (HIGH = 1, LOW = 0)
17	Electronic countermeasure index
18-20	Reserved
21	Number of lines in History table (N)
22	Number of lines in Weapon table (M)
23-24	Reserved
25	Number of lines in primary mission
26	Reserved
27-A	History table (A = 26+(3*N)), containing 1) Time of event (1st N words) 2) Place index of event (2nd N words) 3) Event type code (3rd N words)
(A+1)-8	Weapon table (B = A+(4*M)), containing 1) Warhead type (1st M words) 2) X-coordinate offset (2nd M words) 3) Y-coordinate offset (3rd M words) 4) Height of burst (4th M words)

Table 40. Format of EVENTAPE Record (Missile Plan)

WORD	DESCRIPTION
1	Side (BLUE = 2, RED = 2)
2	Launch base index
3	Reserved
4	Time of launch
5	Payload index
6-7	Reserved for future use
8	Number of MIRVs per missile
9	Missile type index
10	Target class (MISSILE = 1)
11	Launch region
12	Alert status (ALERT = 1, NONALERT = 2)
13	Number of missile (N)
14	2 x number of targets (M)
15-A	N missile indices (A=14+N)
(A+1)-B	N silo indices (B=A+N)
(B+1)-C	Target data flight times* (C=B+M)

*Up to 270 words appear in pairs, one pair for each target, with the following format:

1st word:

47	33	21	9	6	3 0
INTAR	DGX	DGY	DHOB	NARDEC	NTDEC
15 bits	12 bits	12 bits	3 bits	3 bits	3 bits

INTAR - Target index number

DGX - Offset distance for latitude } (fiftieths of

DGY - Offset distance for longitude } nautical miles)

DHOB - Height of burst code

NARDEC - Number of area penetration decoys

NTDEC - Number of terminal penetration decoys

2nd word: Time of flight from launch to target

Table 41. Format of EVENTAPE Record (Tanker Plan)

WORD	DESCRIPTION	
1	Side (BLUE = 1, RED = 2)	
2	Launch base index	
3	Vehicle index	
4-8	Reserved	
9	Tanker type index	
10	Target class (TANKER = 3)	
11	Reserved	
12	Alert status (ALERT = 1, NONALERT = 2)	
13-20	Reserved	
21	Number of lines in History table (=7)	
22-24	Not used	
25	Number of lines in primary mission (=7)	
26	Not used	
27-33	Time of event	History Table
34-40	Place of event	History Table
41-47	Event type code	History Table

Table 42. Format of EVENTAPE Record (Time Dependent DBL Destruct Event)

WORD	DESCRIPTION
1	Side (Hollerith name)
2	Base index
3	Index to DBL data tables
4-6	Zero
7	Event identifier (=5HNAVAL)
8	0
9	Weapon type index
10	Weapon class index
11	Weapon launch region
12	Weapon alert status

This record precedes the Launch event for the first consecutively launched vehicle from a base of weapons having a time-dependent destruction before launch probability.

Table 43. Format of EVENTAPE Recovery Tables

WORD	PLNTPLAN FORTRAN NAME	DESCRIPTION
1	NWORDS	Zero record length field indicating end of plan record
2-201	INDBAS (50,4)	Recovery base index numbers (four bases per depenetration corridor)
202-401	INDCAP (50,4)	Capacity of recovery bases
402-601	TOF (50,4)	Time of flight from depenetration point to each associated recovery base

462-546 O - 72 - 23

Table 44. Format of PLANTAPE Record
(Bomber Plans)
(Sheet 1 of 2)

Header Block:

WORD	DESCRIPTION
1	Side
2	Group number
3	Penetration corridor number
4	Bomber sortie number
5	Base index number
6	Vehicle index number
7	ICLASS = 2
8	Weapon type
9	Launch region
10	Alert status
11	Payload index
12	Depenetration corridor number
13	Total number of bomber events in table
14	Number of planned bomber events (i.e., excluding Refuel/Abort mission events)
15	Available low-altitude range in precorridor legs
16	Available low-altitude range before first target
17	Available low-altitude range after first target
18-19	Lower plot markers for sortie
20-21	Upper plot markers for sortie
22	Weapon type name (ISIMTYPE)
23	Bomber type (Plan Generator type)
24	Bomber function code
25	Number of targets in total plan (set to 4HLAST after last good record as an end sentinel)

Table 44. (cont.)
(Sheet 2 of 2)

Plan Information Blocks:

One block for each event in plan (regular or refuel abort)

WORD	DESCRIPTION	
1	Time increment since last event	
2	Place index	
3	Event type	
4	Latitude of event	
5	Longitude of event	
6	Offset latitude	for weapon delivery
7	Offset longitude	for weapon delivery
8	Warhead index	
9	Damage expectancy	
10	Cumulative time to event	

Target Information Block:

One block for each weapon delivery

WORD	DESCRIPTION
1	Index number for target
2	Target designator code
3	Target task code
4	Target country location code
5	Target flag code

Table 45. Format of PLANTAPE Record
(Missile Plans)
(Sheet 1 of 2)

Header Block:

WORD	DESCRIPTION
1	Side
2	Group index
3	Zero
4	Missile record counter
5-6	Zero
7	ICLASS (=1)
8	Missile type (ISIMTYPE)
9	Launch region
10	Alert status
11	Payload index
12	Zero
13	Number of missiles
14	Number of targets
15	Time of launch in hours
16-21	Zero
22	Warhead type
23	Missile type (Plan Generator type)
24	Missile function code
25	End sentinel (=4HLAST after last good record; zero otherwise)

Target Information Blocks:

One block for each target in plan

WORD	DESCRIPTION
1	Flight time
2	Site index

Table 45. (cont.)
(Sheet 2 of 2)

WORD	DESCRIPTION
3	Missile index
4	Target latitude
5	Target longitude
6	Weapon site latitude
7	Weapon site longitude
8	Warhead type
9	Reliability
10	Target index number
11	Target designator code
12	Target task code
13	Target country location code
14	Target flag code

Table 46. Format of PLANTAPE Record (Tanker Plans)

Header Block:

WORD	DESCRIPTION
1	Side
2	Group number
3	Zero
4	Sortie number
5	Base index number
6	Vehicle index number
7	ICLASS = 3
8	Weapon type index
9	Zero
10	Alert status
11-12	Zero
13	Total number of events
14-24	Zero
25	End sentinel (=4HLAST after last good record; zero otherwise)

Plan Information Blocks:

One block for each event in plan

WORD	DESCRIPTION
1	Time increment since last event
2	Place index
3	Event index
4	Latitude of event
5	Longitude of event
6	Cumulative time to event

CONCEPT OF OPERATION

Figure 153 gives the overall macro flowchart for program PLNTPLAN. Initialization includes reading the BASFILE and the user data cards. Then the first record from the STRKFILE is read in, and the main processing begins. Whenever a missile plan is read, subroutine PLANTMIS is given control until another bomber record is encountered. (A bomber record with a group number of 201 is the end-of-file indicator.)

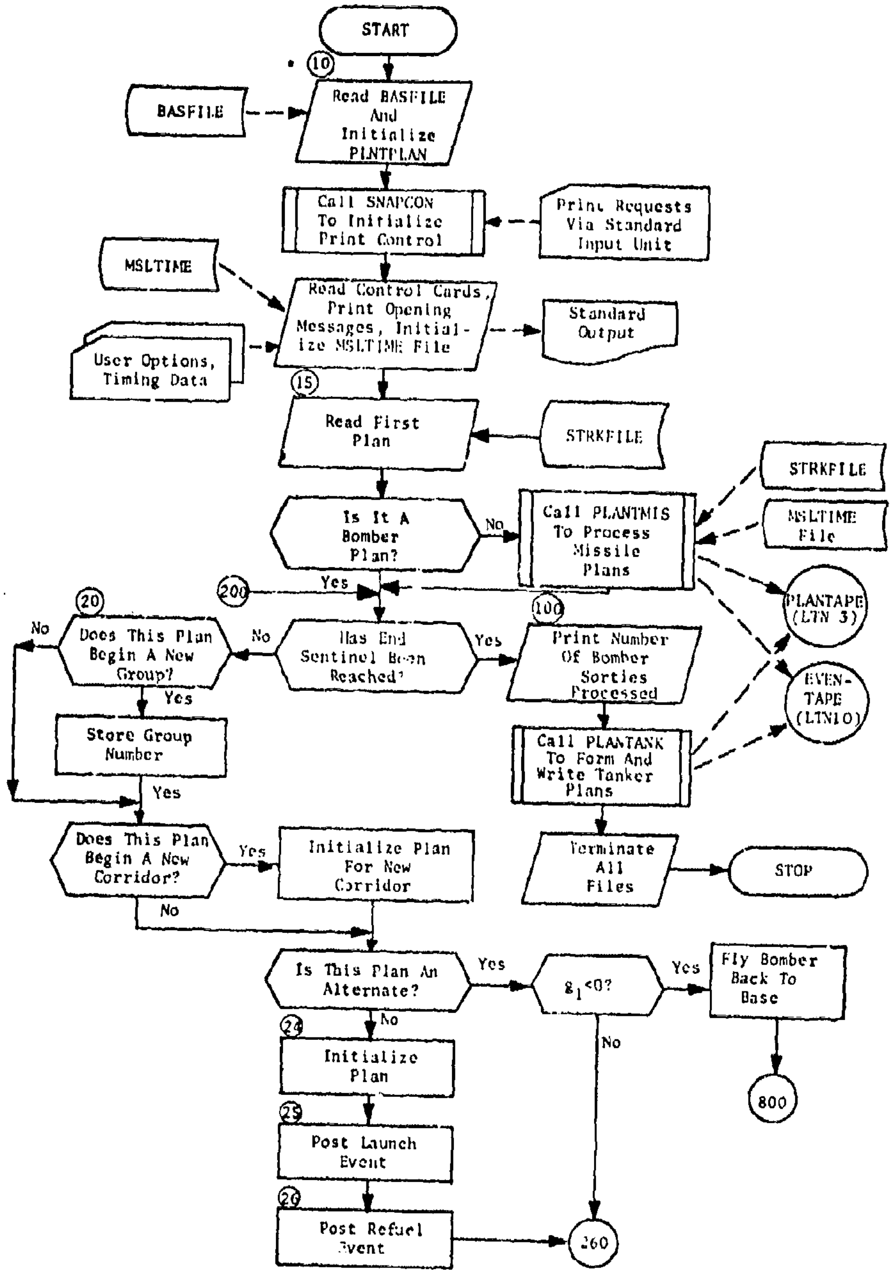
Since subroutine PLANTMIS controls all processing of missile plans, and subroutine PLANTANK controls the generation of all tanker plans, the relevant concepts are discussed under those two subroutines. Bomber processing, however, is as follows.

Figure 154 shows a typical path a bomber would take between the time of its launch and its recovery. The bomber is launched from a base, flies to a refuel point or area if refueling is called for, then to a corridor entry point. It may then fly one or more prespecified doglegs (called precorridor legs) which define a penetration route before reaching the point labeled Corridor Origin. From the origin it flies over the target area and its assigned targets in their proper order. It then enters the depenetration corridor which may also consist of one or more doglegs. From there it flies to the recovery point or base. At any point after the corridor entry it may cross a boundary line between defense zones.

This path may logically be divided into four parts: (1) the launch and refuel portion, (2) the precorridor legs, (3) the target area which is the main part of the plan, and (4) the depenetration and recovery portion.

In PLNTPLAN, each bomber sortie is processed in much the same order as it is flown; that is, first the precorridor section events are posted, then those of the target section, and finally, the depenetration and recovery section events. Besides the posting of the target events themselves, the main processing consists of posting events for changes of altitude, zone crossings, and decoy launches. All postings for bomber events are made in the arrays of common /DINDATA/.

After each bomber plan has been evaluated, and its data stored, PLNTPLAN reads in the next STRKFILE record. If the new record is the alternate plan for the sortie just processed, the alternate events are posted before the EVENTAPE or PLANTAPE is written. Otherwise, the completed plan is output, and processing begins on the new plan.



* Circled numbers refer to start of coding blocks rather than to statement numbers.

Fig. 153. Program PLNTPLAN (Macro flowchart)
(Sheet 1 of 3)

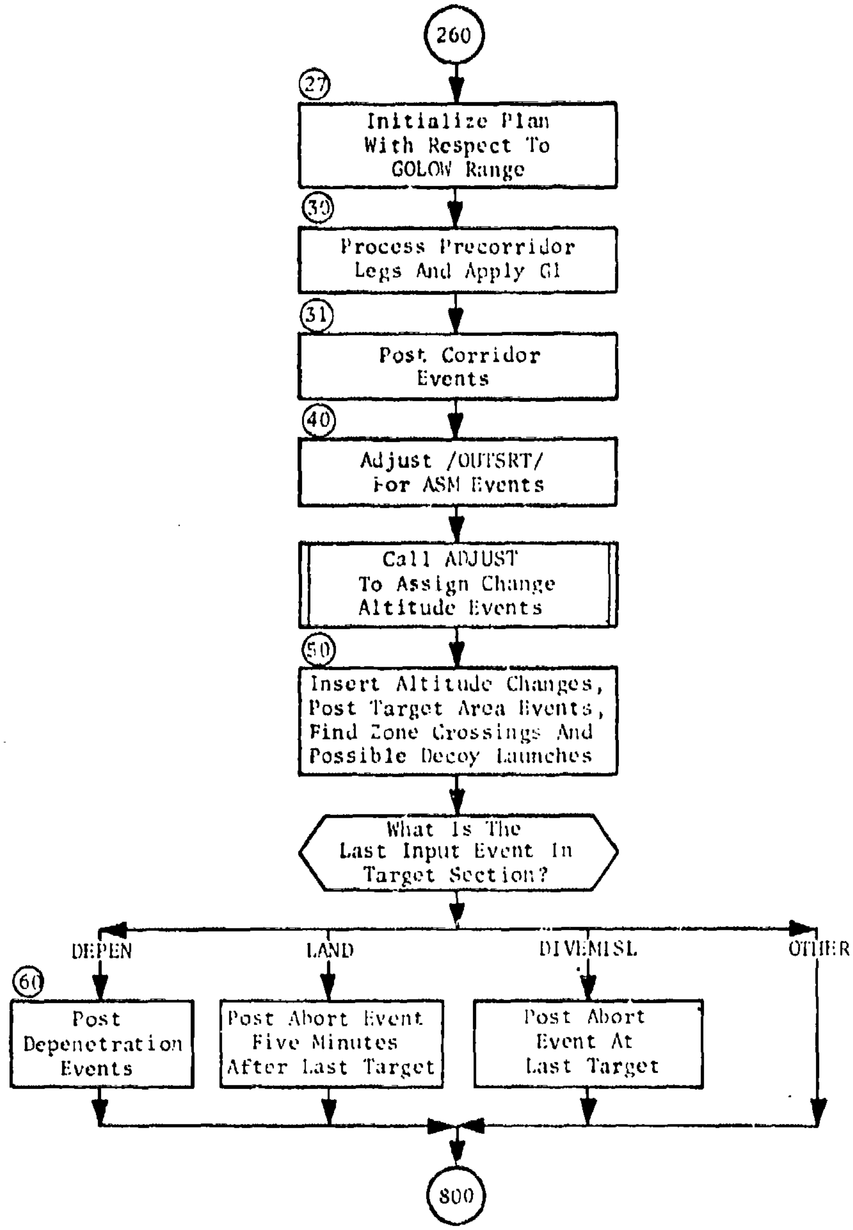


Fig. 153 (cont.)
(Sheet 2 of 3)

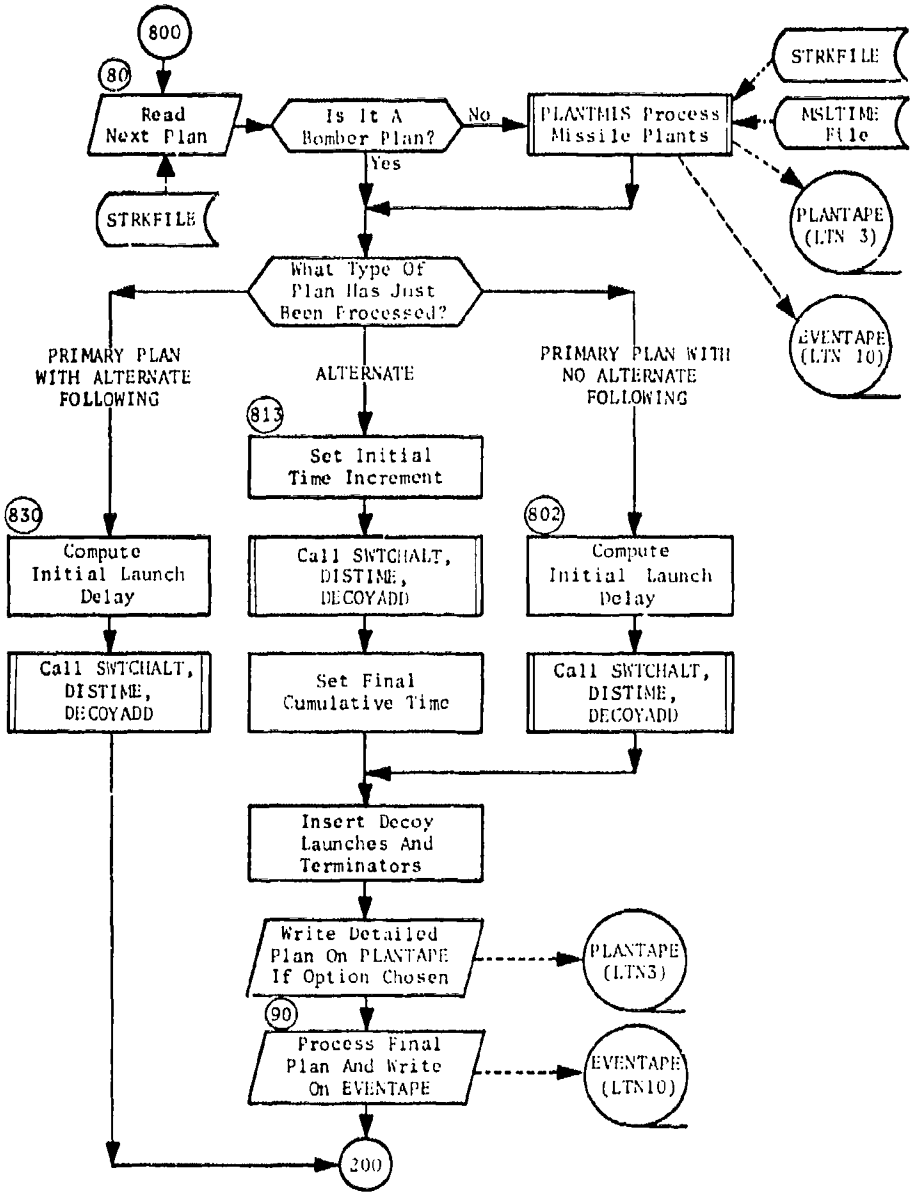


Fig. 153. (cont.)
(Sheet 3 of 3)

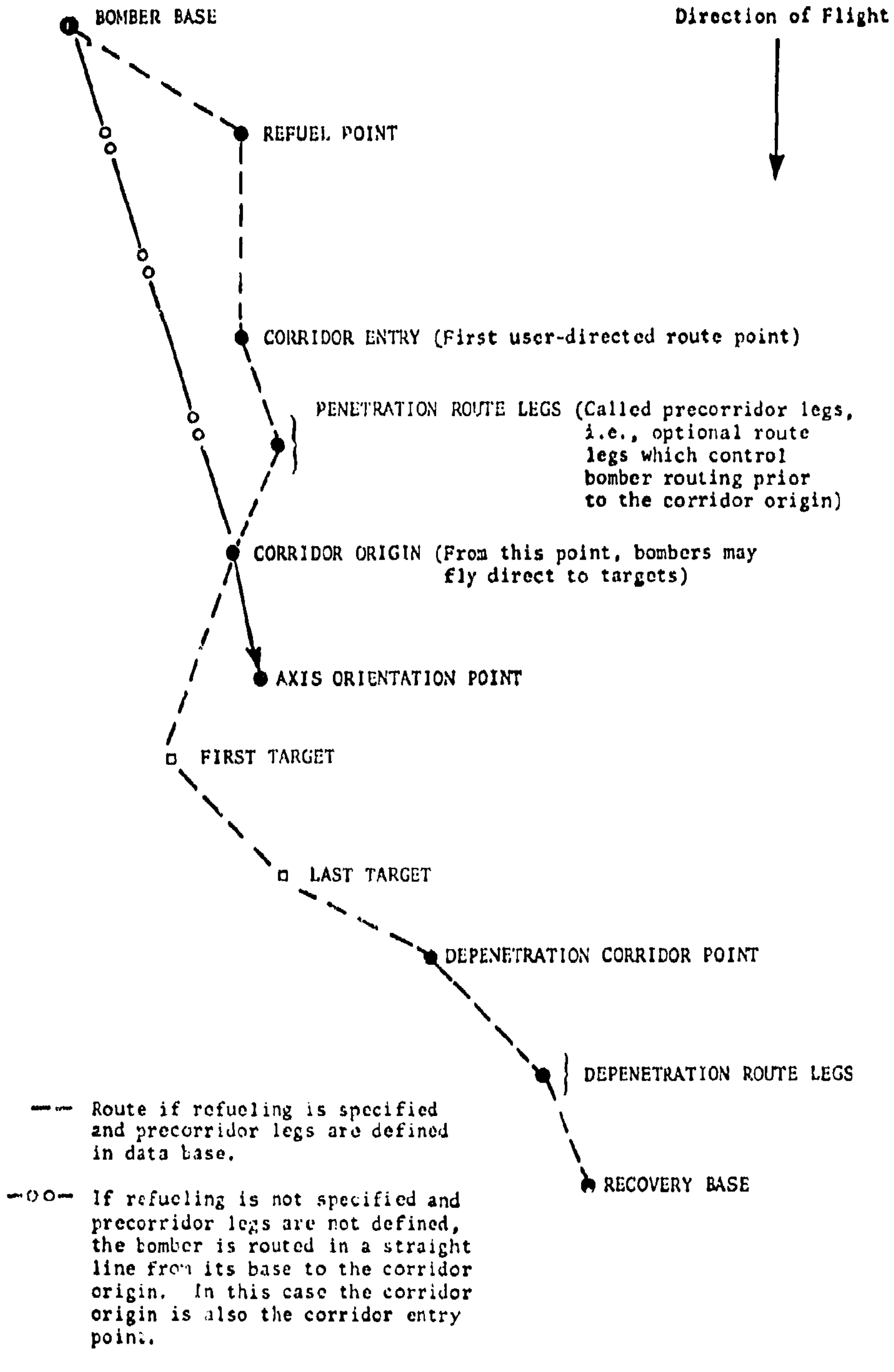


Fig. 154. Path of Typical Bomber Sortie

Tanker plans are generated by subroutine PLANTANK after all bomber and missile plans have been completed. All files are then terminated, and PLNTPLAN exits.

To facilitate discussion, program PLNTPLAN is divided into "blocks" of coding as noted in the macro flowchart. The remaining program description as well as the detailed flowcharts* are organized around these blocks, which are:

BLOCK 10 - Program initialization
15 - Control loop
20 - Determine type of plan
24 - Initialize plan
25 - Post Launch event
26 - Post Refuel events
27 - Initialize plan with respect to GOLOW range
30 - Process precorridor legs and apply GOLOW-1
31 - Post corridor events
40 - Adjust /OUTSRT/ for ASM events
50 - Apply GOLOW-2 before first target
60 - Post depenetration events
80 - Read next /OUTSRT/ record, convert last one
90 - Process final plan and write on EVENTAPE
100 - Program termination

Block 10: Program Initialization (figure 158)

In addition to initializing program variables, coding block 10 reads all required data from the BASFILE, and all user control cards, calling subroutine SNAPCON for the print request cards and subroutine LNCHDATA for the missile timing cards. Files are initialized, and preliminary information is printed.

Block 15: Control Loop (figure 159)

The first bomber plan is then read in from the STRKFILE, and the main processing of PLNTPLAN begins. If a missile plan is read, subroutine PLANTMIS receives control. Otherwise, the number of events is checked and the sortie INDEX is computed. Subroutine SNAPCON is called to

*Figures 158 through 172 are at the end of this section.

determine which prints are to be active for this plan. If the program is to be terminated at this sortie (print request 14), a branch is taken to the termination block.

Block 20: Determine Type of Plan (figure 160)

Each bomber plan, including the first, is immediately checked to see if it is a sentinel record indicating that the end of the strike file has been reached. Otherwise, tests are made to see if the current plan begins a new group or corridor. If so, PLNTPLAN is appropriately initialized. A check is then made to see whether the current plan is an alternate plan. If so, plan initialization in coding block 24, as well as the posting of LAUNCH and REFUEL events in blocks 25 and 26, are skipped. If the low-altitude distance G_1 is negative for an alternate plan, then the unrefueled bomber cannot reach the target area; hence the alternate plan is replaced by the event RECOVER at the launch base.

Block 24: Initialize Plan (figure 161)

The plan initialization at block 24 consists of setting pointers and indices for the appropriate depenetration corridor and writing other information into the final plan. Indices for the Payload table and ASM table are set at this point, as are parameters which are dependent on bomber speed. These parameters are associated with the minimum length of time a bomber flies low, and where a bomber is to change altitude in the neighborhood of a target.

Block 25: Post Launch Event (figure 162)

After initialization, the posting of events in the output arrays of common /DINDATA/ begins with the posting of the Launch event. It is posted by location (latitude and longitude) as well as by type and "place" index. Table 37 lists the types of events admissible for posting, as well as their names. Note that the GO HIGH, GO LOW, and DOGLEG events are not admissible in the final plan.

Block 26: Post Refuel Events (figure 163)

If there is a Refuel event, it is posted next. Refueling is accomplished in one of three ways: (1) at preassigned refuel areas (refuel index >0), (2) buddy refueling by another bomber or tanker launched from the same base (refuel index = -1 or -2), or (3) automatically by PLNTPLAN (refuel index = -4 for single refuel, -5 for two refuels). In the first case the data preparer assigns refuel areas for both bombers and tankers. In buddy refueling, tankers are ignored. Bombers are refueled by the buddy system at maximum range (a great-circle distance from the base equal to refueled range minus range) or just prior to the corridor origin, whichever is sooner.

When a bomber is to be assigned a refuel area by PLNTPLAN the buddy refuel point, X, is first computed a distance ΔR from the base on a great circle between it and the corridor entry, as for buddy refueling. See figure 155. ΔR is the difference between refueled range and range. If there already exist refuel areas which are within ΔR of the base and within some specified distance, D, of the point X, the area nearest X is assigned as the refuel point. Otherwise the point X is assigned and is added to the list of refuel areas.

The list of tanker bases is then scanned to see whether the buddy refuel point is within range of any of them. If not, the closest tanker base is chosen and a new refuel point is computed by interpolation. This point will lie on the line drawn between the tanker base and the original refuel point such that it will be within range of the tanker base.

The actual time of arrival at the refuel area is computed, using the CORBOMB parameter if the plan is for a first strike. The earliest arrival time in each refuel area is saved for later use when generating tanker plans (array ARTIME). Also saved for tanker scheduling is the arrival time and refuel area for each bomber (array ARVLS).

Block 27: Initialize Plan with Respect to GOLOW Range (figure 164)

The low-altitude range available to the bomber in flying the sortie is specified to PLNTPLAN in three separate amounts: the amount during the precorridor legs (G_1), the amount immediately prior to the first target (G_2), and finally, the amount immediately following the first target (G_3).

In block 27, these amounts are examined to make certain that the bomber does not fly low for less than 15 minutes. If $G_1 < 15 * SPDHI$, then G_1 is added to G_2. If $G_2 + G_3 < 15 * SPDHI$, then G_2 and G_3 are set to zero.

If the bomber is a tactical or naval aircraft (denoted by the use of corridor 1 or 2), coding blocks 30 and 31 are skipped.

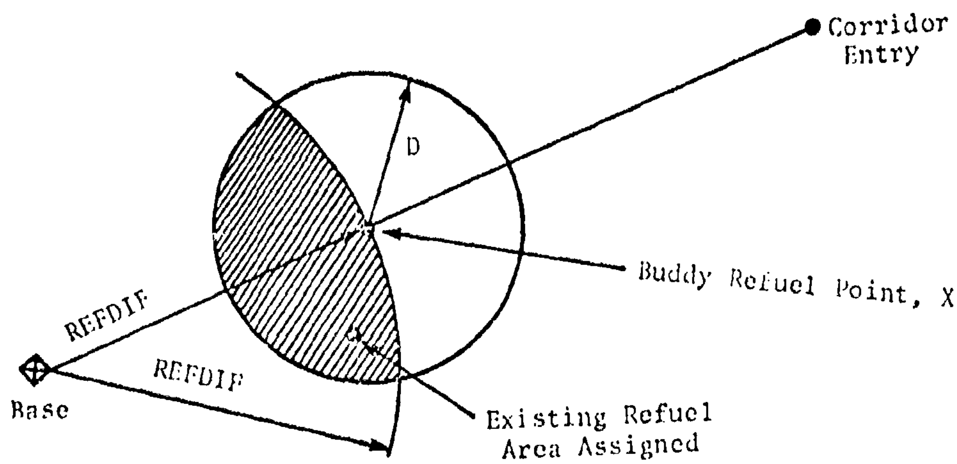


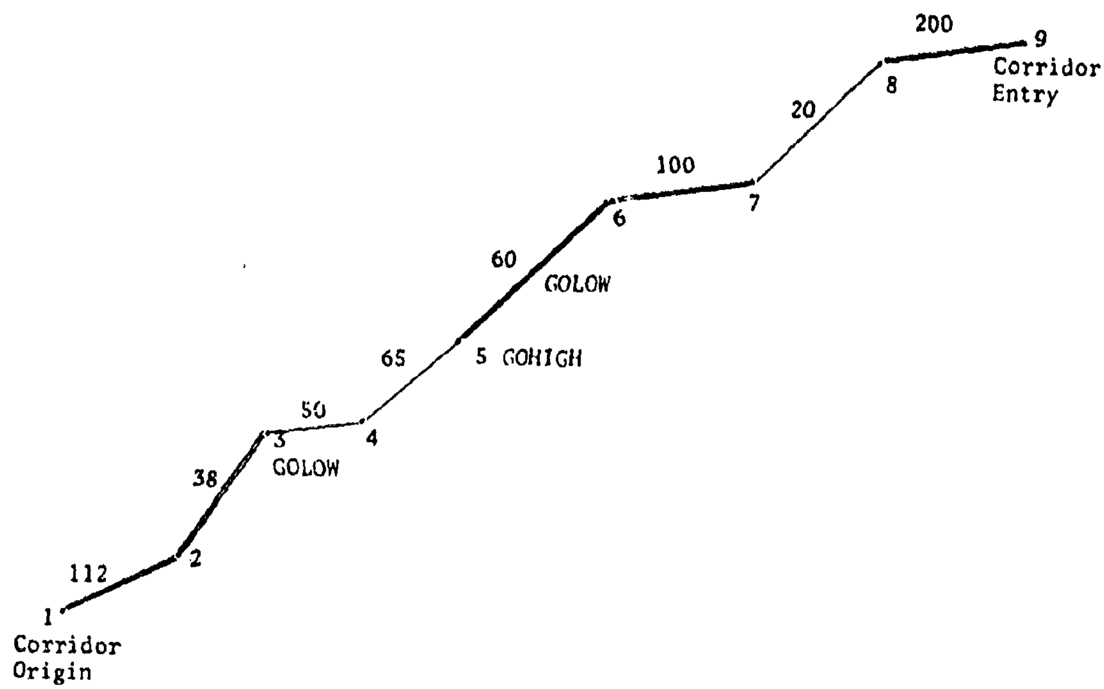
Fig. 155. Acceptable Locations for Refuel Area (Shaded Section)

Block 30: Process Precorridor Legs and Apply GOLOW-1 (figure 165)

The main sortie processing begins then at block 30 with the processing of the precorridor legs. They must be processed in the opposite direction from the bomber flight beginning at the origin and proceeding backward toward the entry. This is because the available low-altitude range (G_1) is measured backward from the corridor origin. Corridor attrition may be associated with the precorridor legs, and low-altitude range is applied against only those corridor sections where the bomber would experience attrition. Any G_1 remaining is added to G_2.

The processing for this block of coding is perhaps best described by referring first to figure 156 which gives an example of precorridor legs in the most complex configuration allowed. It also shows how this corridor is described to the program in /HAPPEN/ (contained in common /9/). The corridor consists of eight separate doglegs or nine points, and so is described in nine lines in /HAPPEN/. Those doglegs where the bomber would experience attrition are indicated by double lines. The corridor is described by listing the location (latitude and longitude) of each dogleg point in order beginning at the corridor origin and proceeding backward toward the corridor entry, as shown in the figure. With each point the distance from the previous point is also noted. If attrition begins at a point, this is noted by entering a 1, 2, or 3 in array JAPTYPE, depending upon whether this is the first, second, or third section. Similarly, if attrition ends at the point, the number 4, 5, or 6 is entered. Thus point 1 is labeled with a 1, and point 3 with a 4, to indicate the beginning and end of the first attrition section. This example, of course, describes an extreme situation where attrition occurs in three separate sections. Usually, there will be attrition in at most one section. The program must know which doglegs have attrition in order to know where to apply the low altitude range G_1. In the figure example, suppose G_1 = 180 miles, then 112 miles would be applied against the first dogleg back of the corridor origin, 38 miles applied to the second dogleg. The balance of 30 miles would be applied to the 5th dogleg beginning with point 5 and ending midway between points 5 and 6. As a result, GO LOW and GO HIGH events would be posted as indicated in the figure. The posting of a GO HIGH at the corridor origin depends on the value of G_2.

This section also sets up arrays which contain the event numbers of all those precorridor events which might possibly call for the launching of a decoy. These arrays are called LOHIMHT and LDMHT. The event number is stored in LOHIMHT for events of priority 2. (See subroutine DECOYADD for table of priorities.) The event number is stored in LDMHT for a launch of priority 3 (or, after the first such event,



(a) Corridor Example

POINT NUMBER	JAPTYPE	HAPLAT	HAPLONG	HAPDIST
1	1	lat_1	$long_1$	0
2	0	lat_2	$long_2$	112
3	4	.	.	38
4	0	.	.	50
5	2	.	.	65
6	5	.	.	60
7	0	.	.	100
8	3	.	.	20
9	6	.	.	200

(b) Content of Common /HAPPEN/

Fig. 156. Example of Precorridor Legs

priority 5). For the priority 5 launches, the distance to be covered by the decoy is accumulated in a corresponding word of array DELDIS.

The flag JA is set to 1 when the beginning of an attrition section is encountered, and back to 0 at the end. IDEL is the indicator to be compared against JAPTYPE in order to determine the beginning and end of attrition events. The counter JDO is advanced each time an additional Change Altitude event is added.

Block 31: Post Corridor Events (figure 166)

After G_1 has been measured out and the necessary Change Altitude events determined, the precorridor legs are examined for possible zone crossings. They are again processed backward beginning at the corridor origin whose zone residence is known. A check is made for zone crossings as far back as the corridor entry. If the entry point is located inside a defense zone, the bomber is considered to enter that zone at that point.

In block 31, zone crossing events are integrated into the event list. The displacement they induce in the event numbers must be reflected in a shifting of the Decoy Launch event numbers. This is the second pass through the event list, and the list still is being processed backwards from origin to entry. No check is made prior to the corridor entry. When a zone crossing is detected, an INSECTOR event is inserted into the event list along with the latitude and longitude of the crossing. The detection of the crossing and computation of the crossing position are done via a call on subroutine BOUNDARY. The indices in the two arrays LDMHT and LOHIMHT are updated to reflect the displacement of event numbers by the insertion of the INSECTOR event. Finally, the numbering of the events is reversed to put them into a sequence corresponding to the entry-to-origin direction of the sortie, and the event numbers in LDMHT and LOHIMHT are changed to reflect the reversal. The possible Decoy Launches then are posted by filling array LMHT with the event number (from array LDMHT or LOHIMHT) and by filling array LPRIORITY with the associated priority (5 if the event number is from array LDMHT, 2 if the event number is from array LOHIMHT). For the priority 5 launch, the index to the associated distance in array DELDIS is the same as the index to the event number in array LDMHT; hence, this index is stored in array NDCYRQ. The priority 2 launch does not require a coverage distance. Whenever this is the case, a 1 is stored in array NDCYRQ. The actual filling of these arrays is done by subroutine POSTLAUN.

Block 40: Adjust /OUTSRT/ for ASM Events (figure 167)

The list of input events in the /OUTSRT/ arrays is next examined for ASM events. If there are any, the aim or launch points for them are now calculated. If necessary, the ASM events are reordered among the list of other events at this time. This is because ASM target events are supplied to PLNTPLAN by POSTALOC without aim points, and approximately in their proper order. They may appear later in the lists than they should, but not earlier. Figure 157 shows a list of happenings with ASM events, to illustrate what is meant. The list indicates a DROPBOMB event at point 2 followed by ASM events at points 3 and 4, then a DROPBOMB event at point 5 and an ASM event at points 6 and 7. Suppose that the ASMs were to be launched or aimed as shown in figure 157, that is, at points 1, 9, 10, and 11. Then the list of happenings would be rearranged as shown. If an ASM point fell before the origin, the origin would be used as the aim point. The aim points 9, 10, and 11 would be computed using the LAUNCH subroutine.

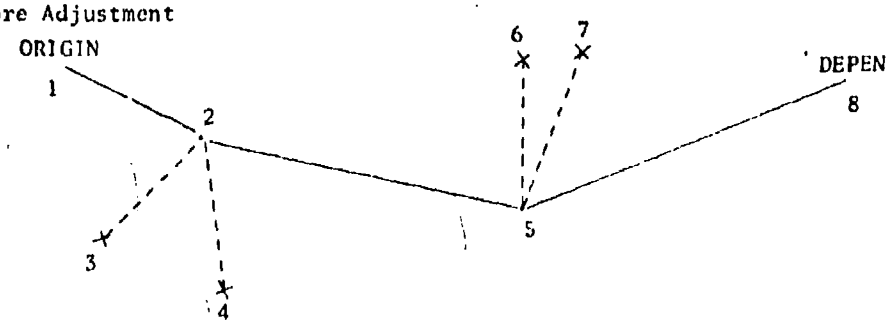
The processing for block 40 is carried out in four separate steps, utilizing the arrays from common /ASMARRAY/. (1) The ASM targets are first examined to see if they are in range of the origin or some other prior fly point such as a drop bomb point. Those which are not are flagged by setting the corresponding cell of array IFLY to 1. In the example in figure 157, points 3 and 4 would not be flagged, but points 6 and 7 would. (2) The ASM targets flagged in the last step would again be examined in the second pass to see if any were in range of a previous ASM fly point. In the example shown, point 7 would be in range of ASM target 6. The aim point (No. 10) for ASM target 6 is computed at this step. The bomber's path is now fully determined. (3) Sort indices are now generated for all events. For all fly points, the point number is taken as the sort index. For all other points, (i.e., all ASM points which are not fly points; in this example, points 3, 4, and 7), the sort index is a number whose integer part is the earliest point just out of range, and its fractional part is the distance to this point. After the sort indices are generated, the list is appropriately rearranged using the ORDER and REORDER functions. (4) The aim points for the rest of the ASMs are calculated.

Subroutine FLYPOINT, which has the entries PREFLY1, PREFLY2, and POSTFLY, is logically an integral part of this coding block. Subroutine LAUNCH is called only by POSTFLY.

If the last event is an ASM event, the depenetration corridor is reselected.

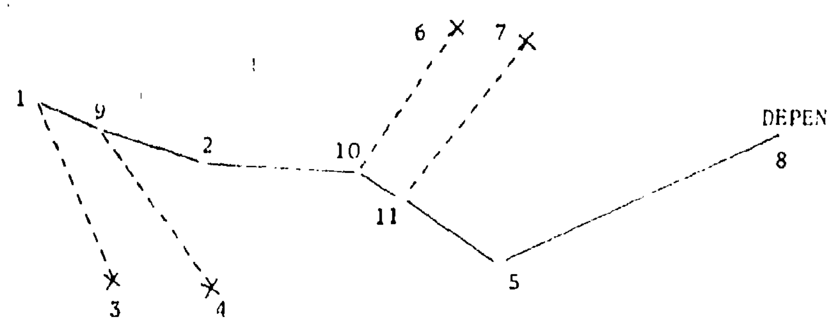
The locations of appropriate Change Altitude events associated with the ranges G_2 and G_3 are now calculated by subroutine ADJUST. If the target area was found to be degenerate, blocks 50 and 60 are skipped.

(a) Before Adjustment



Point	IBTYPE
1	DOGLEG
2	DROPBOMB
3	AIMASM
4	AIMASM
5	DROPBOMB
6	AIMASM
7	AIMASM
8	DEPEN

(b) After Adjustment



Point	IBTYPE
1	DOGLEG
3	AIMASM
4	AIMASM
2	DROPBOMB
6	AIMASM
7	AIMASM
5	DROPBOMB
8	DEPEN

Fig. 157. Illustration of ASM Event Adjustment

Block 50: Apply GOLOW-2 Before First Target (figure 168)

Block 50 posts all events in the target area, including ASM launches from the corridor origin, altitude changes and zone crossings. All events except ASM launches are entered in the detailed History table as one-line events. For ASM launches two lines are required, the first for information pertaining to the launch and the second for information pertaining to the target. As the event list is processed, all possible Decoy Launch situations are flagged by storing the appropriate information pertaining to the launch and the second for information pertaining to the target. As the event list is processed, all possible Decoy Launch situations are flagged by storing the appropriate information in arrays LMIT, LPRIORITY, and NDCYRQ (see block 30).

Processing is terminated with the occurrence of the input events DEPEN, LAND, or DIVEMISL. For DEPEN, a normal exit is made to block 60 described in the next section. For LAND, an abort event is posted five minutes after the last target while the bomber is flying toward the depenetration corridor, and block 60 is skipped. For DIVEMISL, an abort event is posted immediately at the last target.

Block 60: Post Depenetration Events (figure 169)

This block of coding completes the processing for a normal sortie, processing from the last target to the recovery point or base. It computes the most distant recovery base, associated with this depenetration point, that the bomber can reach. The information on this base and the depenetration corridor index are recorded in the depenetration event. In addition, the time of flight to each of the possible recovery bases is computed and stored. These calculations follow the processing of the depenetration leg events. Once again, a check is made for zone crossings. If a zero zone is encountered in zone processing, indicating that the bomber has left the area in which there are defense zones, the zone crossing check is turned off and no further check for zone crossings is made. Thus if a sortie should leave an area of defense zones, and later reenter another area of defense zones, this second area will be ignored. When a crossing into a zero zone is posted, the value of SIDE is posted in lieu of zero.

Block 80: Read Next /OUTSRT/ Record, Convert Last One (figure 170)

After the processing of the sortie has been completed as described above, the next /OUTSRT/ record is read in and checked to determine if it contains the alternate plan for the sortie just processed. The details of the processing at this point are determined by whether the current sortie is the primary plan, its refuel-abort alternate, or whether it is a plan without an alternate. In any case, subroutine SWTCHALT is called to convert the CHANGALT events to GO LOW or GO HIGH events. Subroutine DISTIME then is called to compute distances between events and associated time increments, and subroutine DECOYADD is called to allocate the available decoys. Decoy Launches are now added to the detailed History table by examining each event to see if a launch is to be inserted (indicated if the corresponding word in array ILAUNDEC is nonzero). For low-altitude launches (ILAUNDEC = 0), the actual launch point must be computed. The Decoy Launches are inserted by copying each event into a temporary detailed History table. If a GO HIGH event has a decoy launch indicated, the launch is inserted after the GO HIGH. For all other events with indicated decoy launches, the launch is inserted before the event is copied. Decoy launches are posted by adding the event LAUNDCOY to the event array (JTP) and storing the number of decoys launched (>0) in the array usually reserved for the place index (KPL). The remaining information required in the detailed History table is stored in the normal manner.

Decoys are terminated as the detailed History table is recopied into its original arrays. Each time a high-altitude Decoy Launch event is encountered, the total decoy flight time is computed from the distance in array DISTORE (filled by subroutine DECOYADD) and added to the next odd word in an array (TSTORE) which holds the remaining flight time of all decoys which have been launched but not yet terminated at the time of this event. The number of decoys to be terminated is added to the next even word of TSTORE. As each subsequent event is processed, the time since the last event (HDT) is subtracted from the times in TSTORE. Whenever a decoy has no flight time remaining, a LAUNDCOY event, together with the number of decoys being terminated (stored as a negative number) and other relevant information, is added to the detailed History table. If the bomber depenetrates or aborts while decoys are still flying, the remaining decoys are terminated immediately before the final event. It should be noted that decoys launched at low altitude are not terminated.

If the recently read /OUTSRT/ record was an alternate, control transfers back to block 15. Otherwise PLNTPLAN outputs the final plan to the PLANTAPE if the PLANTAPE option has been exercised. If the EVENTAPE has been requested, control then transfers to coding block 90; else it returns to block 15.

Block 90: Process Final Plan and Write on EVENTAPE (figure 171)

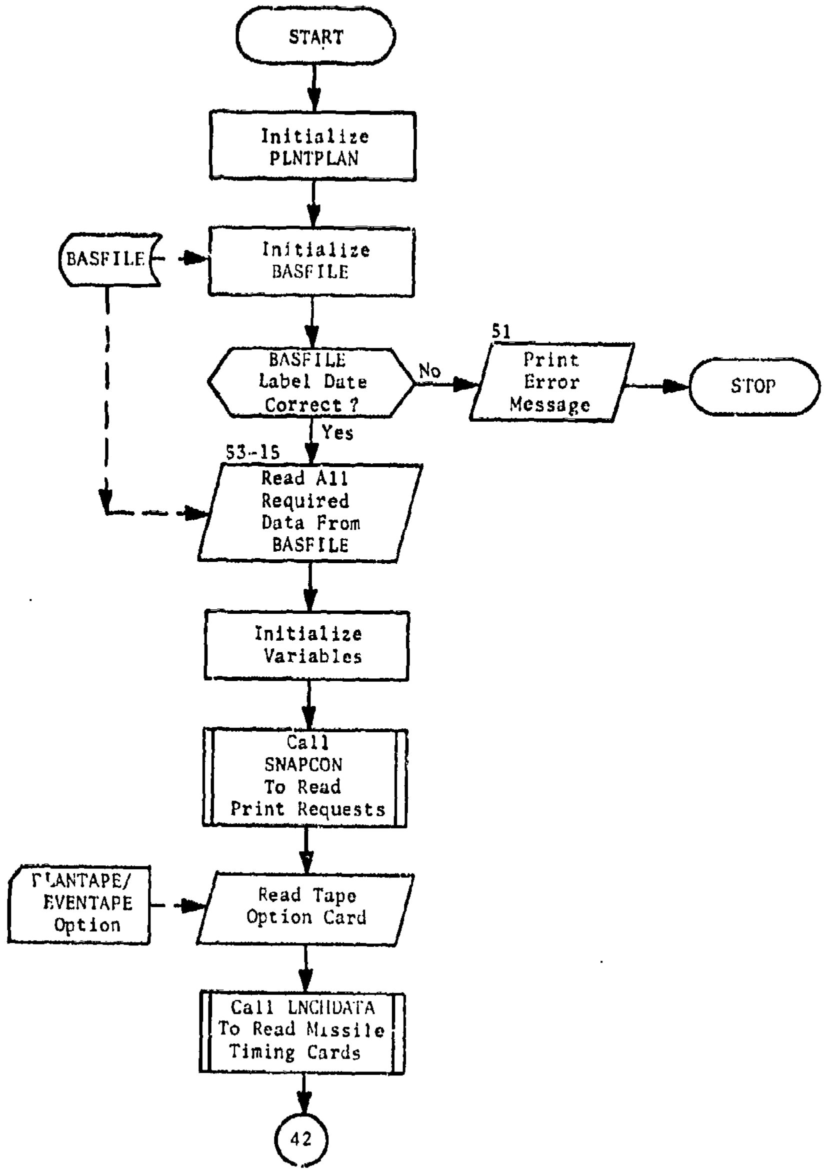
The purpose of this block is to format the plan for output to the Simulator. During this processing, all references to geography (i.e., latitudes and longitudes) are dropped. Dogleg events, which are strictly geographic, are also dropped and time increments associated with them are accumulated with the time increment of the next succeeding event. GO HIGH and GO LOW events are converted back to CHANGALT events and the Weapon table is constructed. (In the event that the plan and its alternate exceed 80 lines, a warning is issued on the standard output unit, and the first 80 lines of this plan are used.)

In the case of weapons which have a time-dependent destruction-before-launch probability (DBL), an extra event is written on the EVENTAPE. This event causes the Simulator to compute a dynamic destruct event for the base using the DBL data tables passed on the SIMTAPE by Program INDEXER. format for this event was shown in table 42 (under OUTPUT). If there are several bombers launched consecutively from the same base, this event precedes only the first bomber Launch event. If two of more of these events for the same base do appear on the EVENTAPE, the Simulator will process only the first.

Block 100: Program Termination (figure 172)

The termination of PLNTPLAN occurs when the end sentinel record is reached on the input STRKFILE. This record is identified by a group number greater than 200. First, subroutine PLANTANK is called to generate and write sorties for all tankers listed on the BASFILE. A sentinel record is written at the end of the EVENTAPE and the PLANTAPE; the bomber recovery information is added to the EVENTAPE and the Refuel Area table to the PLANTAPE. Finally, all files are terminated, and final messages are printed.

This completes the PLNTPLAN processing.



Block 10: Program Initialization

Fig. 158. Program PLTNPLAN
Block 10: Program Initialization
(Sheet 1 of 2)

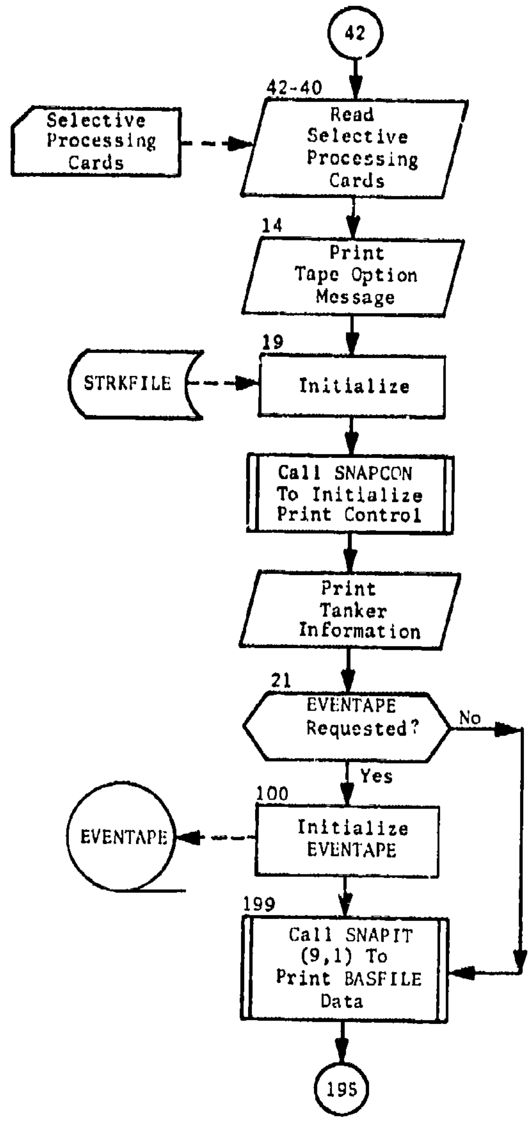


Fig. 158. (cont.)
(Sheet 2 of 2)

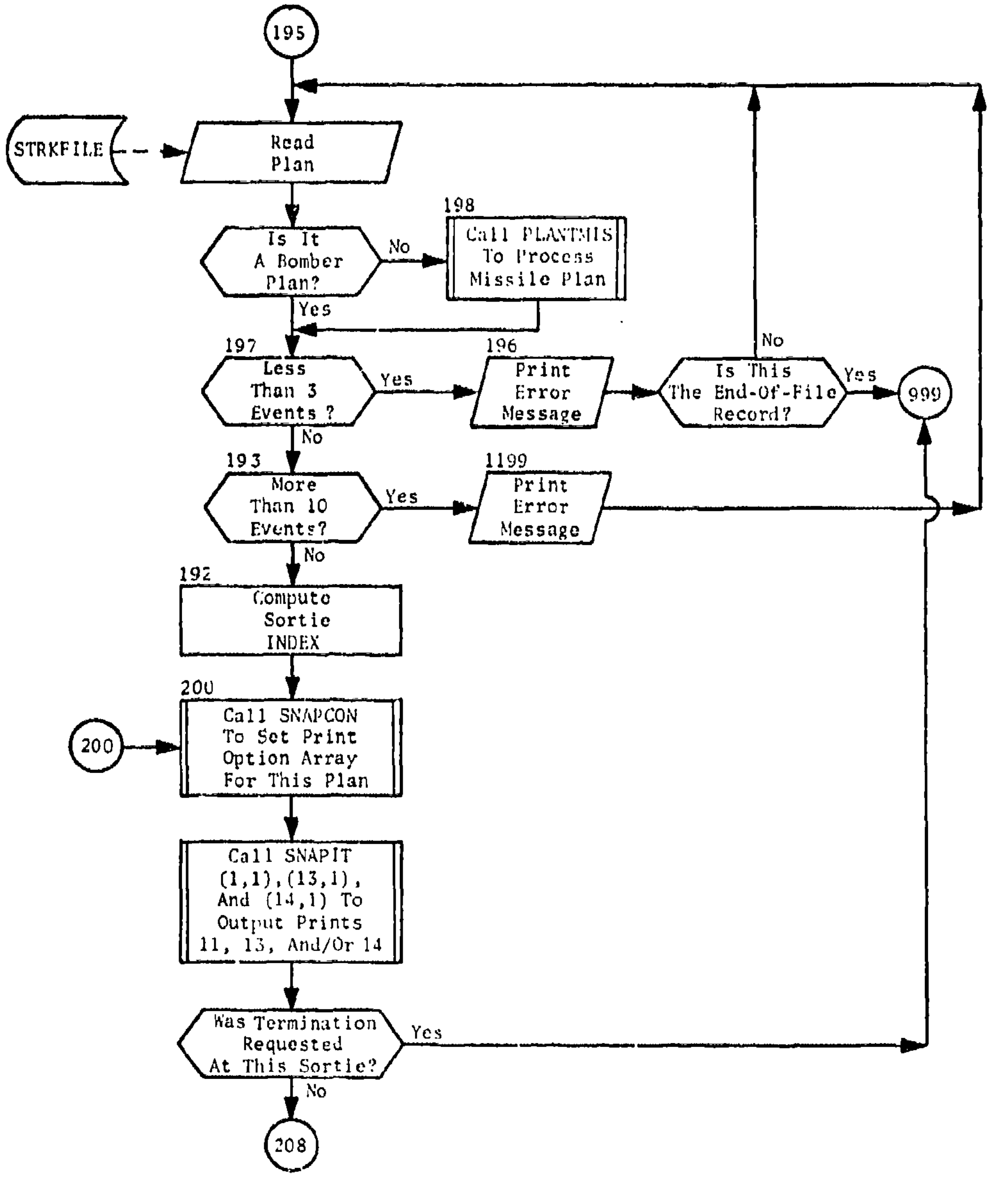


Fig. 159. Program PLNTPLAN
Block 15: Control Loop

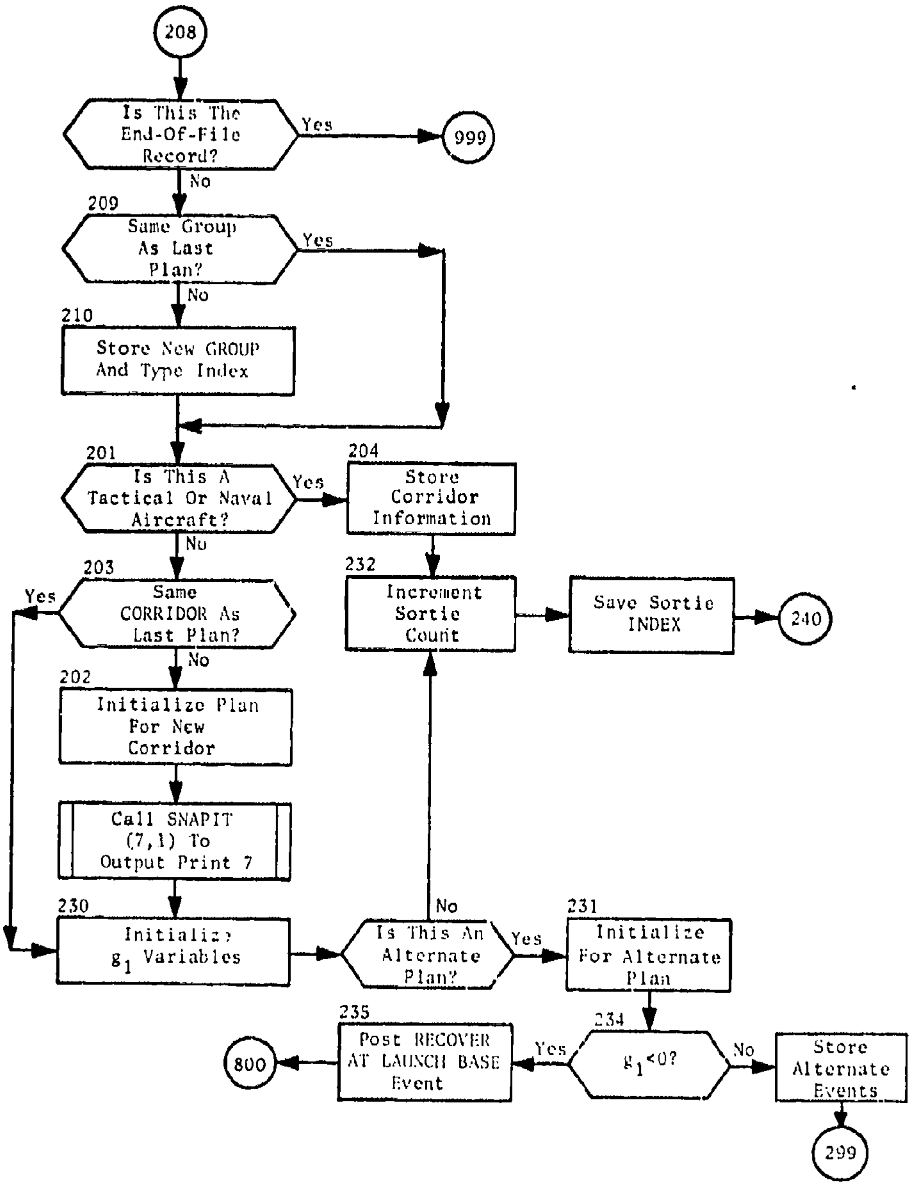


Fig. 160. Program PLTNPLAN
Block 20: Determine Type of Plan

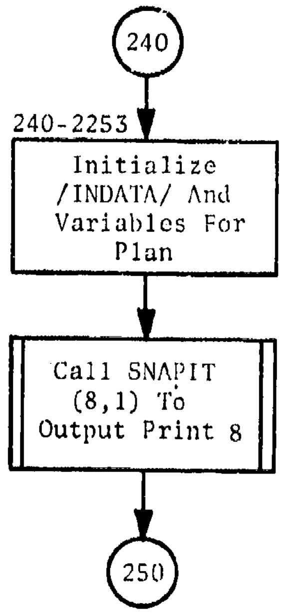


Fig. 161. Program PLNTPLAN
Block 24: Initialize Plan

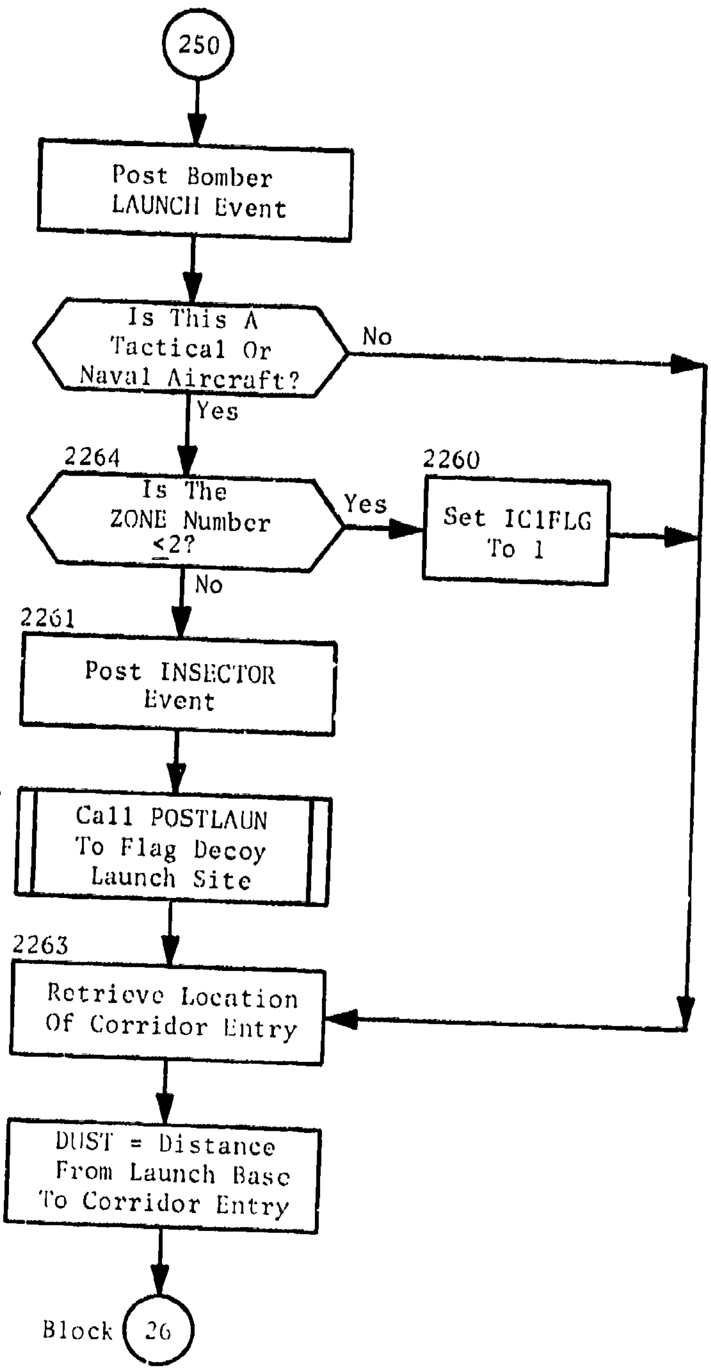


Fig. 162. Program PLNTPLAN
Block 25: Post Launch Event

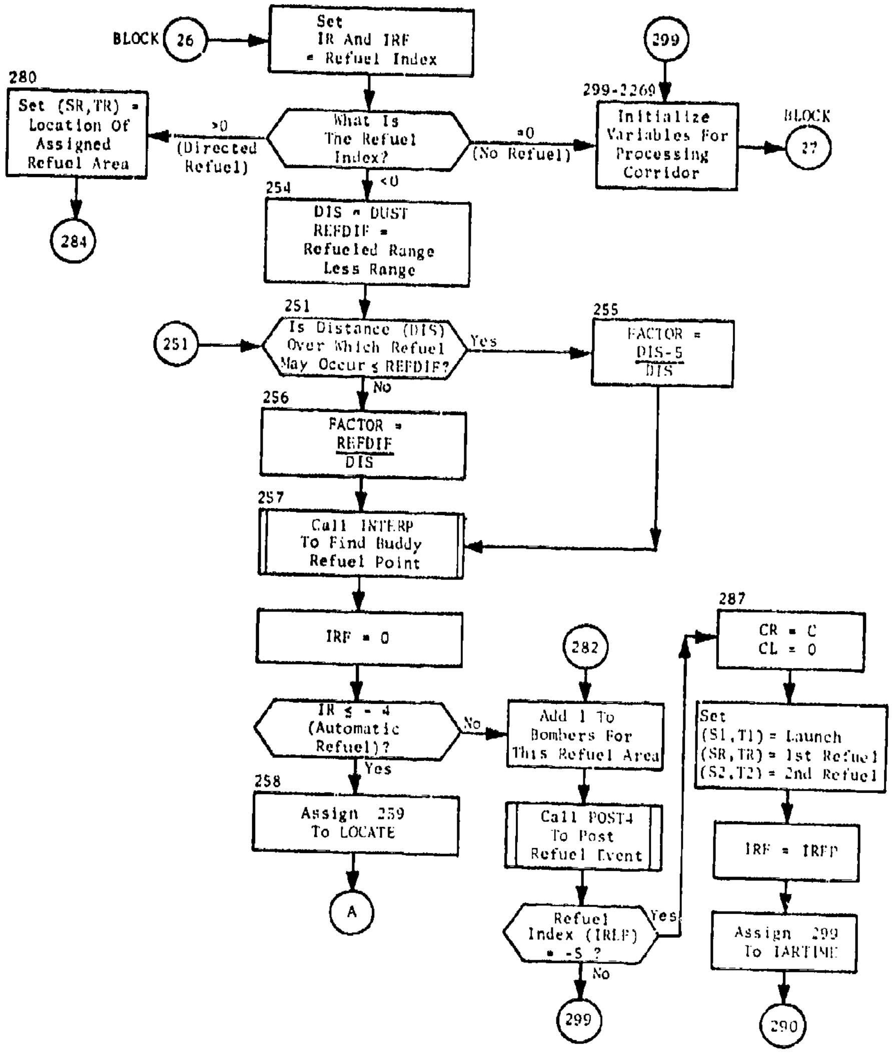


Fig. 163. Program PLNTPLAN
Block 26: Post Refuel Events
(Sheet 1 of 5)

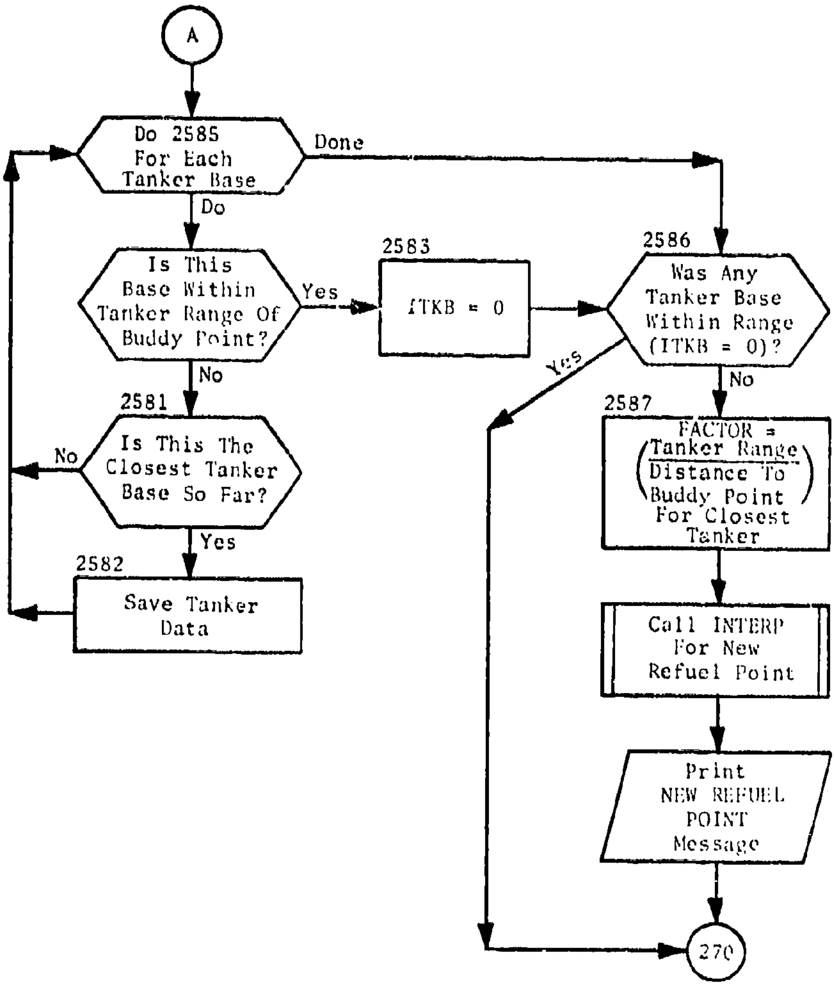


Fig. 163. (cont.)
(Sheet 2 of 5)

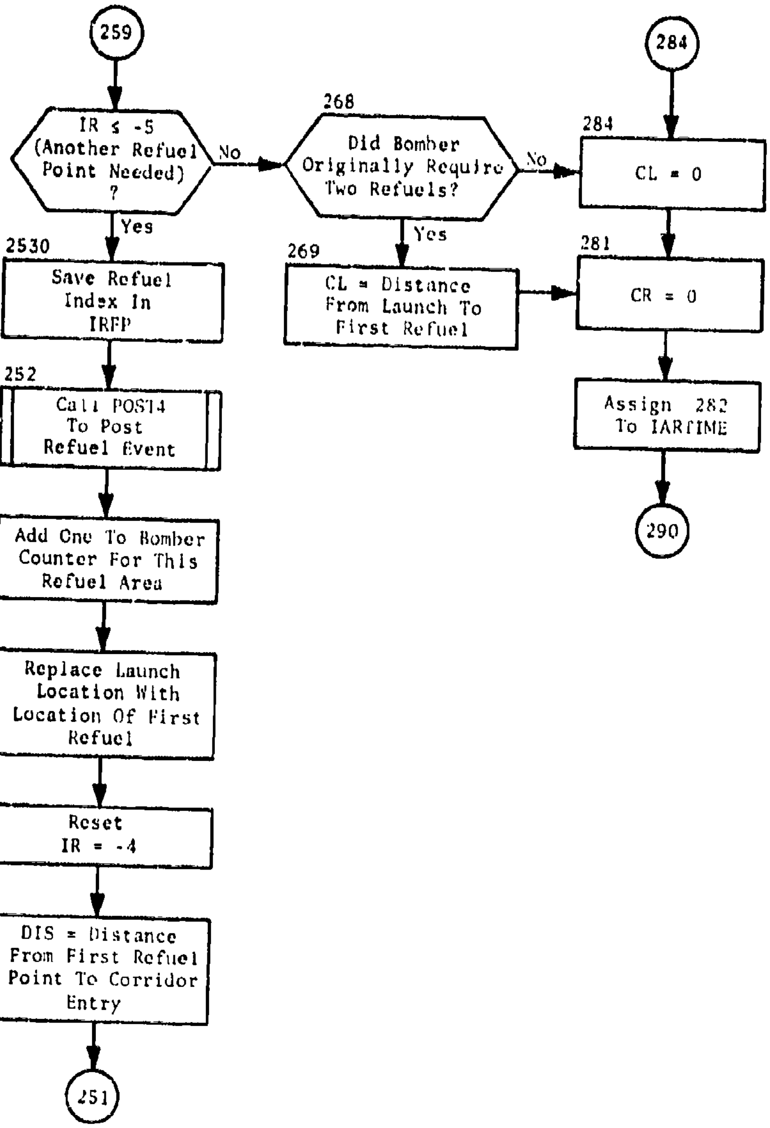


Fig. 163. (cont.)
(Sheet 3 of 5)

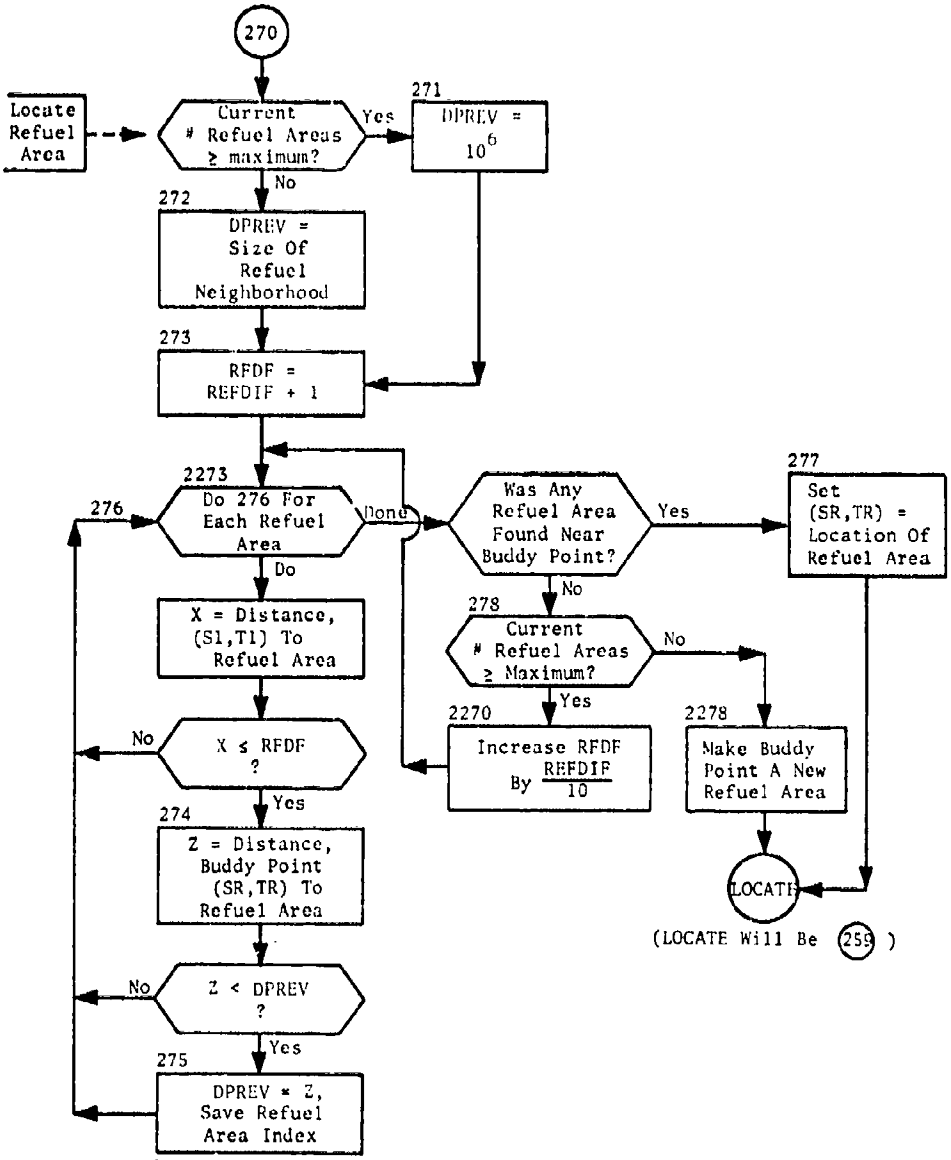


Fig. 163. (cont.)
(Sheet 4 of 5)

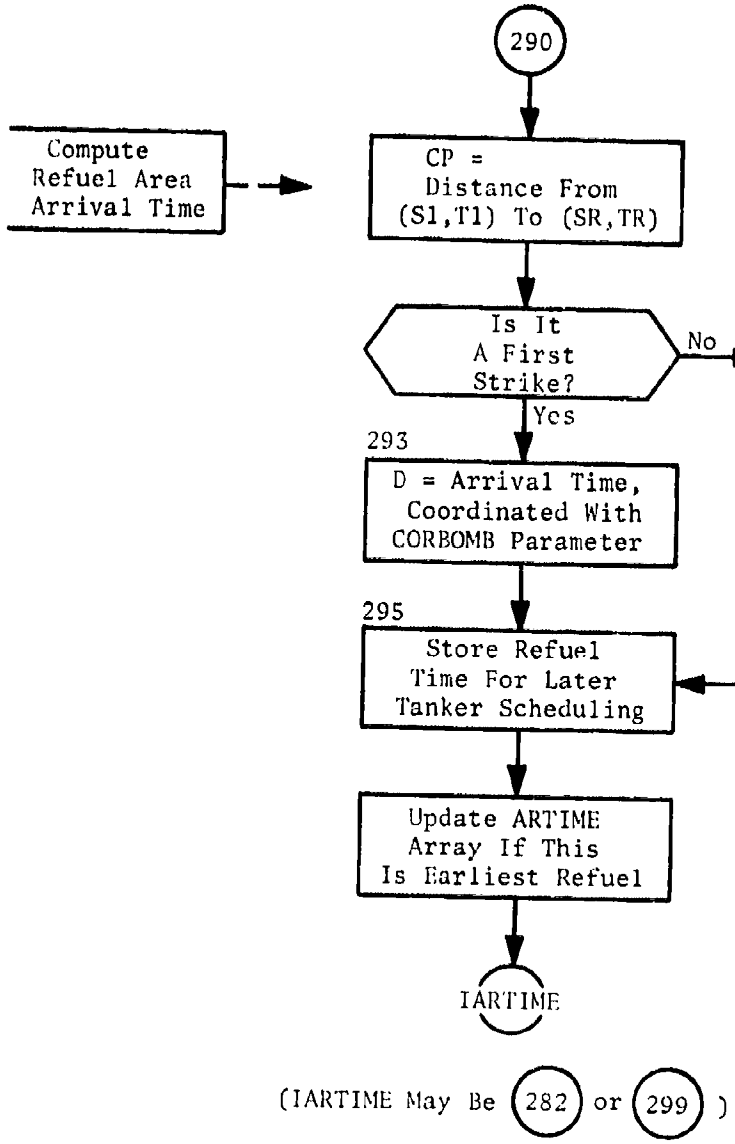


Fig. 163. (cont.)
(Sheet 5 of 5)

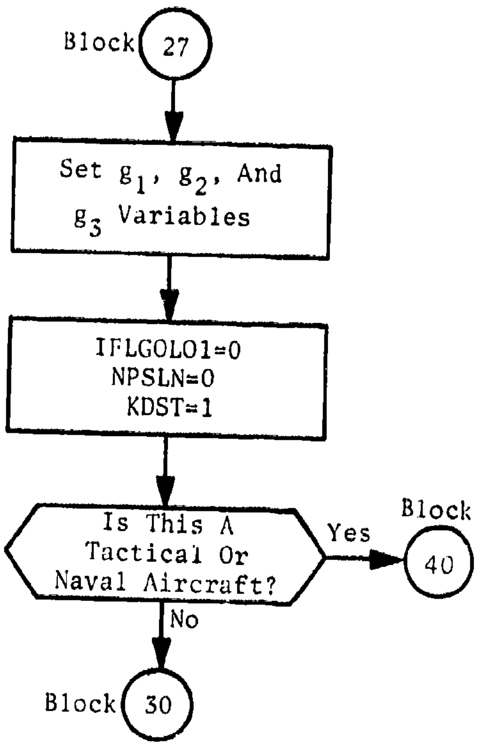


Fig. 164. Program PLNTPLAN
Block 27: Initialize Plan
With Respect to GOLOW Range

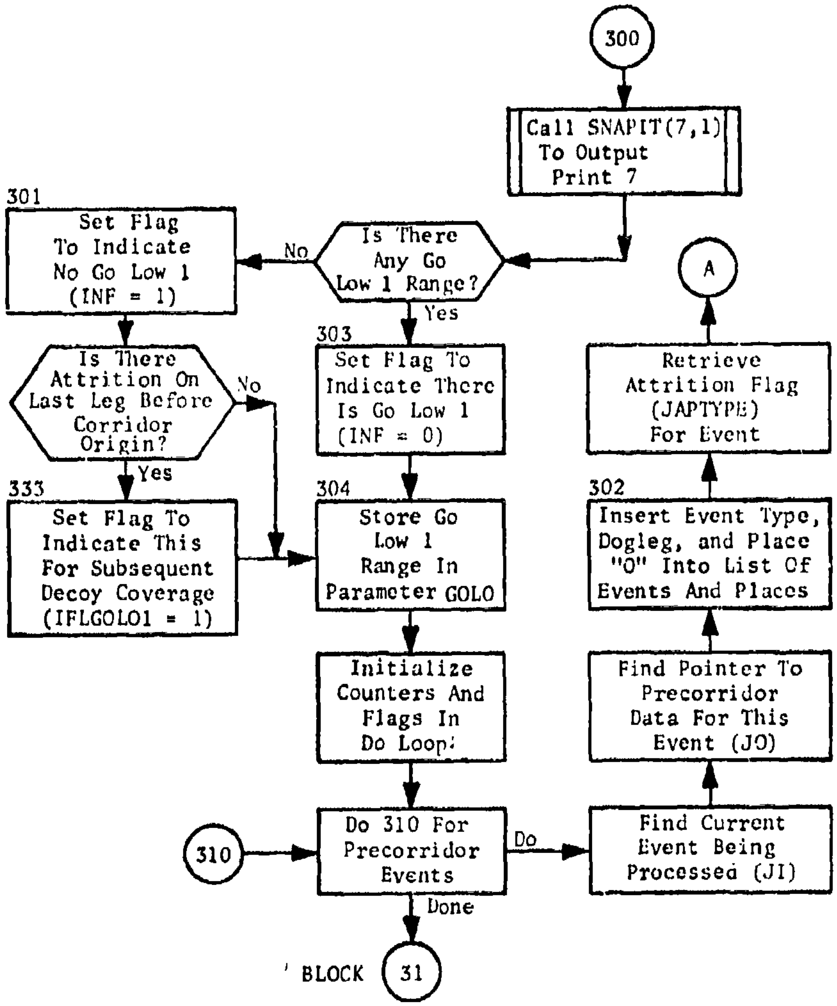


Fig. 165. Program PLNTPLAN
Block 30: Process Precorridor Legs
And Apply GOLOW1
(Sheet 1 of 3)

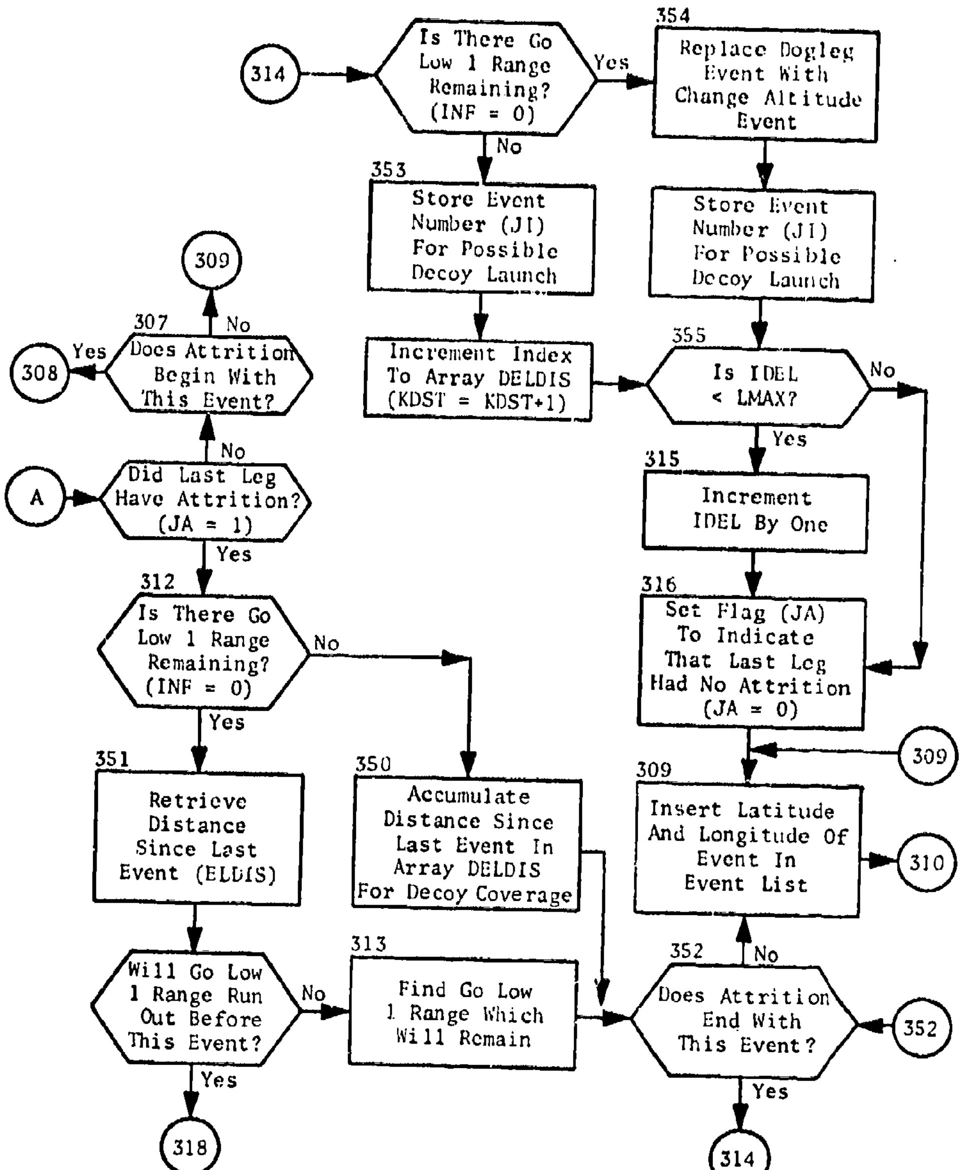


Fig. 165. (cont.)
(Sheet 2 of 3)

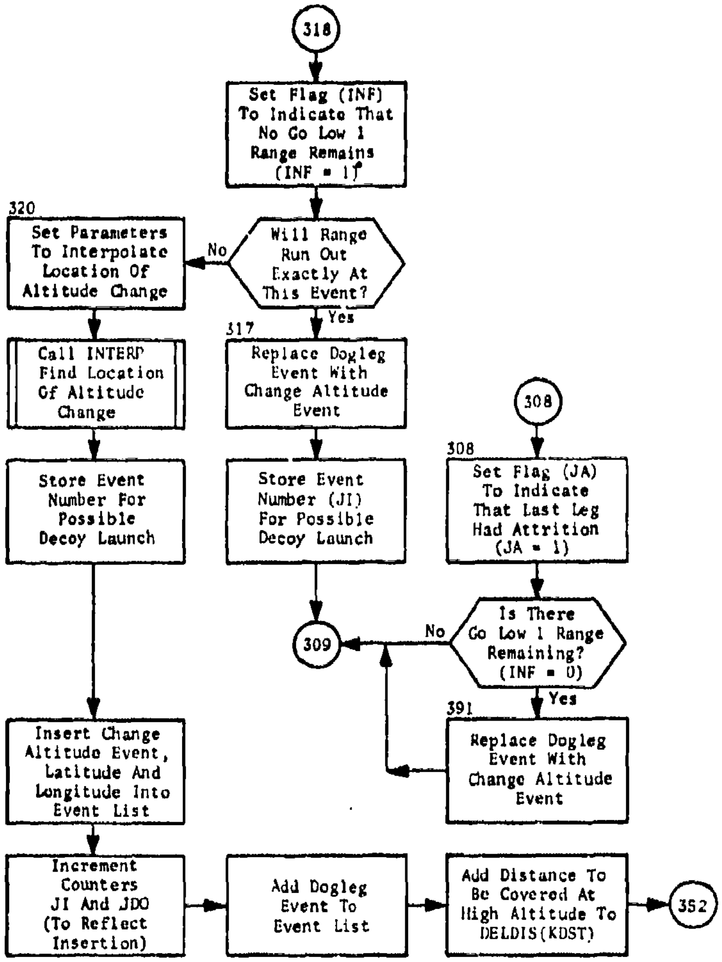


Fig. 165. (cont.)
(Sheet 3 of 3)

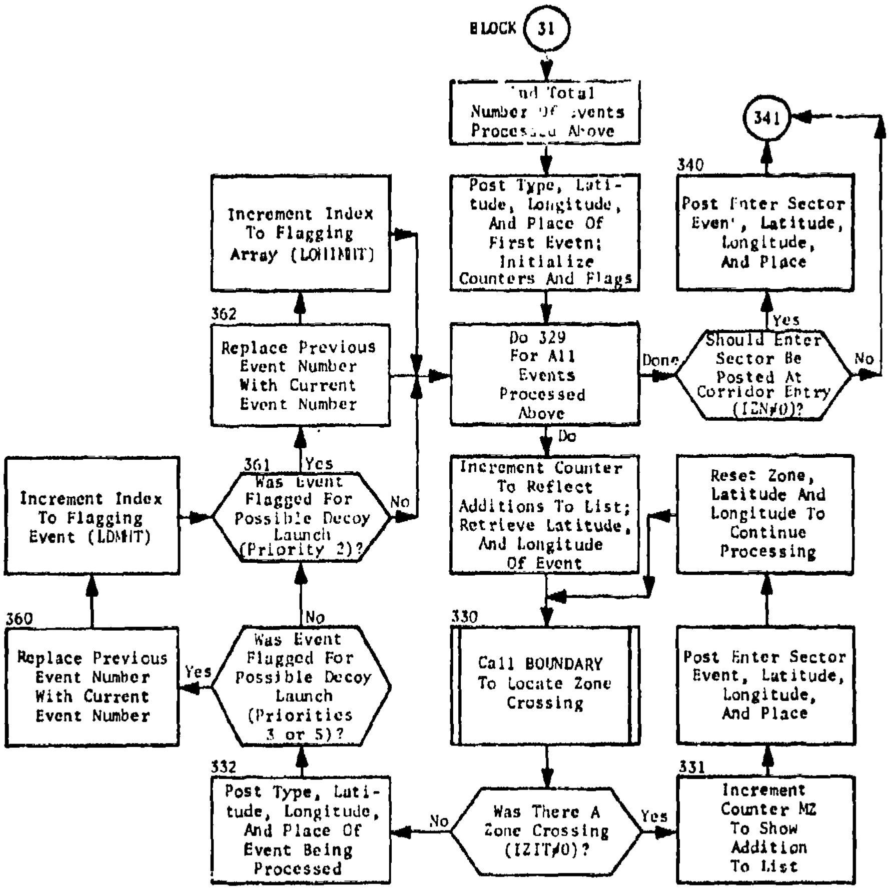


Fig. 166. Program PLNTPLAN
Block 31: Post Corridor Events
(Sheet 1 of 4)

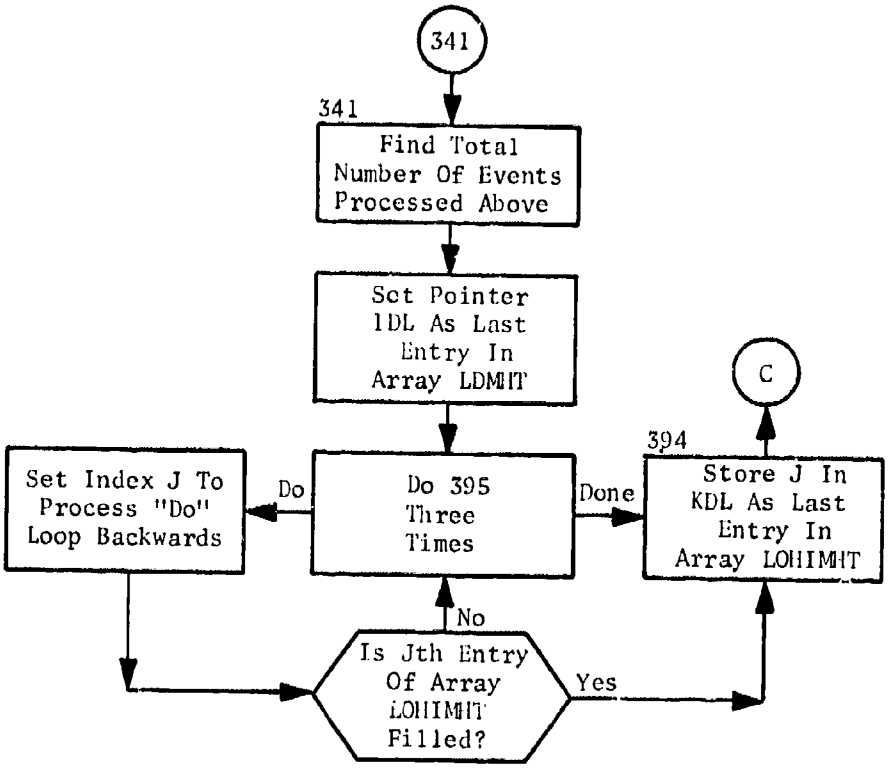


Fig. 166 (cont.)
(Sheet 2 of 4)

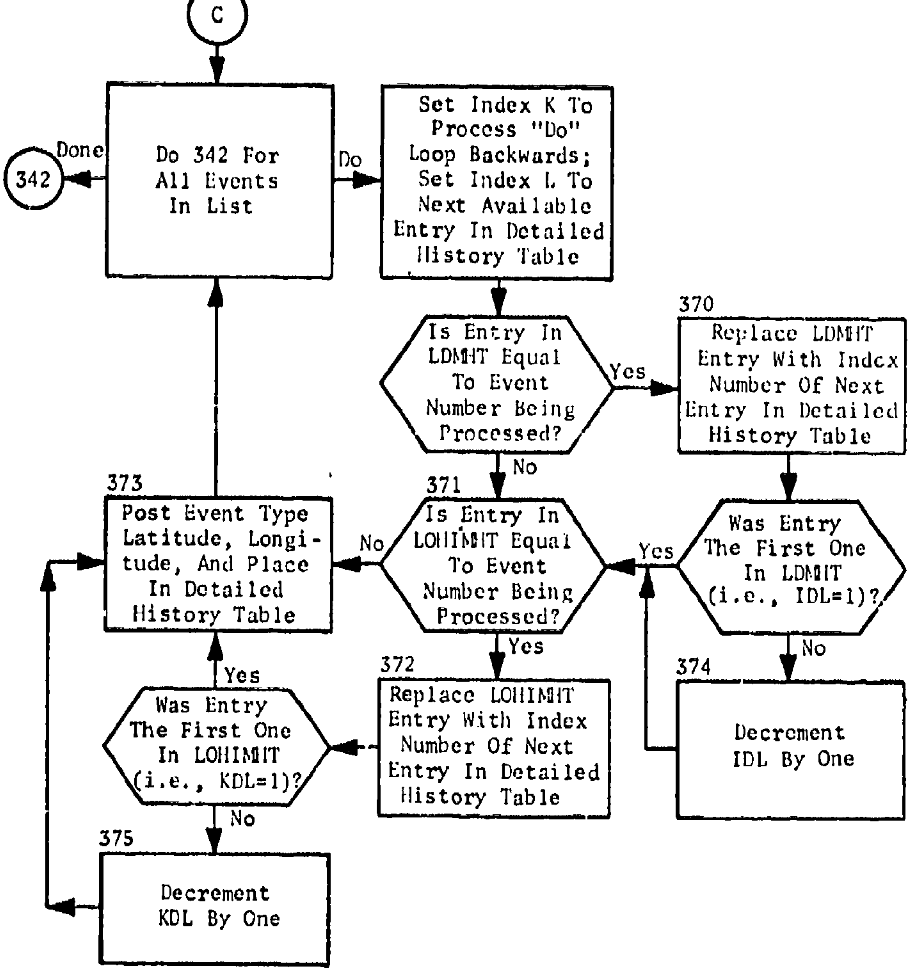


Fig. 166. (cont.)
(Sheet 3 of 4)

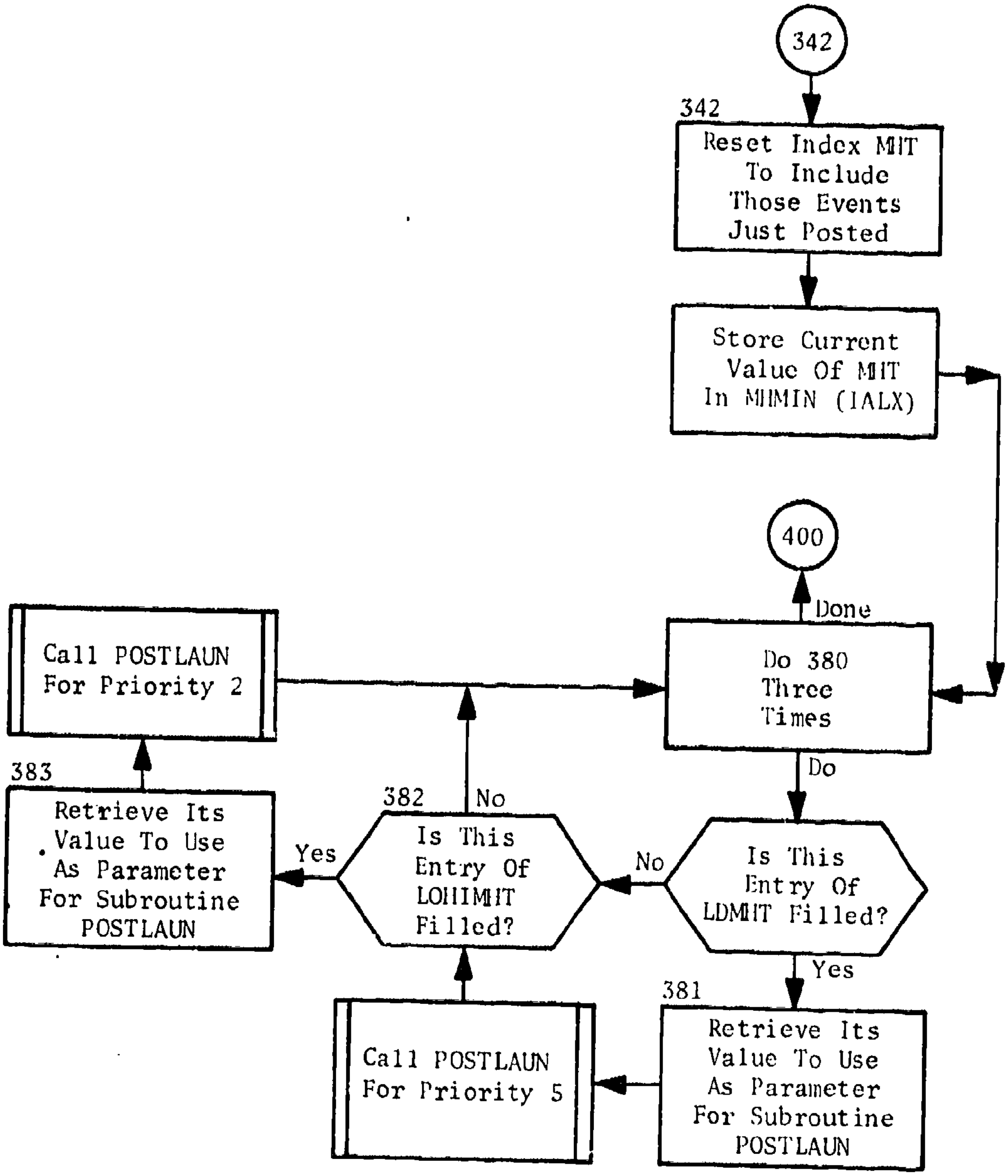


Fig. 166. (cont.)
(Sheet 4 of 4)

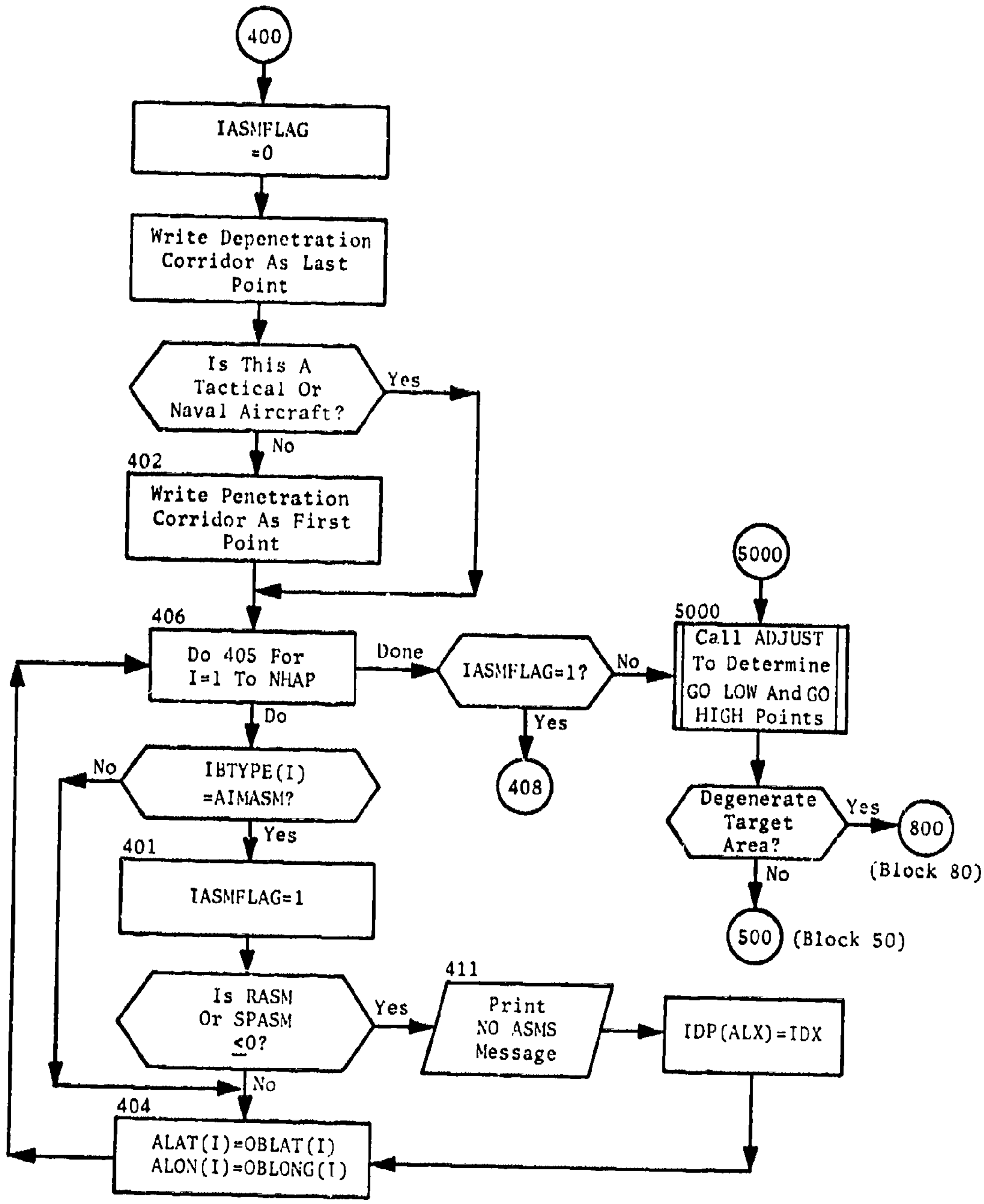


Fig. 167. Program PLNTPLAN
Block 40: Adjust /OUTSRT/
for ASM Events
(Sheet 1 of 4)

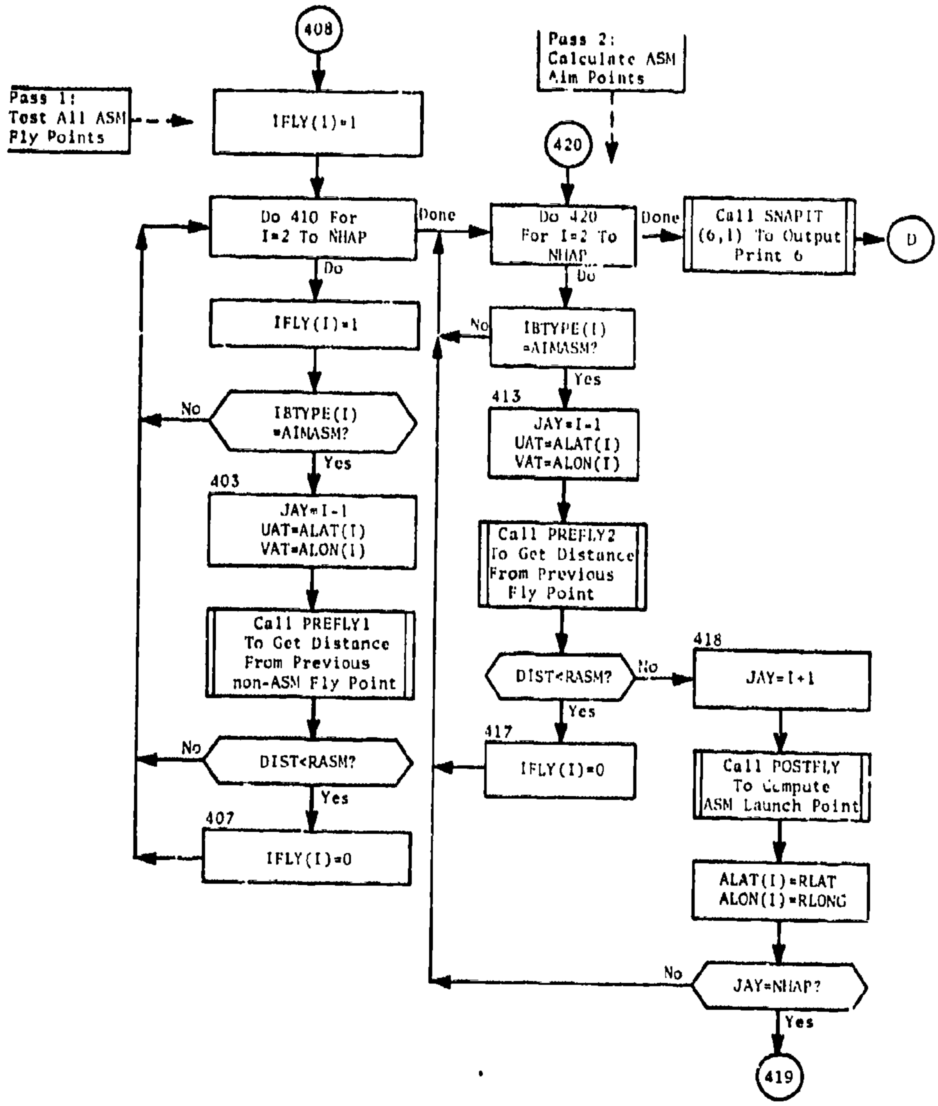


Fig. 167. (cont.)
(Sheet 2 of 4)

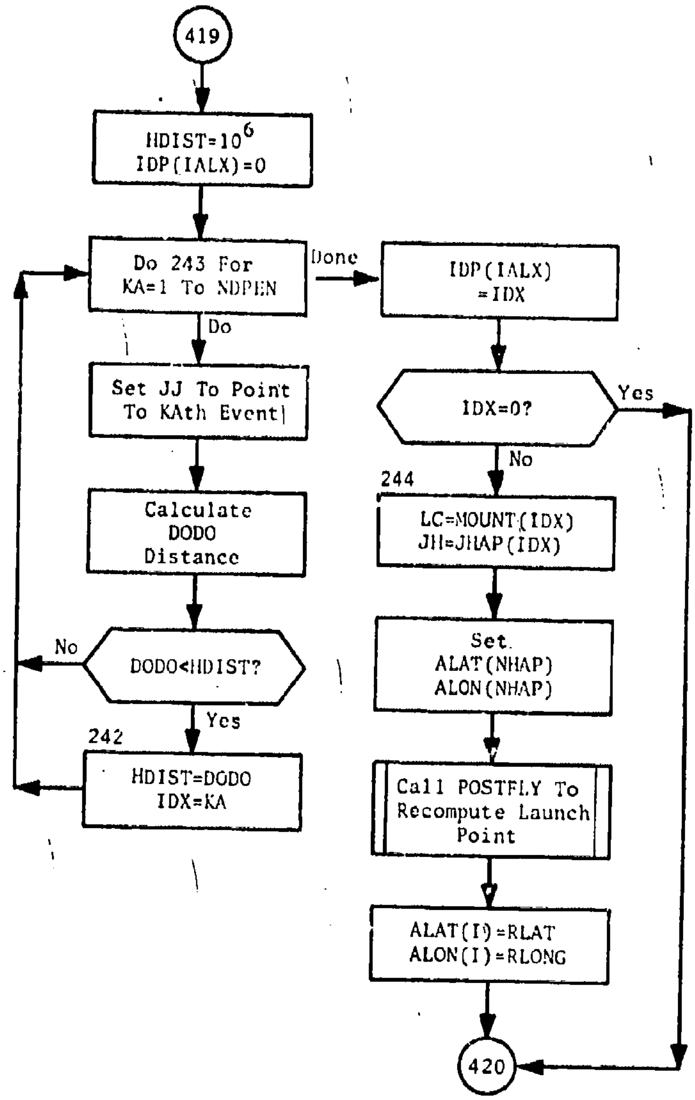


Fig. 167. (cont.)
(Sheet 3 of 4)

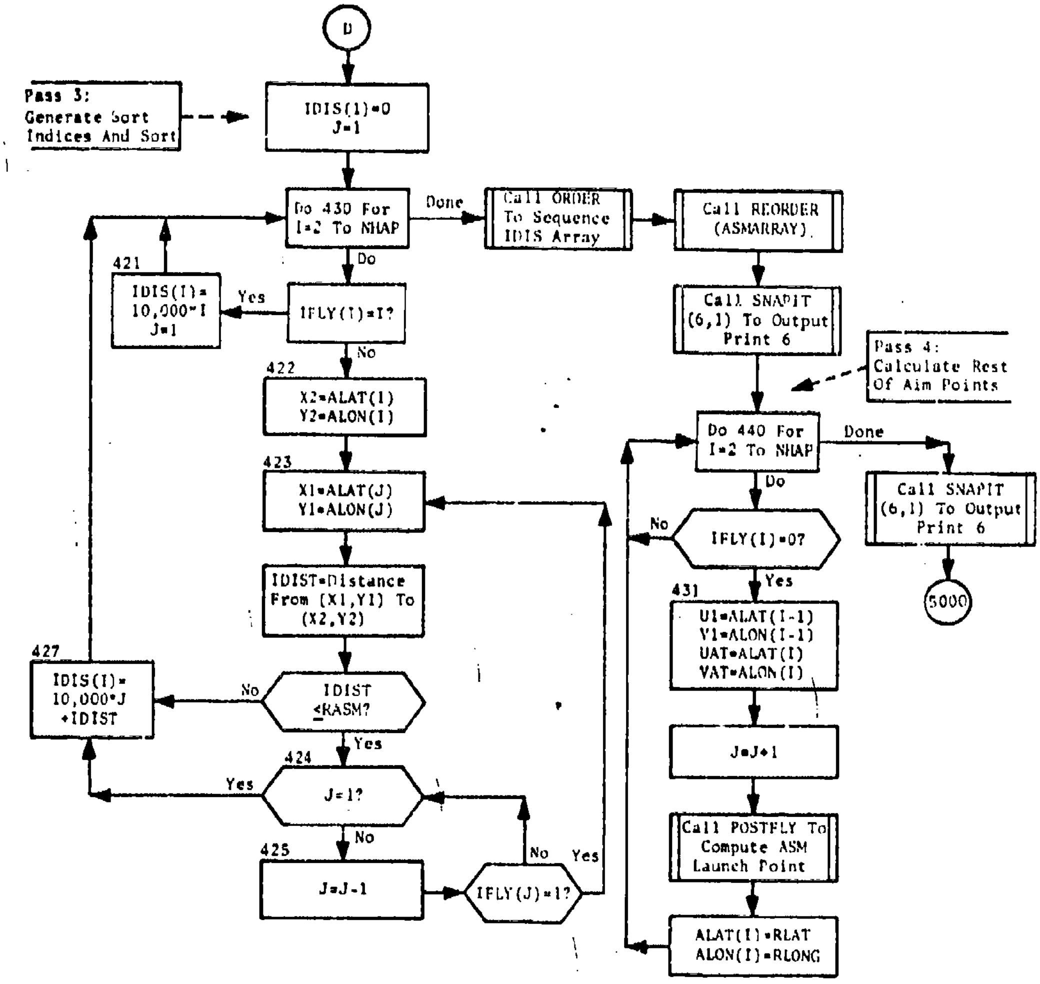


Fig. 167. (cont.)
(Sheet 4 of 4)

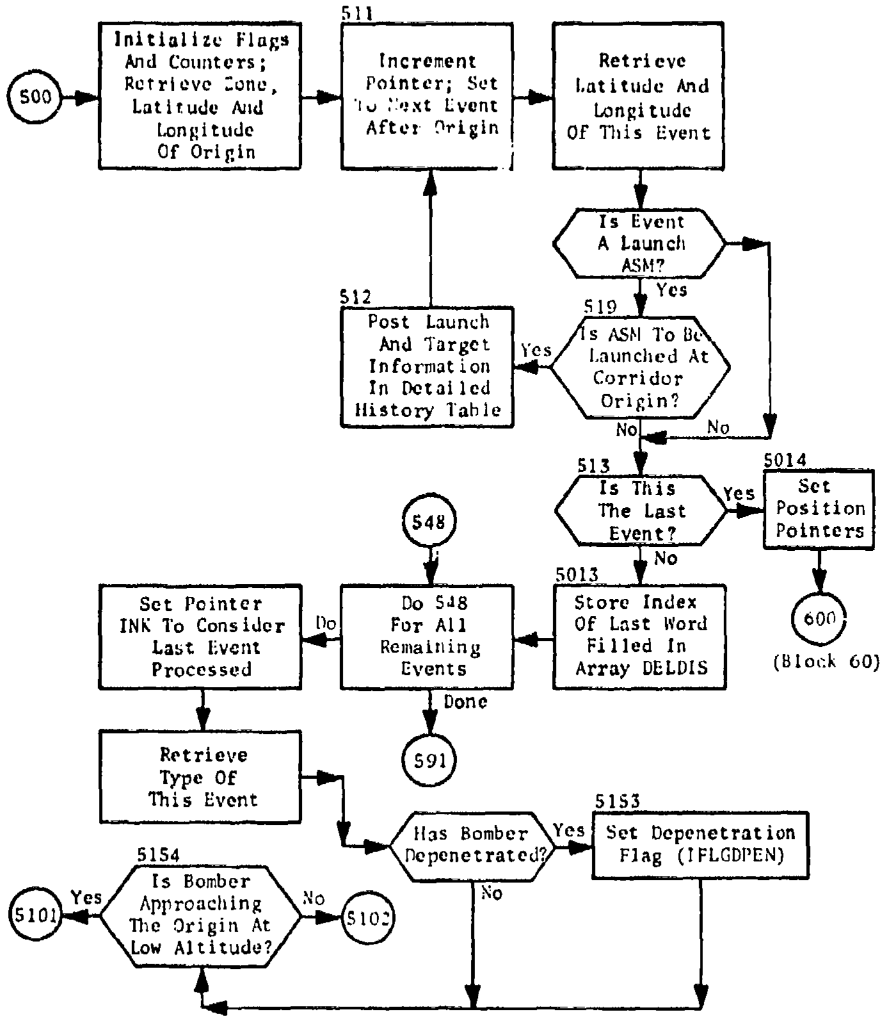


Fig. 168. Program PLNTPLAN
Block 50: Apply GOLOW2 Before First Target
(Sheet 1 of 5)

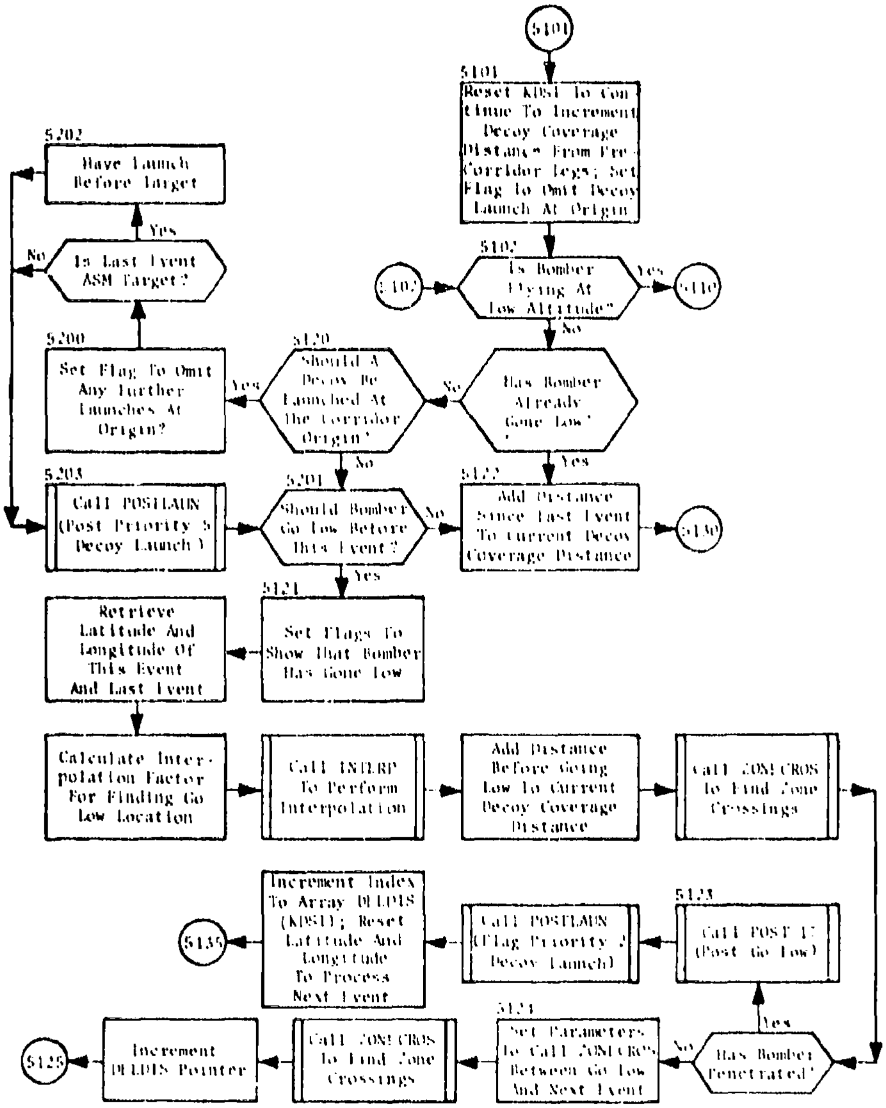


Fig. 168. (cont.)
(Sheet 2 of 5)

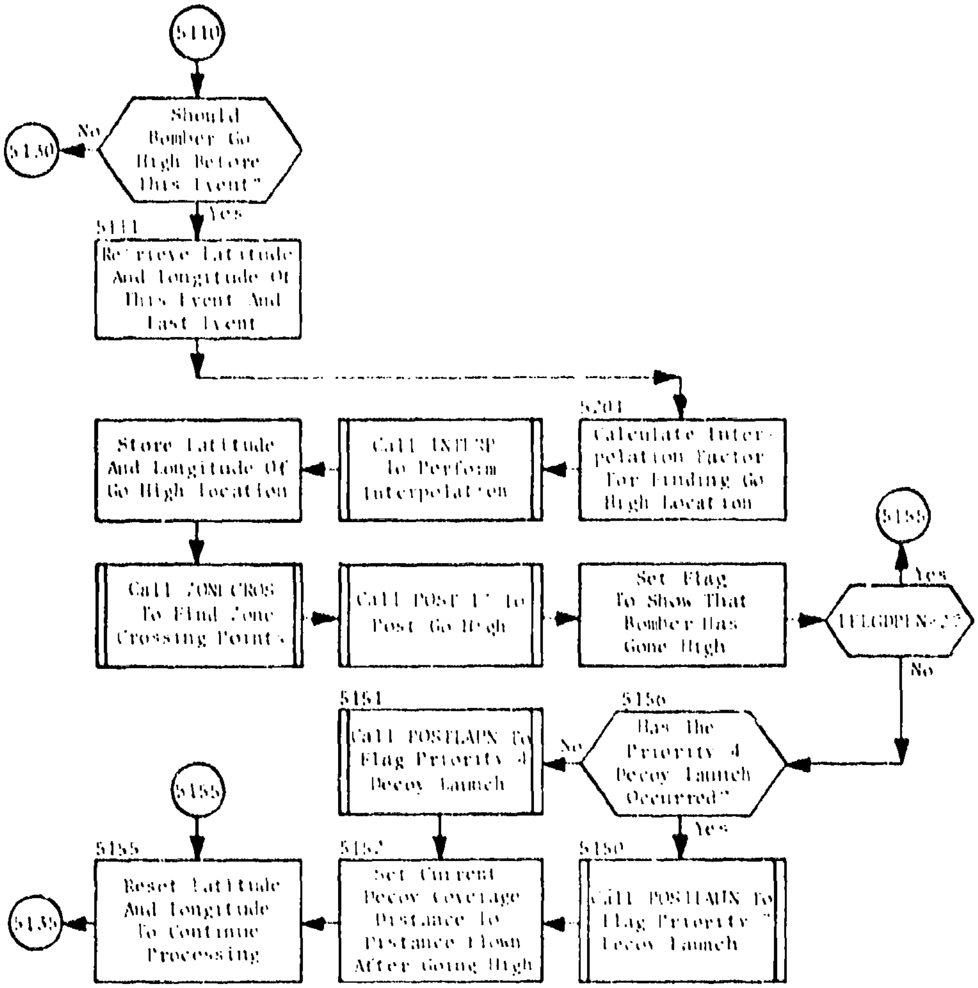


Fig. 168. (cont.)
(Sheet 5 of 5)

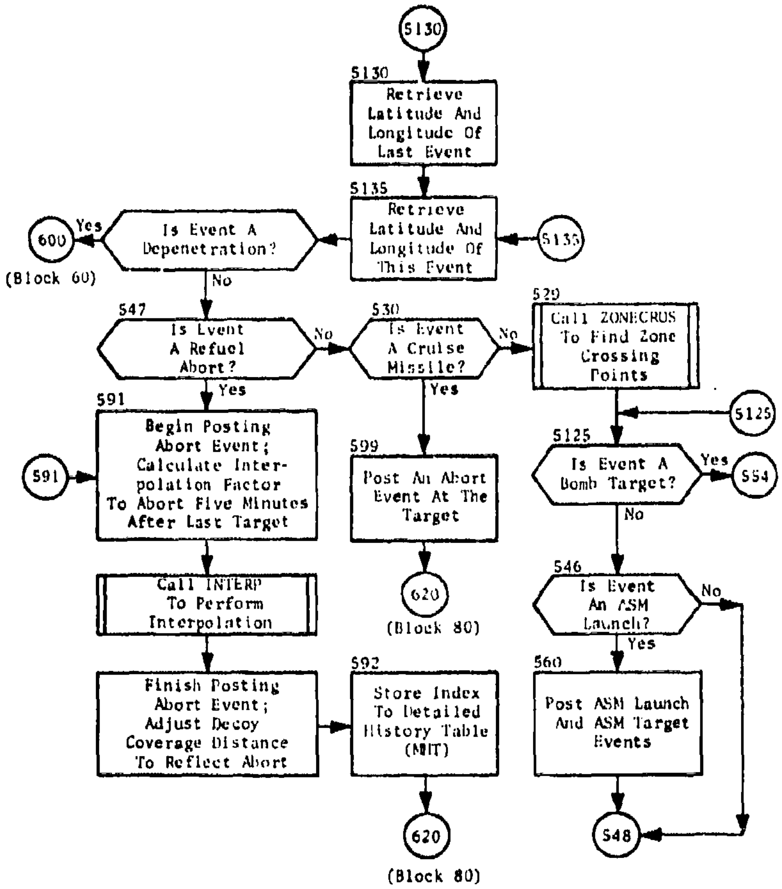


Fig. 168. (cont.)
(Sheet 4 of 5)

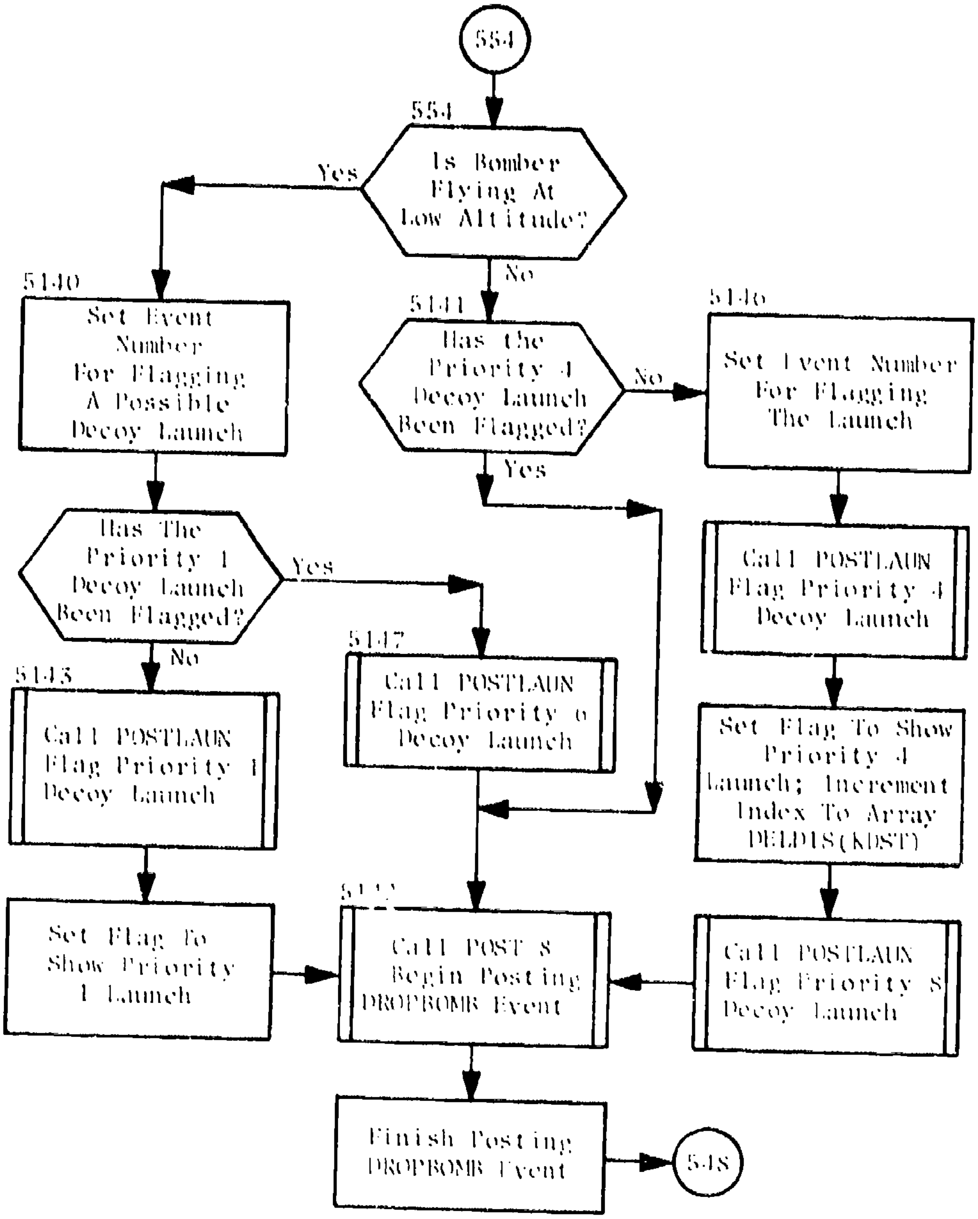


Fig. 168. (cont.)
(Sheet 5 of 5)

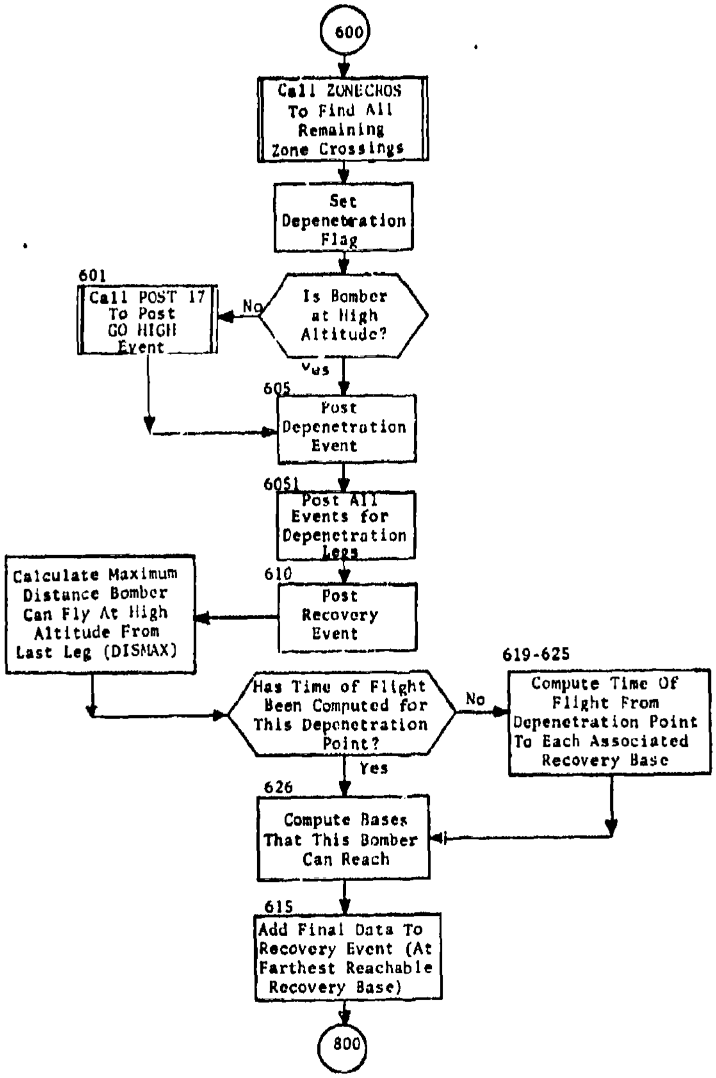


Fig. 169. Program PLNTPLAN
Block 60: Post Depenetration Events

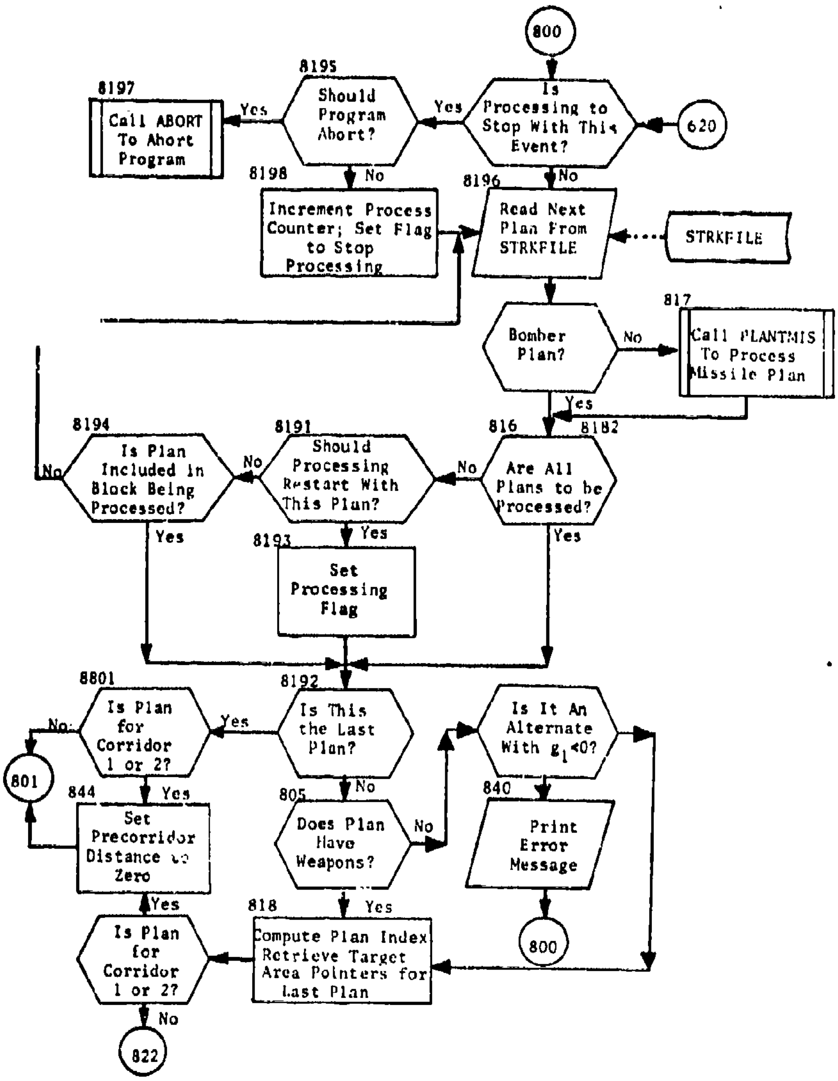


Fig. 170. Program PLNTPLAN
Block 80: Read Next /OUTSRT/ Record, Convert Last One
(Sheet 1 of 10)

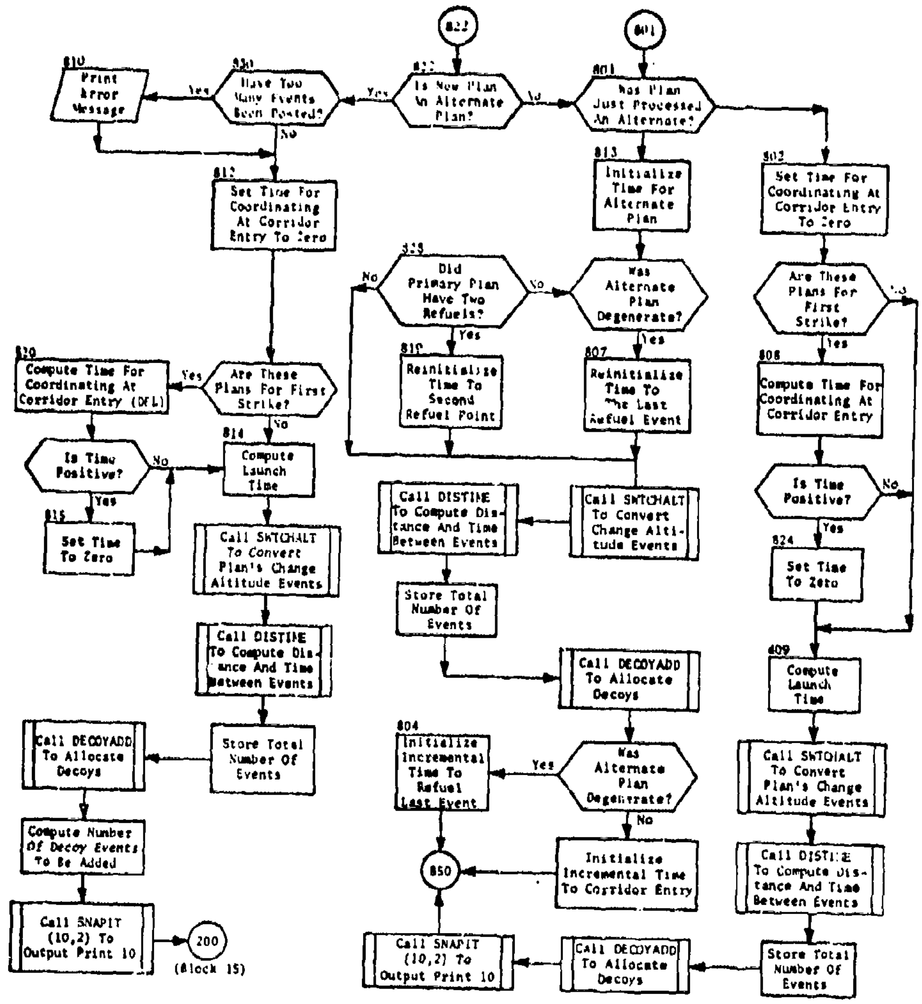


Fig. 170. (cont.)
(Sheet 2 of 10)

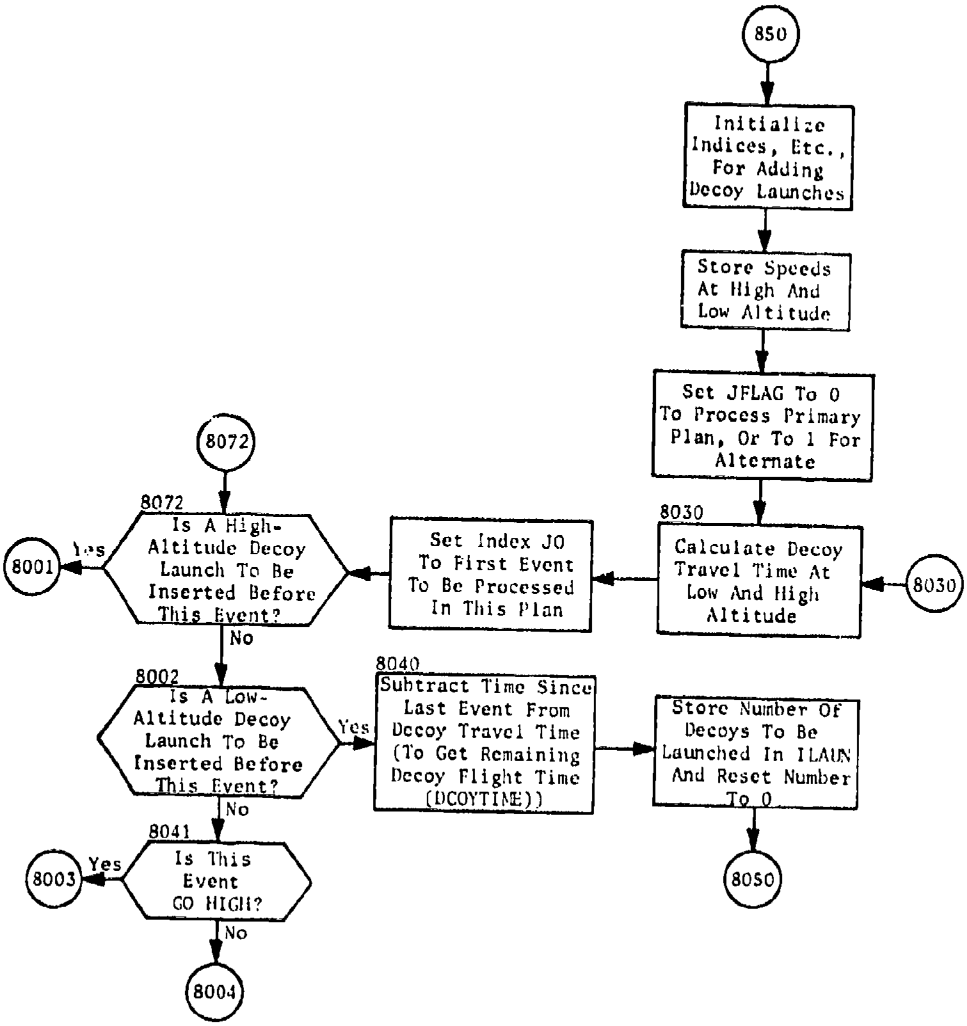


Fig. 170. (cont.)
(Sheet 3 of 10)

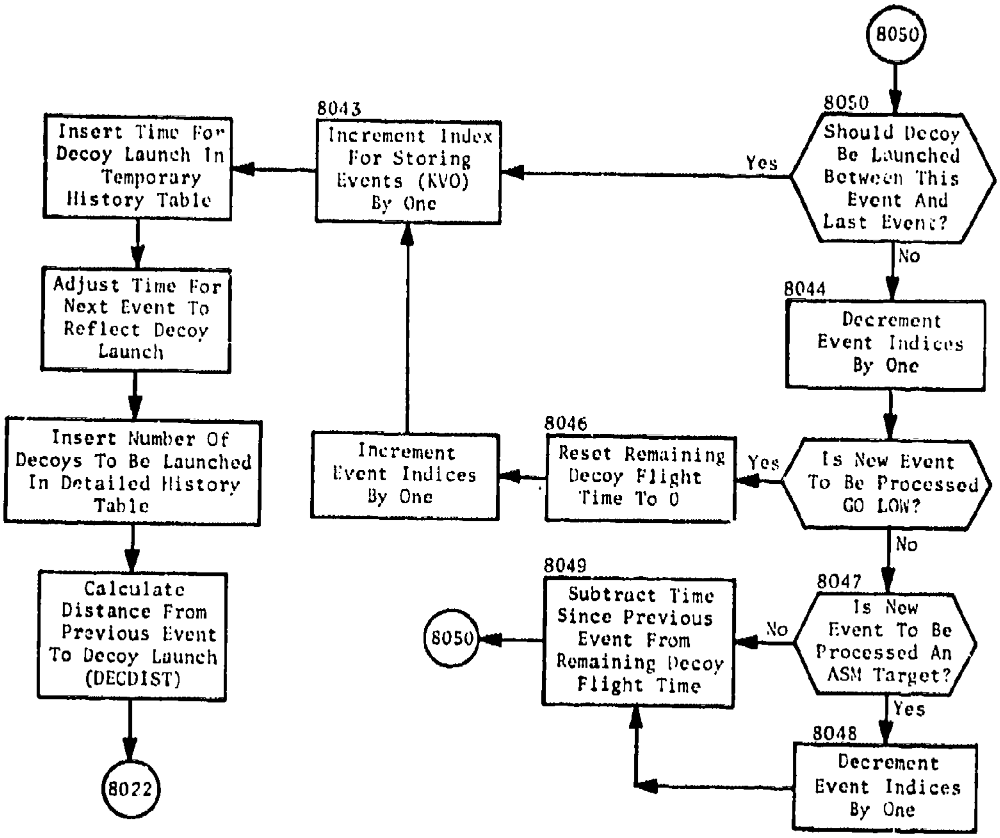


Fig. 170. (cont.)
(Sheet 4 of 10)

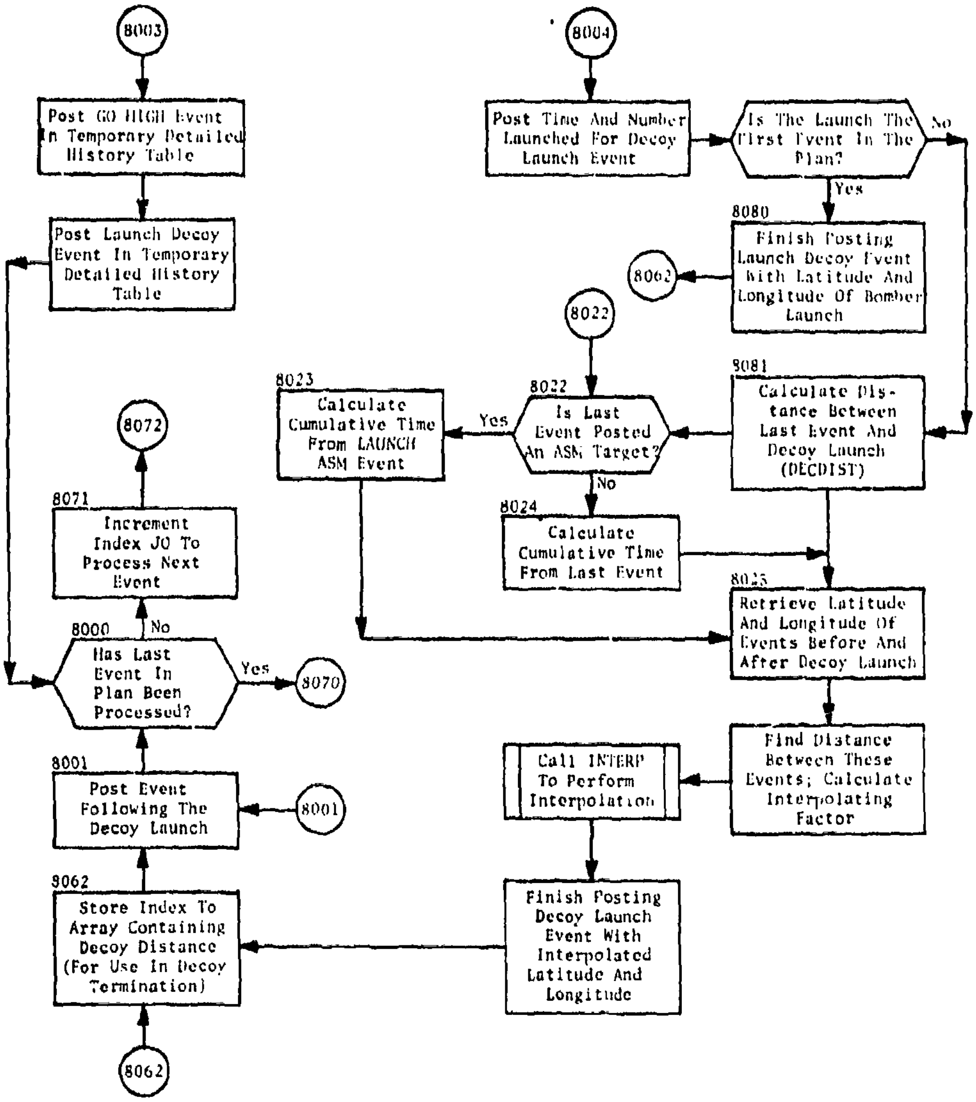


Fig. 170. (cont.)
(Sheet 5 of 10)

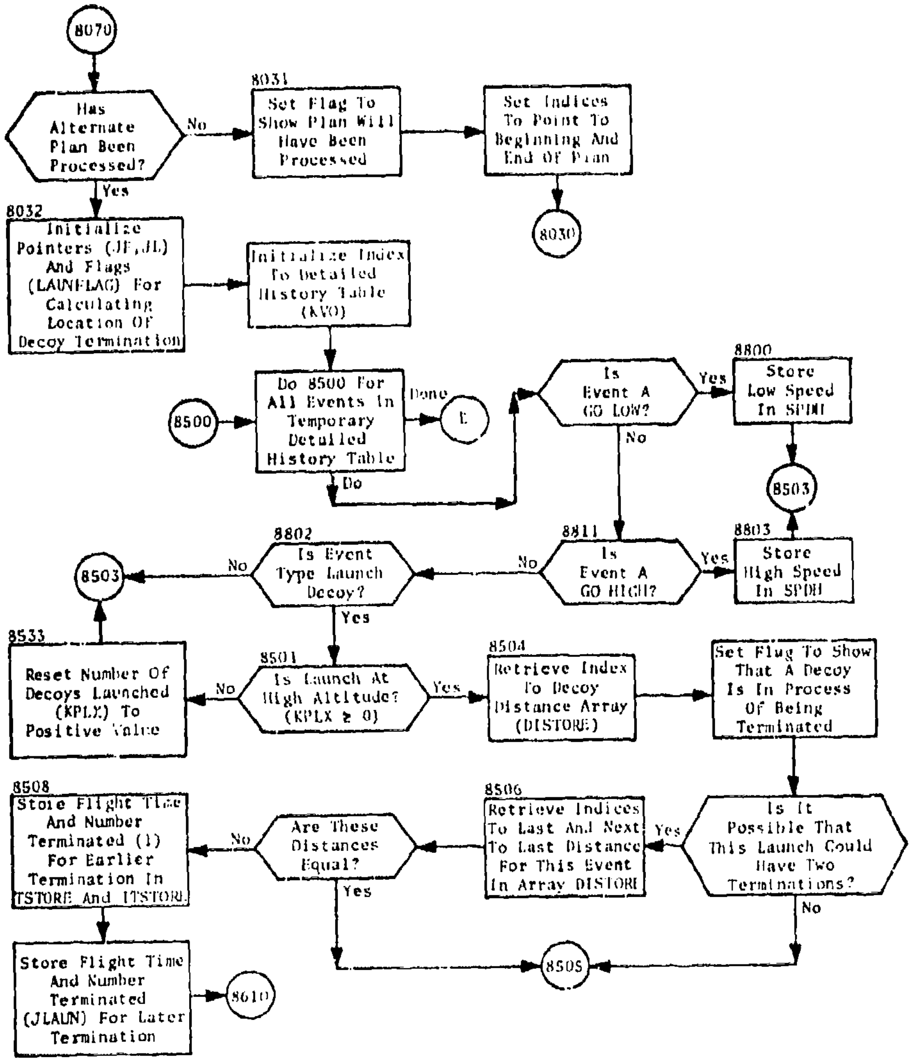


Fig. 170. (cont.)
(Sheet 6 of 10)

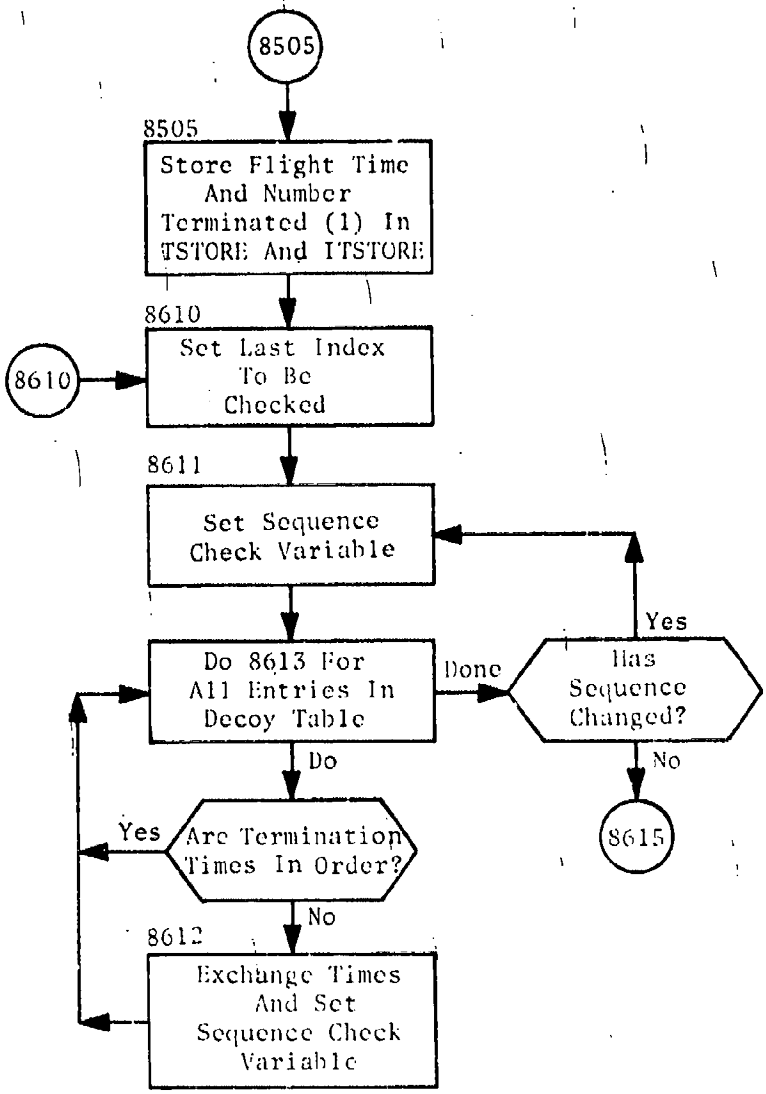


Fig. 170. (cont.)
(Sheet 7 of 10)

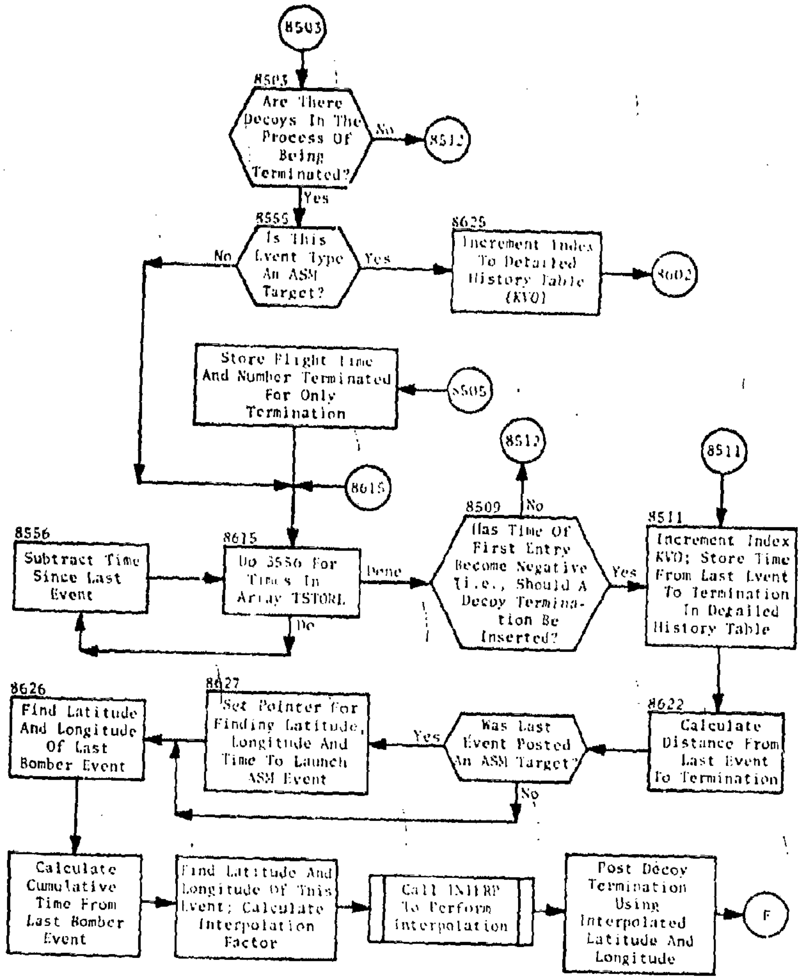


Fig. 170. (cont.)
(Sheet 8 of 10)

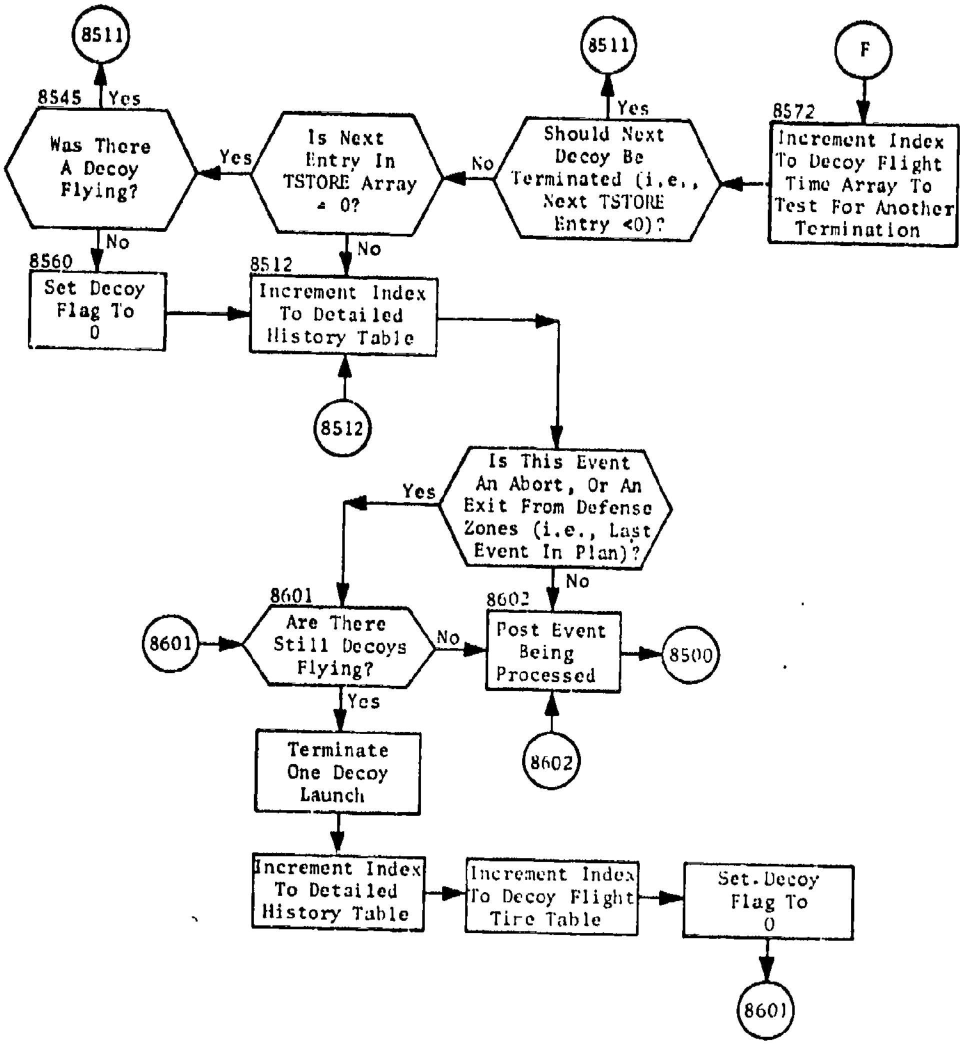


Fig. 170. (cont.)
(Sheet 9 of 10)

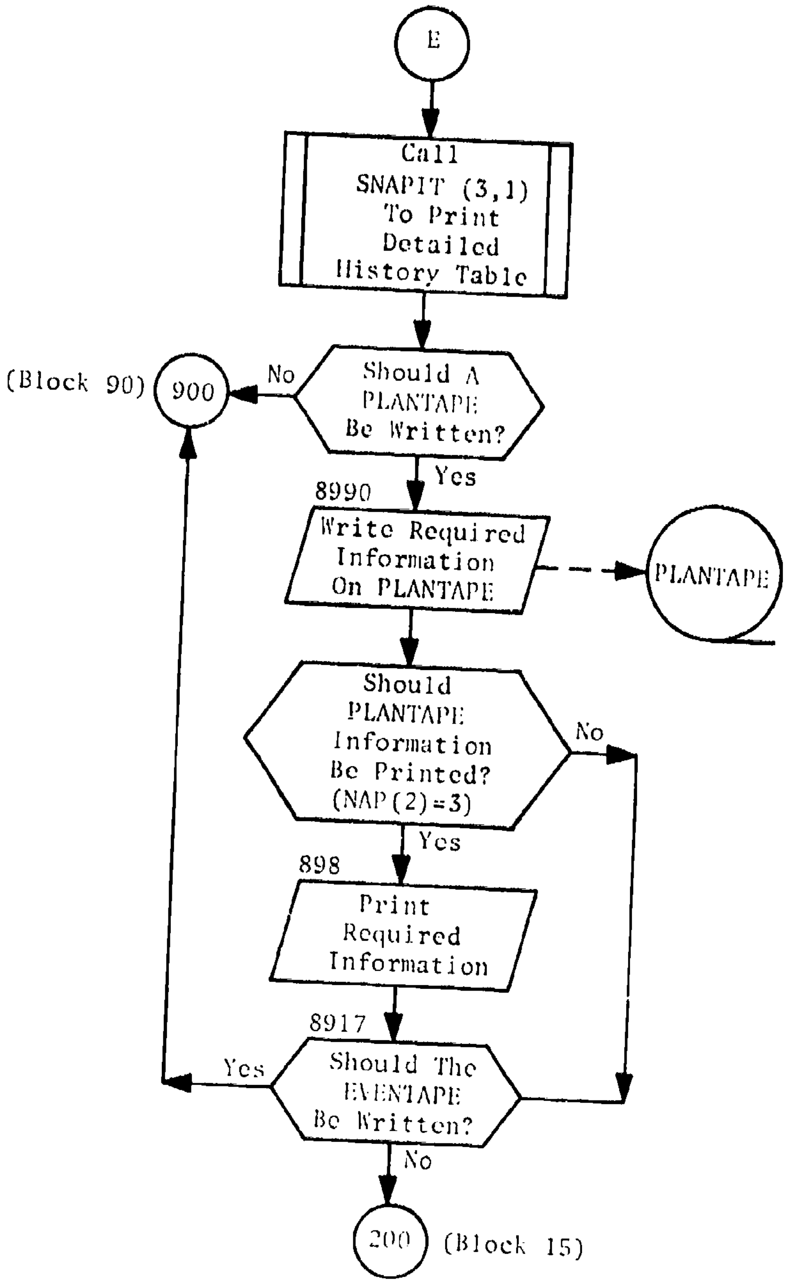


Fig. 170. (cont.)
(Sheet 10 of 10)

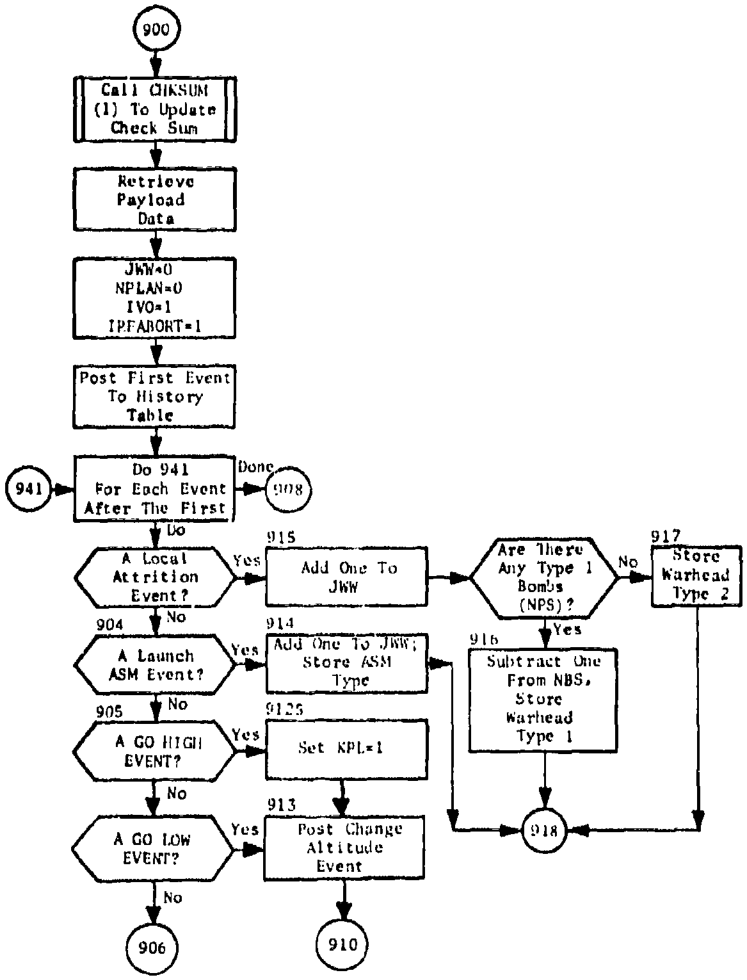


Fig. 171. Program PLNTPLAN
Block 90: Process Final Plan and Write on EVENTAPE
(Sheet 1 of 3)

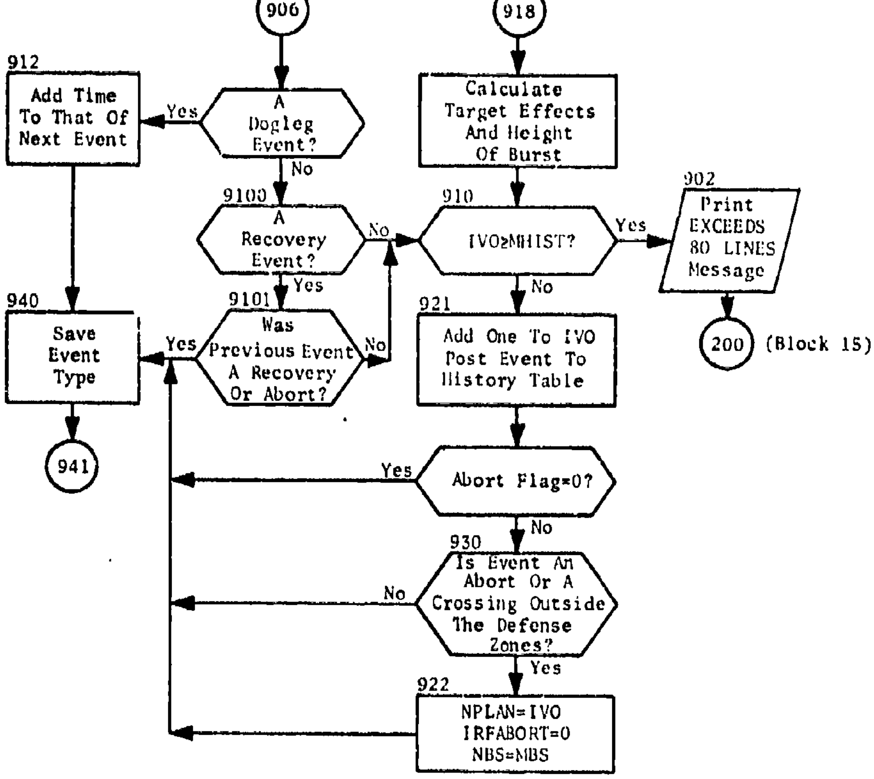


Fig. 171. (cont.)
(Sheet 2 of 3)

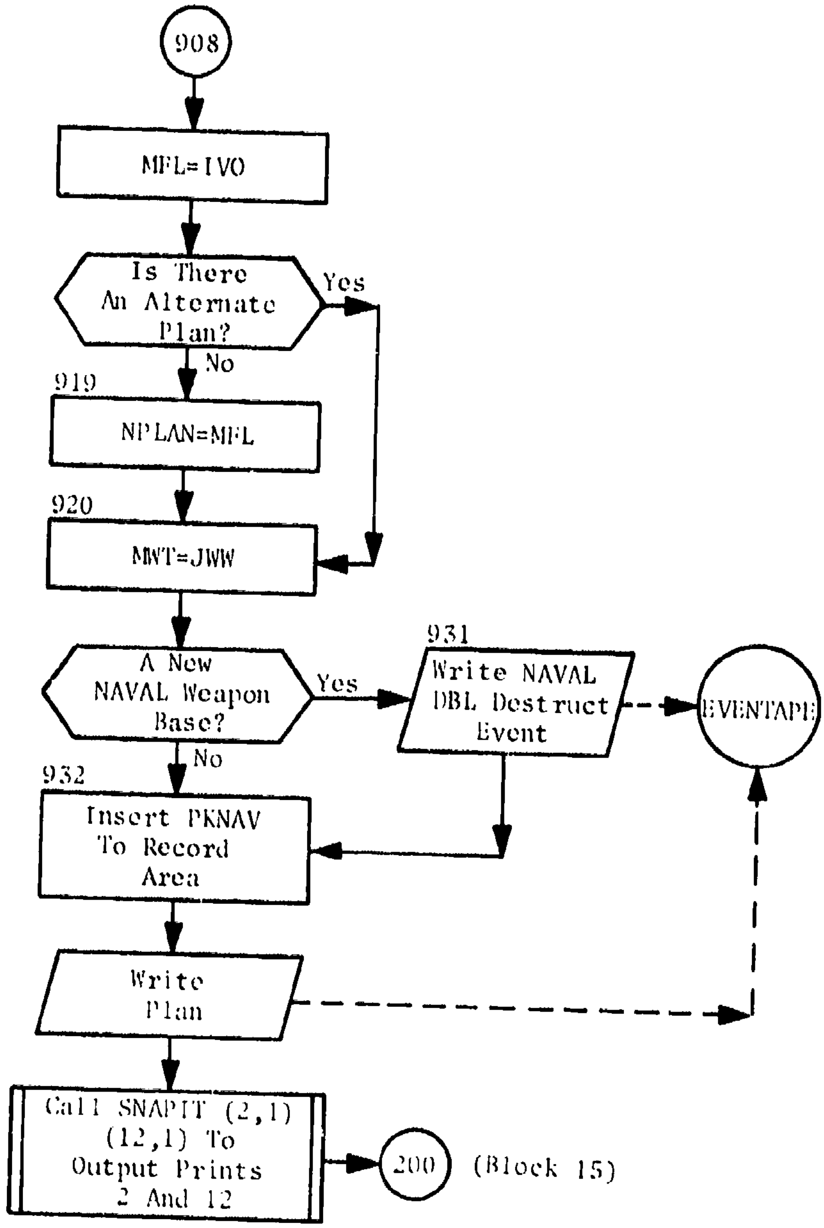


Fig. 171. (cont.)
(Sheet 3 of 3)

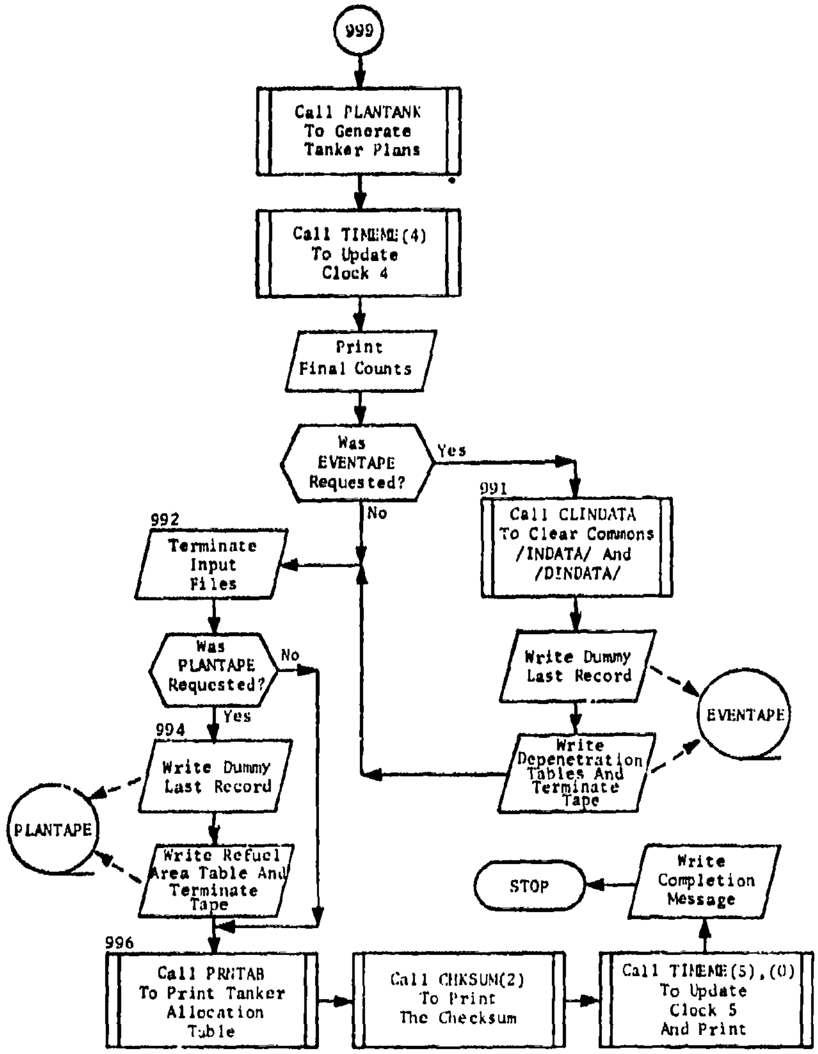


Fig. 172. Program PLNTPLAN
Block 100: Terminations

COMMON BLOCK DEFINITION

External Common Blocks

The common blocks used by program PLNTPLAN in processing input/output files are shown in table 47.

Internal Common Blocks

In addition to the common blocks associated with I/O operations, the common blocks described in table 48 are used within PLNTPLAN.

Table. 47. Program PLNTPLAN External Common Blocks (Sheet 1 of 5)

INPUT DATA FROM BASFILE

BLOCK	VARIABLE OR ARRAY*	DESCRIPTION
ASMTABLE	IWHDASM(20)	Warhead index
	RANGEASM(20)	Range
	RELASM(20)	Reliability
	CEPASM(20)	CEP
	SPEEDASM(20)	Speed
CORRCHAR	PCLAT(30)	Latitude of corridor point I
	PCLONG(30)	Longitude of corridor point I
	PCZONE(30)	Defense zone in which corridor I origin is located
	RPLAT(30)	Latitude of corridor I origin
	RPLONG(30)	Longitude of corridor I origin
	ENTLAT(30)	Latitude of corridor I entry
	ENTLONG(30)	Longitude of corridor I entry
	CRLENGTH(30)	Distance from corridor I entry to corridor I origin
	KORSTYLE(30)	Power of y versus x
	ATTRCORR(30)	High-altitude attrition per nautical mile unsuppressed
	ATTRSUPF(30)	High-altitude attrition per nautical mile suppressed
	HILOATTR(30)	Ratio low- to high-altitude attrition (less than 1)
	DEFRANGE(30)	Characteristic range of corridor defense
	NPRCRDEF(30)	Number of attrition sections this corridor
	DEFDIST(30,3)	Distance of attrition section
	ATTPRPE(30,3)	Attrition in this attrition section
	NDATA	Total number of words in common /CORRCHAR/
DPENREF	DPLINK(50)	Depenetration point I link
	DPLAT(50)	Depenetration point I latitude
	DPLONG(50)	Depenetration point I longitude
	QFLAT(20)	Refuel point I latitude
	QFLONG(20)	Refuel point I longitude

*Parenthetical values indicate array dimensions. All other elements are single word variables.

Table 47. (cont.)
(Sheet 2 of 6)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
FILES	TGTFILE(2)	Target data file (unit and maximum length)
	BASFILE(2)	Data base information file (unit and maximum length)
	MSLTIME(2)	Fixed missile timing file (unit and maximum length)
	ALOCTAR(2)	Weapon allocation by targets file (unit and maximum length)
	TMPALOC(2)	Temporary allocation file (unit and maximum length)
	ALOCGRP(2)	Allocation by group file (unit and maximum length)
	STRKFIL(2)	Strike file (unit and maximum length)
	EVENTAPE	Simulator events tape
	PLANTAPE	Detailed plans tape
MASTER	IHDATE	Date of run which created BASFILE
	IDENTNO	Time of run which created BASFILE
	ISIDE	Side
	NRTPT	Number of route points
	NCORR	Number of corridors
	NDPEN	Number of depenetration corridors
	NRECOVER	Number of recovery bases
	NREF	Number of refuel areas
	NBNDRY	Number of boundary points
	NREG	Number of regions
	NTYPE	Number of weapon types
	NGROUP	Number of weapon groups
	NTOTBASE	Number of bases
	NPAYLOAD	Number of payload types
	NASMTYPE	Number of ASM types
	NWHDTYPE	Number of warhead types
	NTANKBAS	Number of tanker bases
	NCOMPLEX	Number of complex targets
	NCLASS	Number of weapon classes
	NALERT	Number of alert conditions
	NTGTS	Number of targets
	NCORTYPE	Number of corridor types
	NCNTRY	Number of country codes

Table 47. (cont.)
(Sheet 3 of 6)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
NAVAL	NNAVAL	Length of /NAVAL/ data arrays on BASFILE
	IDBL(200)	DBL data table index for group I
	PKNAV(200)	Single shot kill probability against naval targets for group I
	ISVX(6)	Temporary storage array
	LSTDBLDS	Index of last base for which a DBL destruct event was written
PAYLOAD	NOBOMB1(40)	Number of bombs of type 1 (For MIRVs, the number of IRVs)
	IWHD1(40)	Type index of first bomb
	NOBOMB2(40)	Number of bombs of type 2
	IWHD2(40)	Type index of second bomb
	NASM(40)	Number of ASMs
	IASM(40)	ASM type
	NCM(40)	Number of countermeasures (bombers) Degradation factor (missiles)
	NDECOYS(40)	Number of decoys (for MIRVs, the number of terminal decoys per IRV)
	NADECOYS(40)	Number of area decoys
	IMIRV(40)	MIRV system identification number
PLANTYPE	INITSTRK	Indicator for first or second strike
	CORMSL	Coordination time parameter for missiles
	CORBOMB	Coordination distance for bombers
WPNGRPX	ITYPEX(200)	Type number from BASEFILE
	DBLX(200)	DBL probability from BASFILE
WPNTYPEX	REL(80)	Weapon reliability from BASFILE
	IWHTYPE(80)	Type name from BASEFILE
	IFNCTN(80)	Function code for weapon
TANKER	INDEXTK	Tanker index
	TKLAT	Tanker latitude
	TKLONG	Tanker longitude
	IREFTK	Tanker refuel area index
	NPSQNTK	Number of tankers per squadron
	NALRTK	Number of alert tankers
	SPEEDTK	Tanker speed
	DLYALTK	Delay for alert tankers
	DLYNLTK	Delay for nonalert tankers

Table 47. (cont
(Sheet

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
TANKER (cont.)	TTOS	Total time on station
	ITYPETK	Tanker type
	TANKRNGE	Tanker range
7	IINDEXTK(60)	Tanker base index
	TKRLAT(60)	Latitude of tanker base I
	TKRLONG(60)	Longitude of tanker base I
	IIREFTK(60)	Refuel area for tankers where n>0→ must refuel at area n
	NTKPSQN(60)	Number of tankers in squadron I
	NALRTNK(60)	Number of alert tankers at base I
	TANKSPD(60)	Speed of tankers at base I
	TKDLYALT(60)	Delay for alert tankers at base I
	TKDLYNL(60)	Delay for nonalert tankers at base I
	TKTTOS(60)	Total time on station
	IITYPTK(60)	Tanker type
	TRANGE(60)	Tanker range
9*	ILAUNDEC(90)	Number of decoys launched
	TIMELAUN(90)	Time of decoy launch
	DISTORE(90,6)	Distance traveled by decoy
	HDTX(90)	Temporary line array
	KPLX(90)	Temporary place array
	JTPX(90)	Temporary event number array
	HLAX(90)	Temporary latitude array
	HLOX(90)	Temporary longitude array
	TZTX(90)	Temporary weapon offset latitude array
	TZNX(90)	Temporary weapon offset longitude
	IWHX(90)	Temporary weapon type index
	PAX(90)	Temporary probability of arrival array
	CMTX(90)	Temporary cumulative time array
	BPLINK(200)	Boundary point link
Known elsewhere as /BOUNDARY/	BPLAT(200)	Boundary point latitude
	BPLONG(200)	Boundary point longitude
	BPZONE(200)	Zone for boundary point

*This common block is redefined when used in subroutines PLANTANK and VAM. The alternate definition is shown under Internal Common Block, Table 48.

Table 47. (cont.)
(Sheet 5 of 6)

BLOCK BLOCK	VARIABLE OR ARRAY	DESCRIPTION
9 (cont.)	NEXTZONE(200)	Next zone for boundary line
Known elsewhere as /CHARTER/	KOUNT(30) IHAP(30) MOUNT(50) JHAP(50)	Miscellaneous plan counters and indices
Known elsewhere as /HAPPEN/	JAPTYPE(250)	Event type
	HAPLAT(250)	Event latitude } History table
	HAPLONG(250)	Event longitude } History table
	HAPDIST(250)	Incremental distance
8	RECLAT(50)	Tanker recovery latitude
	RECLONG(50)	Tanker recovery longitude
10	RCBLAT(50,4)	Recovery base latitude } Indexed for each of 4 bases assigned to the Ith de-penetration point
	RCBLON(50,4)	Recovery base longitude
	INDBAS(50,4)	Recovery base index
	INDCAP(50,4)	Recovery base capacity
	DISTR(50,4)	Distance to recovery
	TOF(50,4)	Time of flight to recovery

INPUT DATA FROM STRKFILE

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
BLOCK	LOCK/BLOCK(320)	Initially, contains the missile plan from STRKFILE (see STRKFILE format); also used to store the output plan record
OUTSRT	----	Contains the input bomber record from STRKFILE (see STRKFILE format)

Table 47. (cont.)
(Sheet 6 of 6)

OUTPUT DATA FOR EVENTAPE

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
BLOCK	LOCK/BLOCK(320)	Initially, the missile plan record from STRKFILE (see STRKFILE format); later, the missile plan record for the EVENTAPE and/or PLANTAPE
INDATA	----	The output EVENTAPE bomber or tanker record (see EVENTAPE format)

Table 48. Program PLNTPLAN Internal Common Blocks (Sheet 1 of 11)

BLOCK	VARIABLE OR ARRAY*	DESCRIPTION
ARTIME	ARTIME(50)	Earliest bomber arrival time at refuel area I
	NBUDREF	Number of "buddy" refuelings required
	NBOMBREF(50)	Number of bombers assigned to refuel area I
	NTANKREF(50)	Number of tankers assigned to refuel area I
	IARVLS/ARVLS(2,1000)	ARVLS(1,I) = time of the Ith bomber refuel processed by PLNTPLAN; IARVLS(2,I) = the refuel area for that bomber refuel
ASMARRAY	ALAT(10)	Aim point latitude
	ALON(10)	Aim point longitude
	IFLY(10)	Fly point flag
	IDIS(10)	Distance from fly point to ASM target
	IORD(10)	Sort index
	JAY	Index communicated to PREFLY1, PREFLY2
	DIST	Distance communicated to PREFLY1, PREFLY2, POSTFLY
BOUND	X1	Latitude of beginning point
	Y1	Longitude of beginning point
	X2	Latitude of end point
	Y2	Longitude of end point
	IZN	Current defense zone or sector number
	XR	Latitude of zone crossing
	YR	Longitude of zone crossing

*Parenthetical values indicate array dimensions. All other elements are single word variables.

Table 48. (cont.)
(Sheet 2 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
BOUND (cont.)	NZN	New zone number
	IZIT	Zone crossing indicator
BOUNDPT	ZLAT1(20)	Beginning latitude of boundary line
	ZLON1(20)	Beginning longitude of boundary line
	ZLAT2(20)	End latitude of boundary line
	ZLON2(20)	End longitude of boundary line
	KK	Index of boundary line
CONTROL	ICE	Indicates which output option selected (1 = EVENTAPE only, 2 = PLANTAPE only, 3 = both)
	LSIDE	Indicates which side
	NPLOT	Indicates number of plots per page
	ISZE	Plot size
	SCALE	Plot scale
CORCOUNT	IH	Points to line of /HAPPEN/ where current corridor begins
	KC	Number of lines in /HAPPEN/ describing current corridor
	JH	Points to line in /HAPPEN/ where current depenetration corridor begins (1 = depenetration point)
	LC	Number of lines describing current depenetration corridor (see common block /9/ for description of /HAPPEN/)
DINDATA	HDT(90)	Time - for detailed history - of event I
	KPL(90)	Place - for detailed history - of event I
	JTP(90)	Event type - for detailed history - of event I

Table 48. (cont)
(Sheet 3 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
DINDATA (cont.)	HLA(90)	Latitude - for detailed history - of event I
	HLO(90)	Longitude - for detailed history - of event I
	TZT(90)	Weapon offset latitude - of event I
	TZN(90)	Weapon offset longitude - of event I
	PA(90)	Probability of arrival at target - of event I
	MHT	Total number of lines in detailed plan (primary and alternate)
	NPL	Number of planned events (primary)
DINDATA2	CMT(90)	Cumulative time - for detailed history
	IWH(90)	Weapon type index - for detailed history
DISTC	DISTC(20)	Distances between target events
EVENTS	LAUNM	Launch missile code
	LAUNB	Bomber launch code
	LEREFUEL	Refuel code
	INSECTOR	Defense zone boundary or sector crossing code
	LOCLATTR	Local attrition or drop bomb event code
	LAUNASM	Launch ASM event code
	LAUNDCOY	Launch Decoy event code
	LANDHO	Recovery Event code
	LOHI	Change Altitude event code
	MISSATTR	Missile attrition event code
	LEGDOG	Dogleg event code
	LABORT	Abort event code

Table 48. (cont.)
(Sheet 4 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
EVENTS (cont.)	LENTEREF	Enter refuel area event code
	LEAVEREF	Leave refuel area event code
	IGOHI	Go to high-altitude event code
	IGOLOW	Go to low-altitude event code
FINDZONE	FINDLAT	Latitude of point for which zone number is needed
	FINDLON	Longitude of point for which zone number is needed
	IFOUNDZN	Zone for (FINDLAT, FINDLON)
HILO	ISTOREHI	Number of event in /OUTSRT/ after which GO HIGH occurs
	ISTORELO	Number of event in /OUTSRT/ after which GO LOW occurs
	IGOLEFT	Set to 1 if GO LOW range is available after depenetration
	FACHI	Distance after event ISTOREHI at which GO HIGH is located
	FACLO	Distance after event ISTORELO at which GO LOW is located
	GOLO	Amount of GO LOW range remaining for depenetration
ICLASS	IBOMBER	Bomber class index
	ITANKER	Tanker class index
IDP	IDP(2)	Depenetration corridor index number as reassigned when last target is an ASM target; IDP(1) is for primary plan, IDP(2) is for the alternate
IFLGDPEN	IFLGDPEN	Depenetration flag
	LOWZONE:	= 1 at calculated GO LOW = 2 if GO LOW precedes first insector = 3 if GO LOW posting moved to subroutine ZONECROS

Table 48. (cont.)
(Sheet 5 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
IFLGDPEN (cont.)	IC1FLG	Tactical aircraft flag
IGO	IGO800	Set to 1 for degenerate target area
IOUTOLD		(Save area for /OUTSRT/ information)
	IOUTOLD	Index of current sortie
	KOKO	Index of current ASM type
	MYGROUPO	Index of current group
	MYCORRO	Index of current corridor
	LPAYLOAD	Index of current payload
	LREF	Index of current refuel area
	LDPEN	Index of current depenetration point
	SPDHIO	Speed at high altitude
	SPDLOO	Speed at low altitude
	FFCTNO	Function of vehicle
	NHAPO	Number of events
	IBJECO(20)	Index of target I
	IBDESO(20)	Designation of target I
	IBTSKO(20)	Task number of target I
	IBCTYO(20)	Country code of target I
	IBFLGO(20)	Flag of target I
IRF	IRF	Assigned refuel area index
	NRF	Number of refuel areas, including those assigned by PLNTPLAN
IRFTK	IRFTK	Refuel area index
ITAB	ITAB	Flag for subroutines FINDZONE and BOUNDARY
KEYLENG	LOS	Length of /OUTSRT/ record (STRKFILE)

Table 48. (cont.)
(Sheet 6 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
KEYLENG (cont.)	LIN	Length of /INDATA/ record (EVENTAPE)
	LDN	Length of common block /DINDATA/
	LINC	Length of /INDATA/ except for last array
	LTK	Length of tanker record
	LMIS	Length of missile record, STRKFILE
	LMO	Number of good words in missile /INDATA/
	LDBL	Length of record for naval DBL event
KEYS	KEYDPEN	Key for packing depenetration point
	KEYBMX	Key for packing recovery point
LASM	U1	Latitude of beginning point of bomber path
	V1	Longitude of beginning point of bomber path
	U2	Latitude of end point of bomber path
	V2	Longitude of end point of bomber path
	UAT	Latitude of ASM target
	VAT	Longitude of ASM target
	RASM	Range of ASM
	RLAT	Latitude of ASM aim point
	RLONG	Longitude of ASM aim point
LAUNSNAP	INRANGE	Set to zero if ASM target is in range of flight path; otherwise to one
	FRACPATH	Fraction of total path at which ASM is launched

Table 48. (cont.)
(Sheet 7 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
MAX (contains the QUICK maximum limits)	MAIRDEZ	ABM defense zones (20)
	MALERT	Alert conditions (2)
	MASMTYP	ASM types (20)
	MBNDRY	Boundary (200)
	MCCREGN	Command/control (20)
	MCLASS	Weapon classes (2)
	MCNTRYS	Country codes (250)
	MCORR	Penetration corridors (30)
	MCORTYP	Corridor types (5)
	MDPEN	Depenetration corridors (points) (50)
	MDEPNLG	Depenetration legs (50)
	MGROUP	Weapon groups (200)
	MPAYLOD	Payload types (per side) (40)
	MRECOVR	Recovery bases (points) (200)
	MRECVLG	Recovery legs (60)
	MREF	Refuel points (directed) (20)
	MRTLEG	Route legs (200)
	MRTPT	Route points (200)
	MSPERMT	Sites per multiple target (5)
	MTANKBS	Tanker bases (50)
	MTARCLS	Target classes (15)
	MTARCOL	Targets, collocated (4000)
	MTARCPX	Target complexes (total) (4000)
	MTARERS	Targets per collocation island (100)
	MTARGET	Targets (allocator) (5000)
	MTARIND	Target index numbers (12000)
	MTARSEC	Targets per earth sector (4000)

Table 48. (cont.)
(Sheet 8 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
MAX (cont.)	MTARTEI	Targets with terminal ABM interceptors (500)
	MTARTYP	Target types (total) (250)
	MTARVAL	Target complex with value > 0 (2500)
	MTELMCM	Target elements per complex (40)
	MTOTBAS	Weapon bases per group (150)
	MTYPE	Weapon types (missiles + bombers per side) (80)
	MVULN	Unique target vulnerabilities within the game base (63)
	MWEAPGP	Weapons per group (missiles + bombers) (1000)
	MWHDTPE	Warhead types (50)
	MZONEPT	Zone points (200)
	MZONES	Zones (63)
	MTARPCL	Target types per class (40 for ICLASS 1 or 2; 20 for others)
MH	MHMINA(10)	Line in common /DINDATA/ where target area begins
	MHMAXA(10)	Line in common /DINDATA/ where target area ends
	MHMN	Lower plot marker for sortie
	MHMX	Upper plot marker for sortie
MH2	MHMIN(2)	Lower plot markers for sortie
	MHMAX(2)	Upper plot markers for sortie
MISCT	MISCT	Missile booster count
	MTARGCT	Missile target count
MRVFLG	MRVFLG	Set to 1 if plan contains MIRVs
POLITE	S1	Latitude of beginning interpolation point
	T1	Longitude of beginning interpolation point

402-548 O - 72 - 28

Table 48. (cont.)
(Sheet 9 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
POLITE (cont.)	S2	Latitude of interpolation end point
	T2	Longitude of interpolation end point
	FACTOR	Interpolation factor or fraction
	SR	Latitude of interpolated point
	TR	Longitude of interpolated point
REF	RFLAT(50)	Latitude of refuel area I
	RFLONG(50)	Longitude of refuel area I
RL	RL	Decoy low-altitude range
	RH	Decoy high-altitude range
SNAPON	NAP(15)	Set to three for active print I; set to one for inactive print I
SPASM	SPASM	Speed of ASM currently used
TEMPO	DT(50)	Distance or time temporary storage
	JT(50)	Event type temporary storage
	TLT(50)	Latitude temporary storage
	TLN(50)	Longitude temporary storage
	LPL(50)	Place index temporary storage
TIMELINE	ITIMETYP(40)	CORMSL type (0-Flight; 1-Line)
	CORMSLX(40)	Percent flight complete or time on line
	FLTMIN(40)	Minimum flight time in minutes
	INDXFIX(1000)	Target index numbers for fixed weapons
	TIME(1000)	Arrival times for fixed weapons
	TDATA(252)	Temporary storage area for flight data

Table 48 (cont.)
(Sheet 10 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
TIMELINE (cont.)	ZLAT(50,2)	Latitude of timing line endpoints
	ZLONG(50,2)	Longitude of timing line endpoints
	XC(50)	X-coordinate of cross product vector of timing line
	YC(50)	Y-coordinate of cross product vector of timing line
	ZC(50)	Z-coordinate of cross product vector of timing line
	DL(50)	Length of timing line
	TMLNCH(18)	Launch time
	GOLD	Last fixed weapon group processed
	NFIXWPS	Number of fixed weapons
	NLEFT	Counter for fixed weapons
	NLINES	Number of timing lines
	MSLFIL(5)	Input area for record from MSLTIME file (see MSLTIME format)
VICINITY	VHB	Bomber cannot go high within VHB miles before target
	VHA	Bomber cannot go high within VHA miles after target
	VLB	Bomber cannot go low within VLB miles before target
	VLA	Bomber cannot go low within VLA miles after target
	GOMIN	Bomber cannot fly low for less than GOMIN minutes
1	DELDIS(6)	Decoy coverage distance
	LPRIORITY(20)	Possible decoy launch priority
	LMHT(90)	Possible Decoy Launch event number
	NDCYRQ(20)	Pointer to array DELDIS
	NPSLN	Number of possible decoy launches
	NUMDCOYS	Number of decoys available

Table 48. (cont.)
(Sheet 11 of 11)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
	COST(I,J)	Distance between tanker base I and refuel area J (where I=60, J=50)
	SOURCE(60)	Number of tankers at base I to be automatically assigned
	ISOL(110)	The Ith nonzero element in final VAM solution
	RBASLOC(110)	Tanker base corresponding to the Ith solution element
	CBASLOC(110)	Refuel area corresponding to Ith solution element
	NSOL	Number of nonzero elements in VAM solution
	RMAX	Number of rows (tanker bases) in VAM problem
	IRCHK(60)	Set to one if base I tankers are not to be automatically allocated
	IRCDIF	Number of bases for which IRCHK(I)=1
	DISTREF(50)	Distances from current tanker base to refuel area I

SUBROUTINE ADJUST

PURPOSE:	To examine the target section of the plan to determine where GO HIGH and GO LOW events are to be placed with respect to the target events, and to adjust these as appropriate.
ENTRY POINTS:	ADJUST
FORMAL PARAMETERS:	None
COMMON BLOCKS:	ASMARRAY, DINDATA, DISTC, EVENTS, HILO, IGO, OUTSRT, POLITE, VICINITY
SUBROUTINES CALLED:	DISTF, INTERP, SNAPIT
CALLED BY:	PLNTPLAN

Method

Subroutine ADJUST allocates the go low ranges of G_2 (low-altitude range before the first target) and G_3 (low-altitude range after the first target) beginning at the corridor origin and covering the entire target area. The values for G_1, G_2, and G_3 are input from POSTALOC on the STRKFILE; the G_1 is allocated by blocks 27 and 30 of PLNTPLAN. ADJUST is called by PLNTPLAN just before the target list is processed. ADJUST begins by calculating the distance from event I to event I+1 in /ASMARRAY/ and storing it in DISTC(1). The initial go low point is then determined from the value of G_2. If $G_2>0$, the go low will occur G_2 miles before the first target. Here, the first target is defined to mean the first bomb target or the first ASM launch point after the corridor origin. If G_2 is such that the go low point is within 15 minutes (GOMIN) of the corridor origin, it is extended so that the go low occurs at the origin. If it is to go low at the origin according to G_2, any go high event posted at the end of PLNTPLAN's block 30 is cancelled and the go low event for G_2 is omitted. If an ASM Launch had been scheduled at the origin, and a go low is also to occur there, the ASM Launch point is recalculated to occur 5 minutes after the origin along the original flight path.

For plans in which $G_2 = 0$, the bomber will go low at the first target, provided that the range to be flown at low altitude after the first target $(G_3)>0$. If G_3 also equals 0, it will fly the entire mission at high altitude. If $G_2<0$, it will fly $- G_2$ miles beyond the first target before going low. The total low-altitude range in this case is $G_3 - (-G_2)$ miles.

Once the go low point has been found, the number of the event preceding the altitude change is stored in ISTORELO and the distance from that event to the go low is stored in FACLO. The point at which the go high event will occur then is determined by subtracting the distances in array DISTC from the available go low range. When the range becomes negative or zero, the index to the array will be set to the number of the event preceding the altitude change. This number is stored in ISTOREHI and the distance from that event to the go high event is stored in FACHI.

These preliminary locations must now be checked to ensure that the bomber does not change altitude within a critical distance of a target or ASM Launch. These distances are described by the variables VHB, VHA, VLB, and VLA, contained in common /VICINITY/. These variables represent the mileages which correspond to the constant time parameters THB, THA, TLB, and TLA. The settings in PLNTPLAN for these parameters are shown on page 876 in figures 174 and 175. If a change does occur within a critical distance, the values of ISTORELO (ISTOREHI) and FACLO (FACHI) are adjusted so that the altitude change is moved to a point which is the required distance away from the target. (See figures 174 and 175.) The distance flown at low altitude never is decreased by the move. For example, if the bomber originally were to go low less than VLA miles after the target, the altitude change is moved to a distance VLB miles before the target. If the critical distances to two (or more) targets overlap, the altitude change is moved, either forward or backward as the situation requires, past the entire cluster of targets.

In making these adjustments the amount of low altitude flight may be increased. This is illustrated by the example shown in figure 173. It shows two targets T_1 and T_2 with their associated neighborhoods drawn taking account of the parameters in figures 174 and 175 and a section of bomber path shown by a dotted line. In this case, a GO HIGH found, say at point p, would be moved first to point q, and finally to point r. The time of low-altitude flight would be increased, in this case, to at most twice the sum of THB + THA. For this to occur, targets would have to be within THB + THA minutes of flying time to each other.

Communication between subroutine ADJUST and program PLNTPLAN is established through common /HILO/. The final values of ISTOREHI, ISTORELO, FACHI, and FACLO are used by the main program to insert the CHANGALT events as the target list is being processed. Later, the CHANGALT events will be interpreted as GO HIGH or GO LOW events by subroutine SWTCHALT.

Figure 176 illustrates program ADJUST.

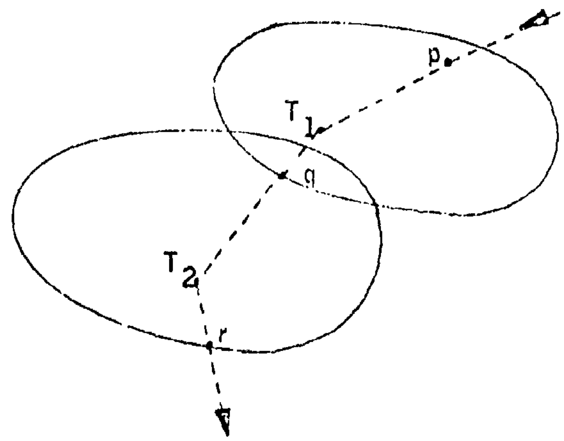


Fig. 173. Increase In Low-Altitude Flight

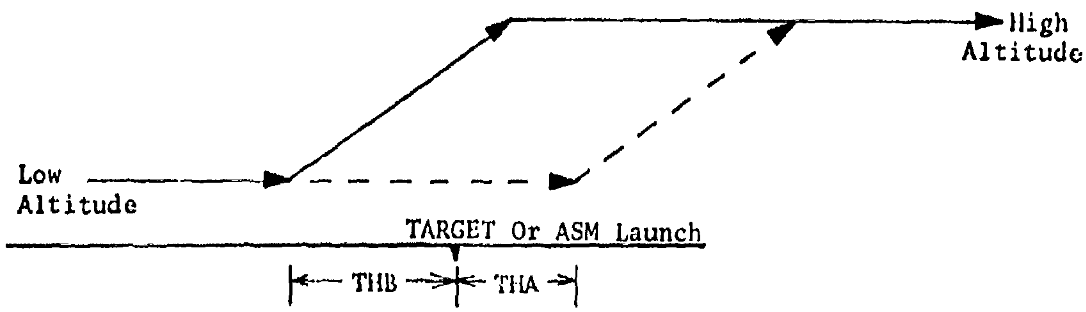


Fig. 174. High-Altitude Adjustment

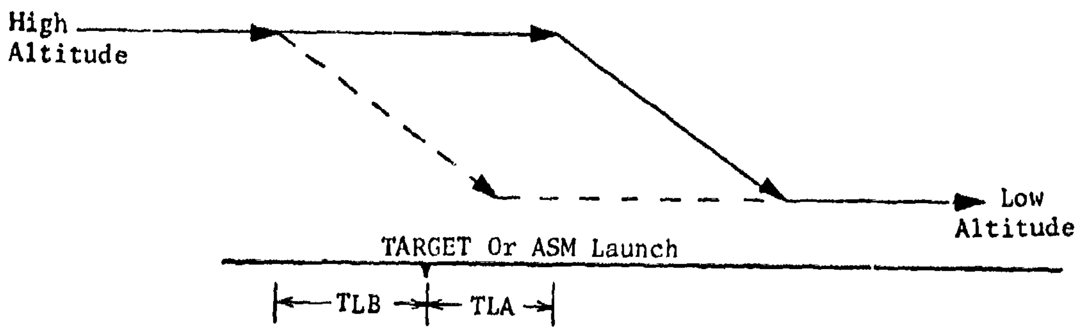


Fig. 175. Low-Altitude Adjustment

PARAMETER	TIME(MINUTES)	DESCRIPTION
THB	15	Time before a target during which the bomber may not go high
THA	2	Time after a target during which the bomber may not go high
TLB	10	Time before a target during which the bomber may not go low
TLA	3	Time after a target during which the bomber may not go low

(Variables VHB, VHA, VLB, and VLA in common /VICINITY/ represent the mileages which correspond to the time parameter THB, THA, TLB, and TLA)

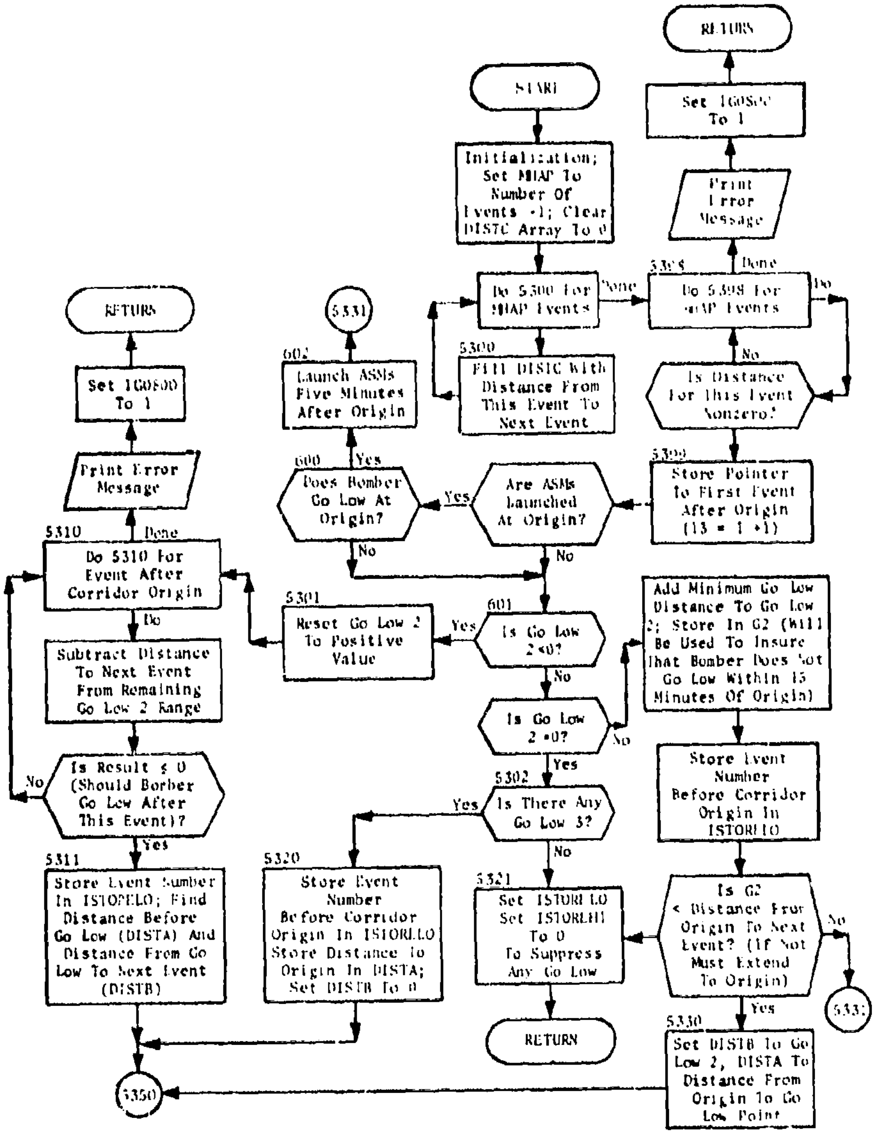


Fig. 176. Subroutine ADJUST
(Sheet 1 of 7)

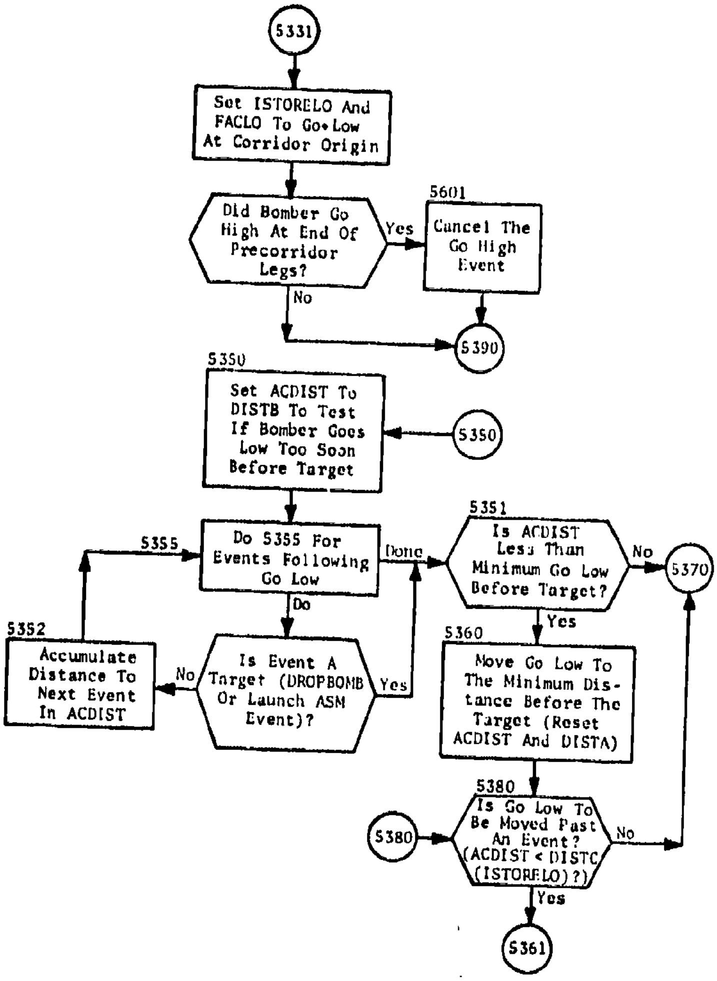


Fig. 176. (cont.)
(Sheet 2 of 7)

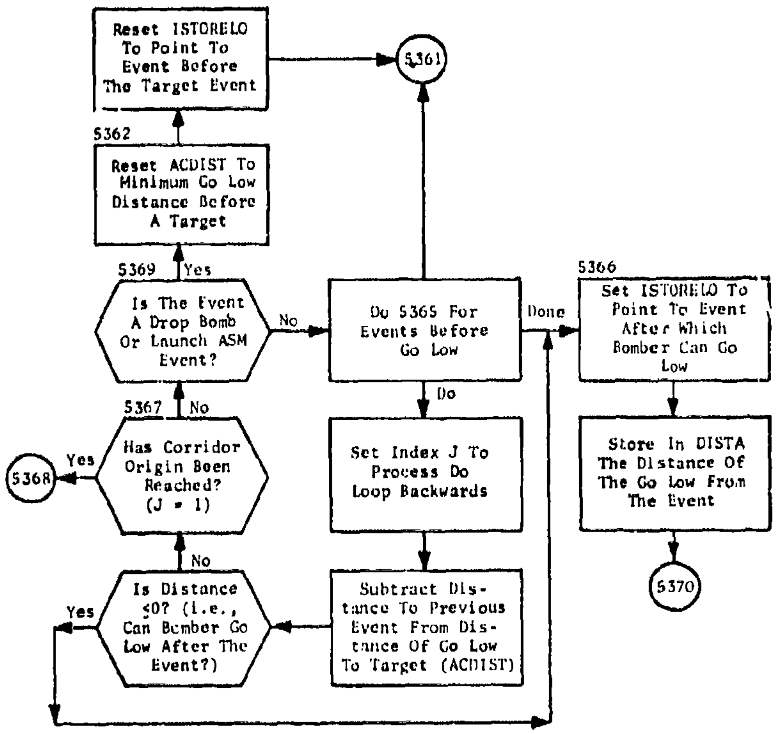


Fig. 176. (cont.)
(Sheet 3 of 7)

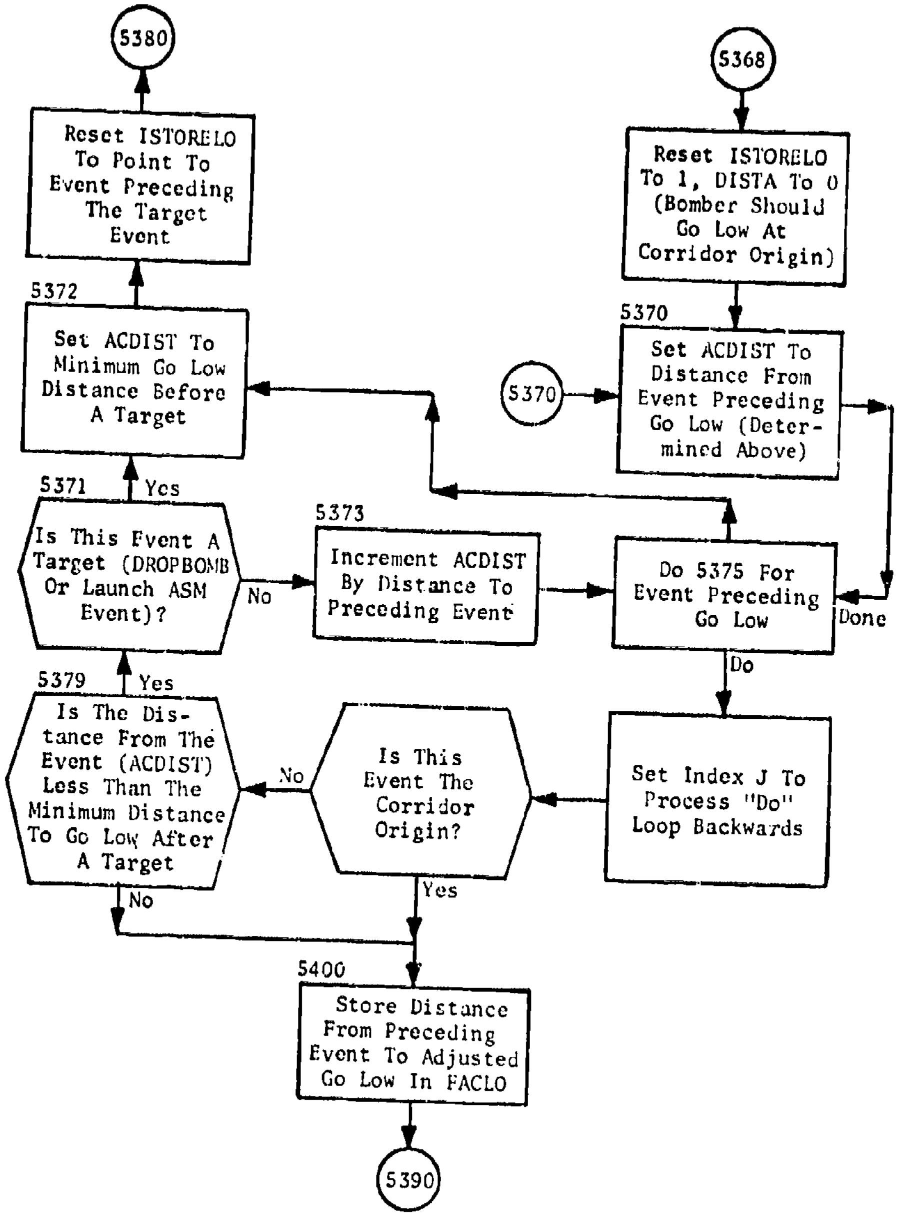


Fig. 176. (cont.)
(Sheet 4 of 7)

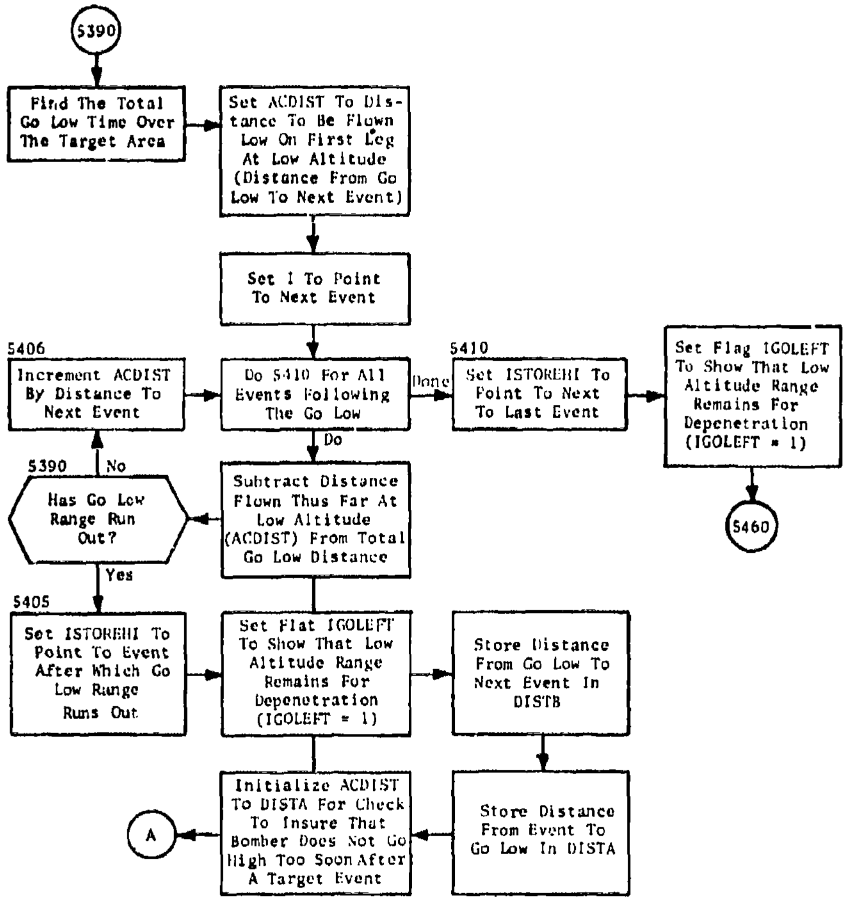


Fig. 176. (cont.)
(Sheet 5 of 7)

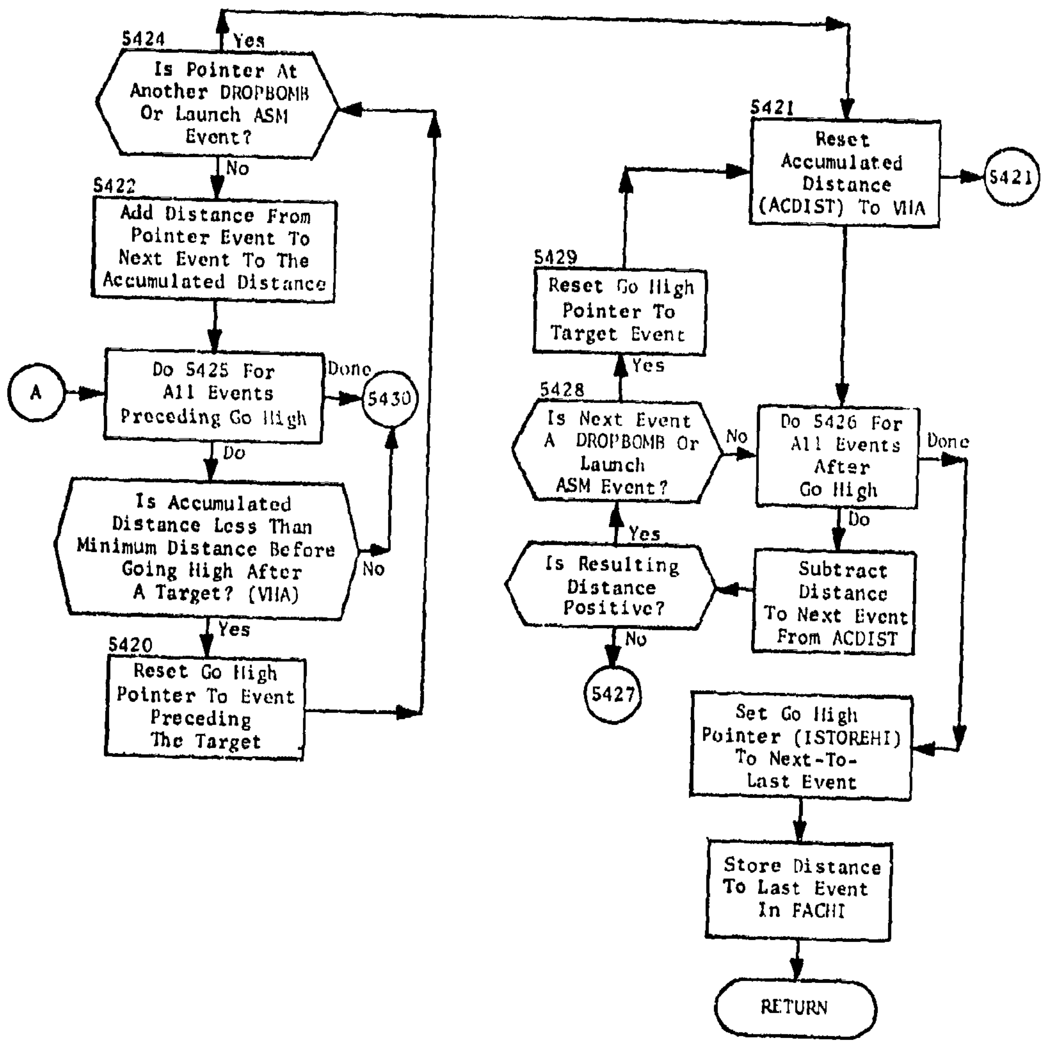


Fig. 176. (cont.)
(Sheet 6 of 7)

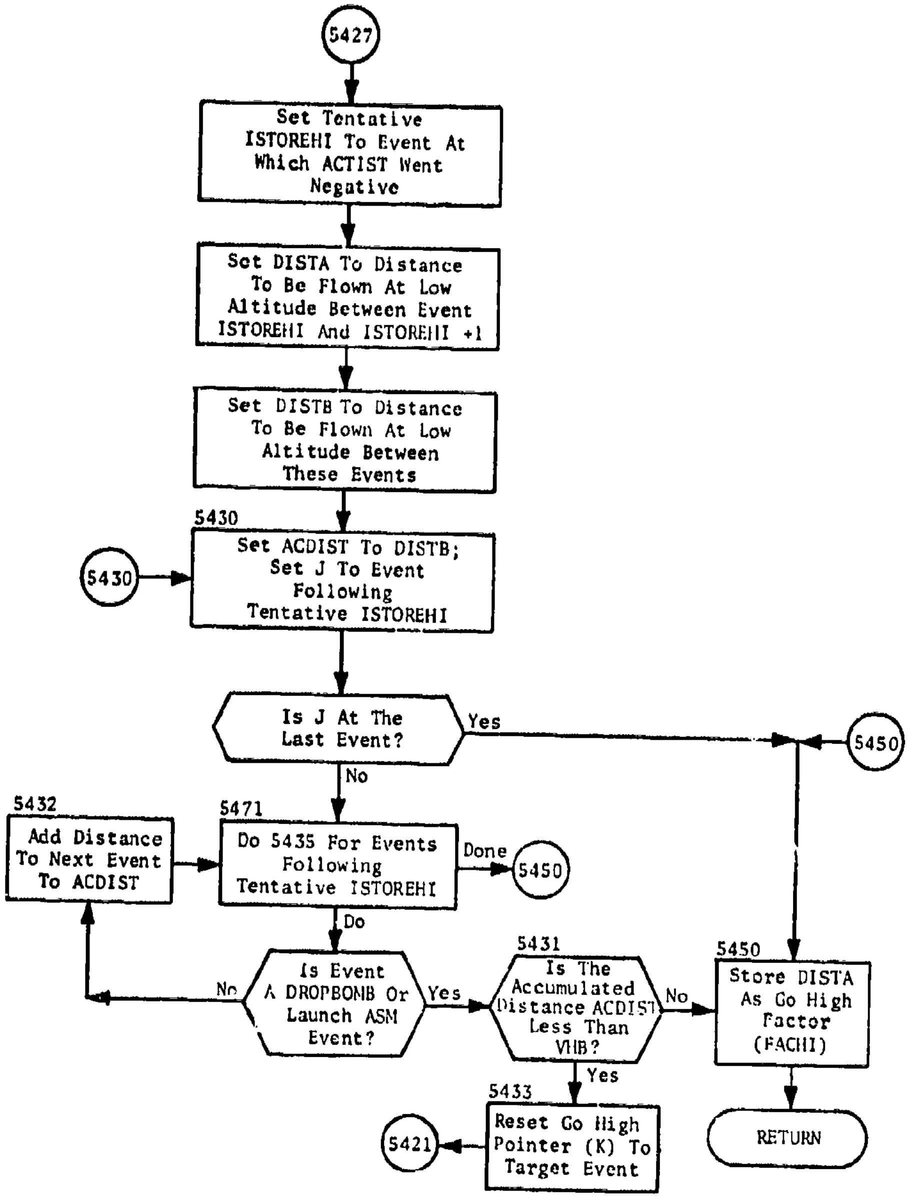


Fig. 176. (cont.)
(Sheet 7 of 7)

SUBROUTINE BOUNDARY

PURPOSE: To examine a given line segment to determine at what point, if any, it crosses a defense zone boundary.

ENTRY POINTS: BOUNDARY

FORMAL PARAMETERS: None

COMMON BLOCKS: BOUND, BOUNDPT, MASTER, 9, ITAB

SUBROUTINES CALLED: SNAPIT, ABORT

CALLED BY: PLNTPLAN, ZONECROS

Method

Both input and output of this subroutine are contained in common /BOUND/ whose variables are: X1, Y1, X2, Y2, IZN, XR, YR, NZN, IZIT. BOUNDARY is given the point (X1, Y1) in zone IZN, and the point (X2, Y2) as input. It determines if (X2, Y2) lies in a different zone from (X1, Y1). If it does, it records this zone in NZN and the crossing point (XR, YR) and sets IZIT = 1. Otherwise, it sets NZN = IZN and IZIT = 0.

Subroutine BOUNDARY also requires an input description of the zone boundaries themselves. These are contained in the arrays of common /BOUNDPT/.

It is called as needed by the subroutine ZONECROS, or in block 31 of PLNTPLAN. The method may be illustrated by referring to figure 177. A directed line segment or vector going from (X1, Y1) to (X2, Y2) is shown residing in zones 15, 27, and 62. Given that the point (X1, Y1) resides in zone 15, the crossing points p and then q must be determined. The BOUNDARY subroutine does this as follows. The zone boundary lines themselves may be looked upon as vectors, as suggested by the arrows placed on the boundary lines of zone 15. Then the vector product or cross-product may be taken between the input line segment and each of the boundary line vectors in turn. If this is done, it will be noted that the cross-product for each of the dotted boundary lines will have a different sign from the cross-product for the solid boundary lines. This simple test enables the irrelevant boundary lines to be dropped immediately from consideration. It leaves for consideration only the boundary lines (X3, Y3)

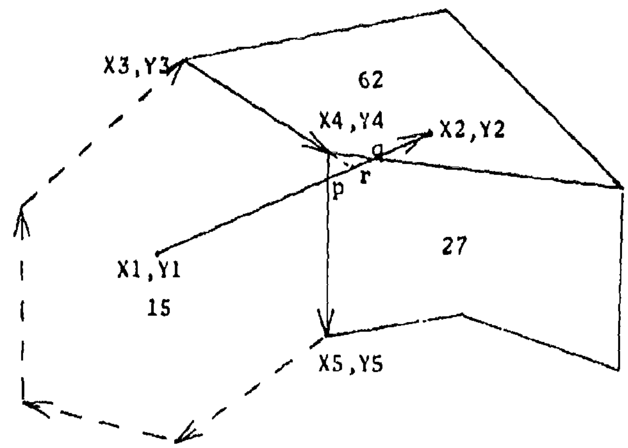


Fig. 177. Example of a Zone Crossing

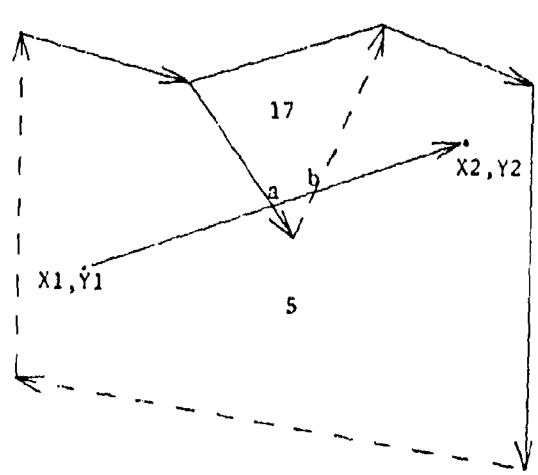


Fig. 178. Example of Crossing for a Non-Convex Zone

Point No.	BPLINK	BPZONE	NEXTZONE
1	2	5	0
2	3	5	0
3	4	5	7
4	5	5	20
5	1	5	10
6	7	7	0
7	8	7	14
8	9	7	20
9	10	7	20
10	6	7	5

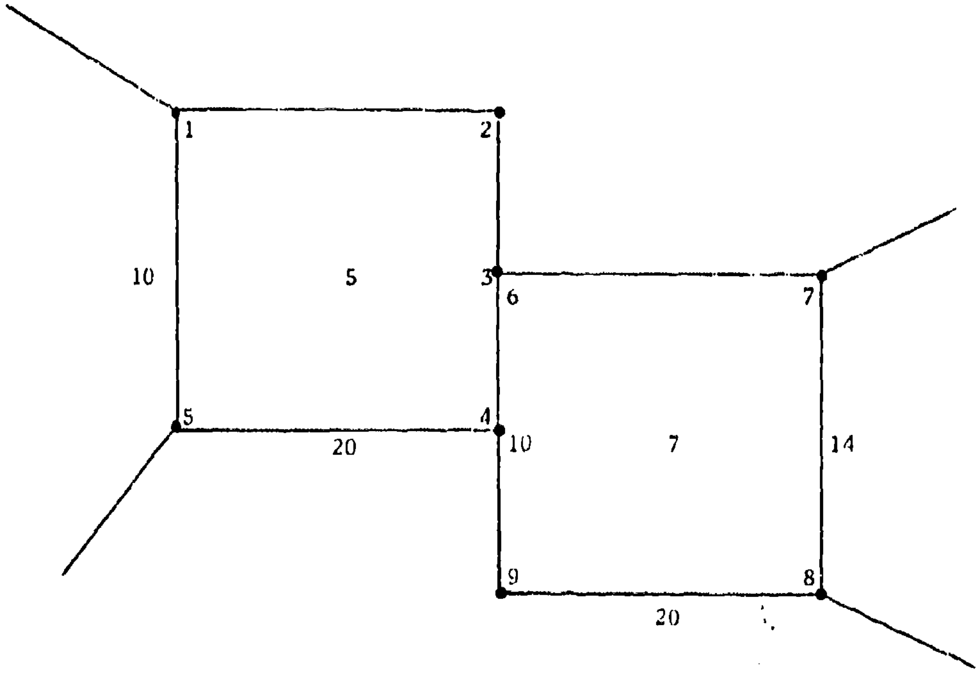


Fig. 179. Example of Zone Boundary Description

to (X4, Y4) and the line from the point (X4, Y4) to the point (X5, Y5). To determine the crossing point, use is made of the notion that any point on a line segment is some weighted average of its extreme points. Consider first the line (X3, Y3) to (X4, Y4); then the point of intersection (X,Y) between it and (X1, Y1) to (X2, Y2) may be written as follows:

$$X = \alpha X_1 + (1 - \alpha)X_2 = \beta X_3 + (1 - \beta)X_4$$

$$Y = \alpha Y_1 + (1 - \alpha)Y_2 = \beta Y_3 + (1 - \beta)Y_4$$

These equations may be solved for α and β. The cross-over point will be found at r. Since this is not on the segment from (X3, Y3) to (X4, Y4), β will be outside the range from zero to one, hence the point r will be rejected. For the line segment (X4, Y4) to (X5, Y5), however, the corresponding value β will lie in the interval 0 to 1, and so this point will be accepted and its coordinates given as (XR, YR). Also, NZN is set to zone 27. Then the segment from point p to point (X_2, Y_2) is used as the input line segment, and the BOUNDARY routine is re-entered to get the crossing at point q in figure 177.

In a situation such as shown in figure 178, the ends of the input line segment (X1, Y1) to (X2, Y2) are both in zone 5 but cross-overs occur at points a and b, into and out of zone 17. BOUNDARY again examines only the solid boundary lines of zone 5, and so produces the crossing at point a as a result. The crossing at point b is found upon entry with the line segment from a to (X2, Y2) and zone 17 specified as the current zone.

Zone boundary lines are described to this subroutine in the arrays of /BOUNDPT/ as illustrated by the example given in figure 179. This shows how zones 5 and 7 are described, and their adjacent zones indicated. Zone 5 consists of line segments joining points 1 through 5 in order and is described first. A line is indicated by a link between two points. Thus point 1 is linked to 2, point 2 to 3, and so forth, and finally, point 5 is linked back to point 1 to complete the description of zone 5. Since the segments 1-2 and 2-3 are not adjacent to another zone, NEXTZONE is recorded as zero. Segment 3-4 has the next zone 7. Segment 4-5 has the next zone 20 and so forth. Similarly, zone 7 is described by linking points 7 through 10 together and then linking point 10 back to point 6. For the purpose of this description, points are repeated; that is, points 3 and 6 are the same point and points 4 and 10 are the same point in the figure.

Figure 180 illustrates subroutine BOUNDARY.

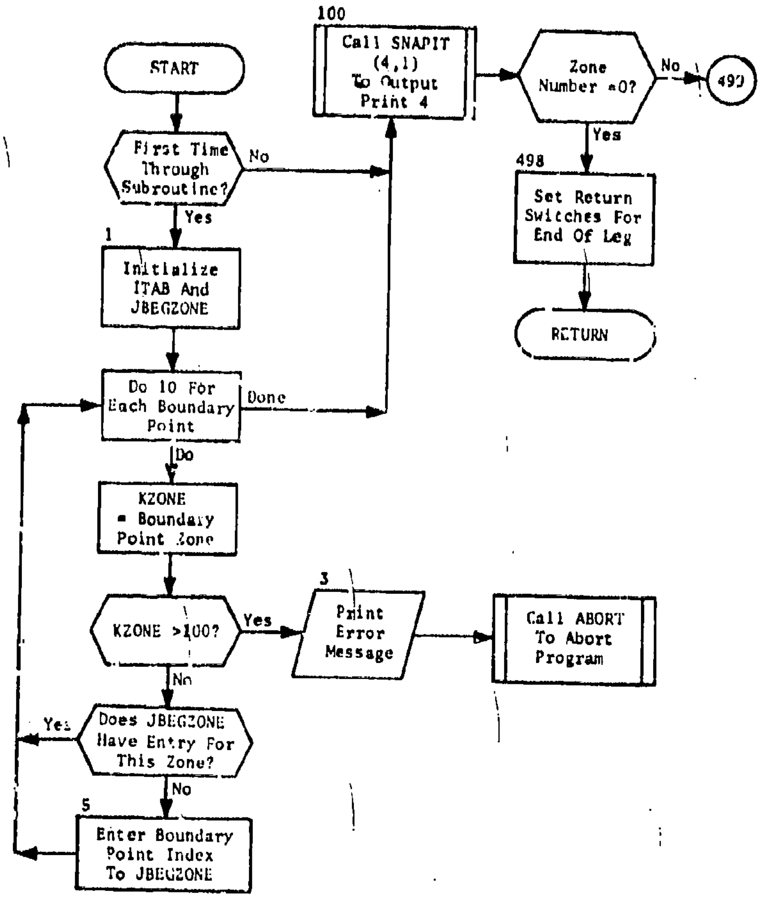


Fig. 180. Subroutine BOUNDARY (Sheet 1 of 7)

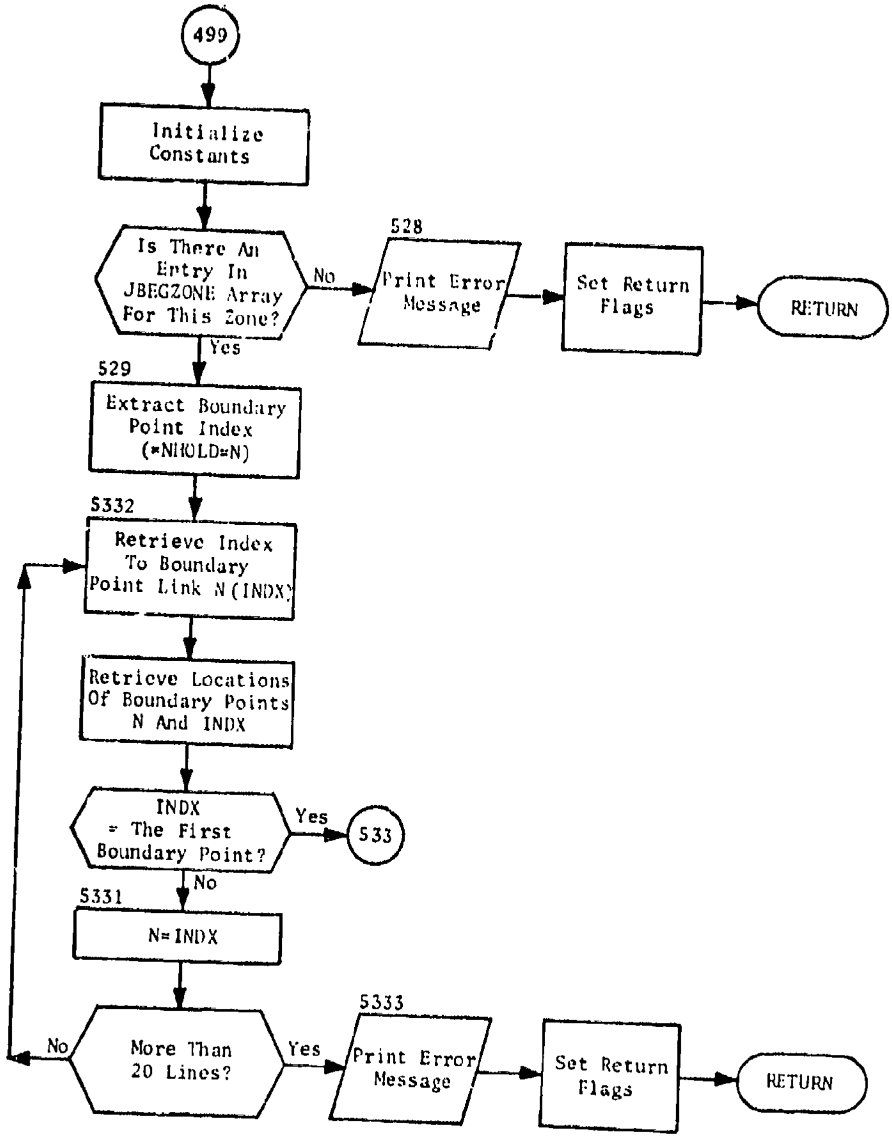


Fig. 180. (cont.)
(Sheet 2 of 7)

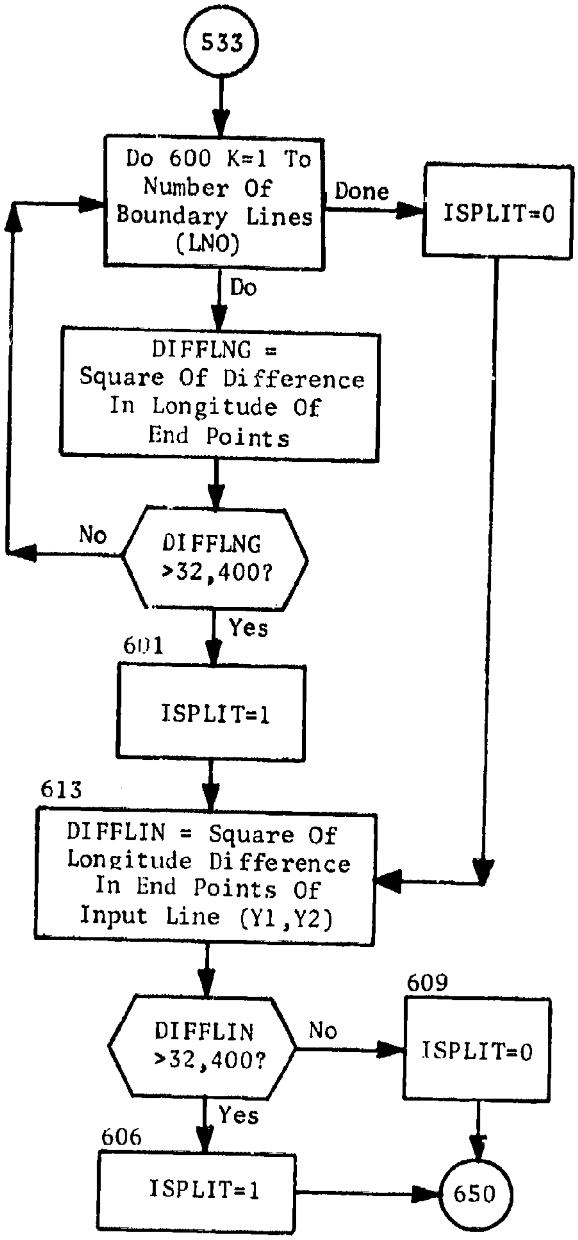


Fig. 180. (cont.)
(Sheet 3 of 7)

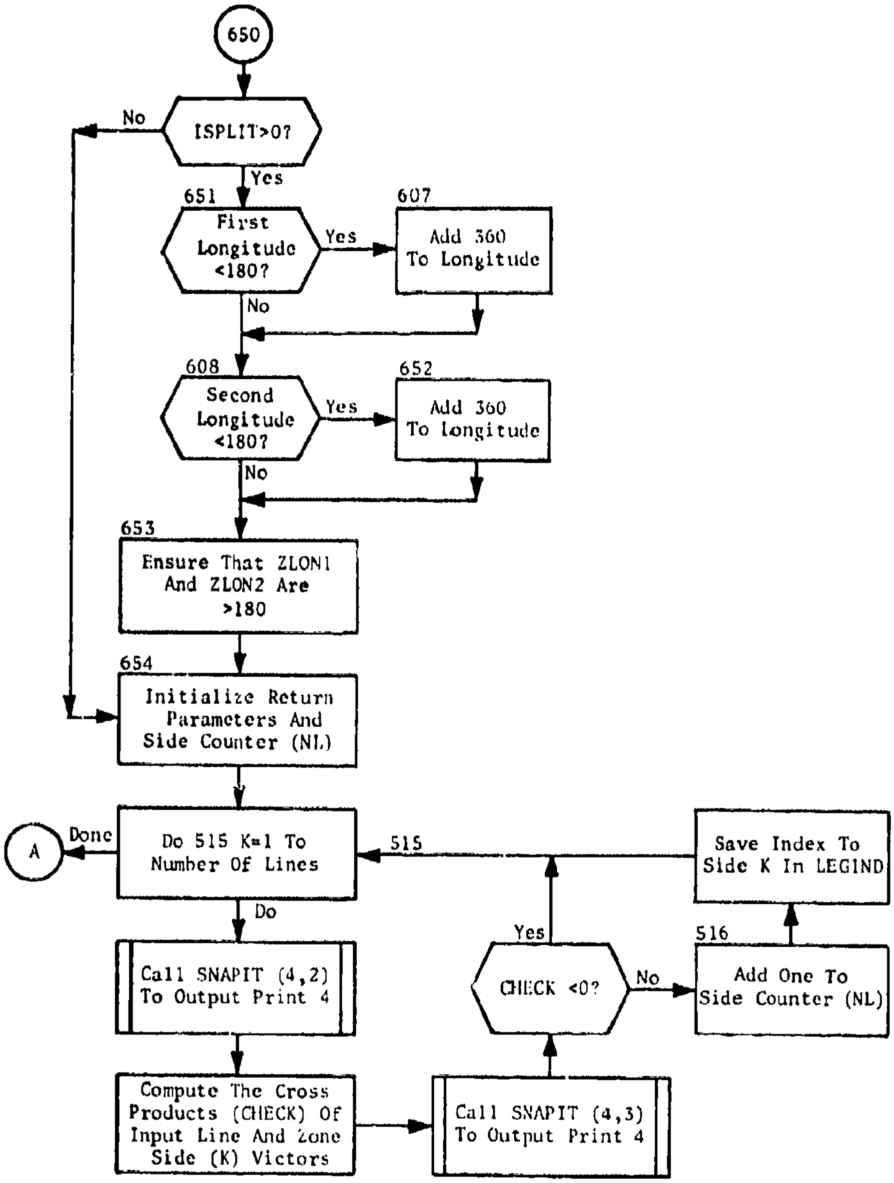


Fig. 180. (cont.)
(Sheet 4 of 7)

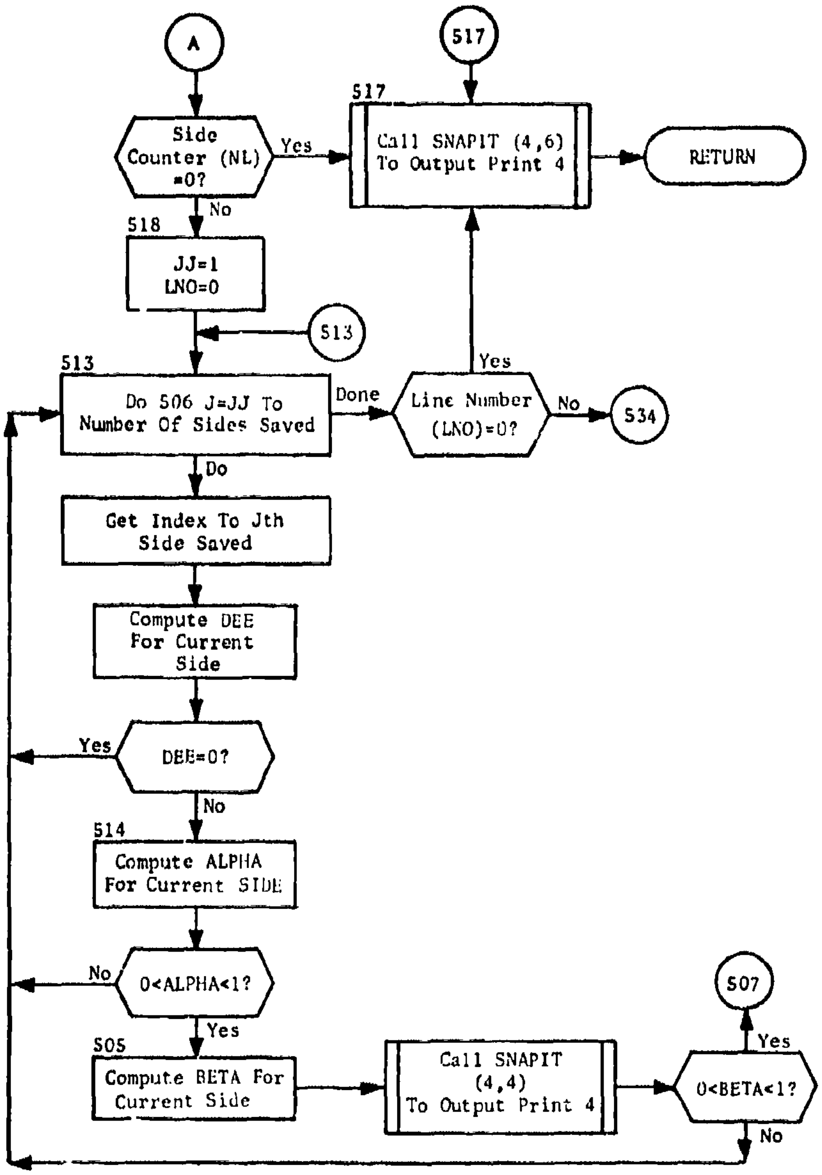


Fig. 180. (cont.)
(Sheet 5 of 7)

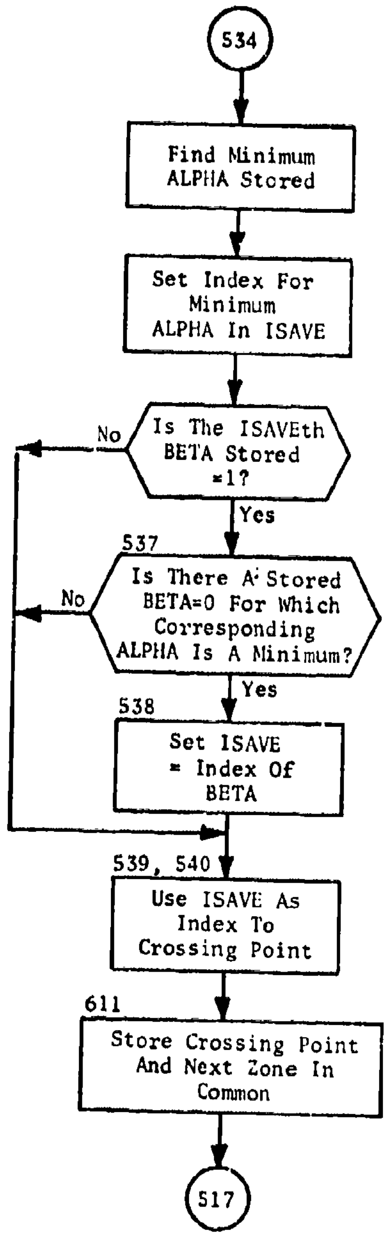


Fig. 180. (cont.)
(Sheet 6 of 7)

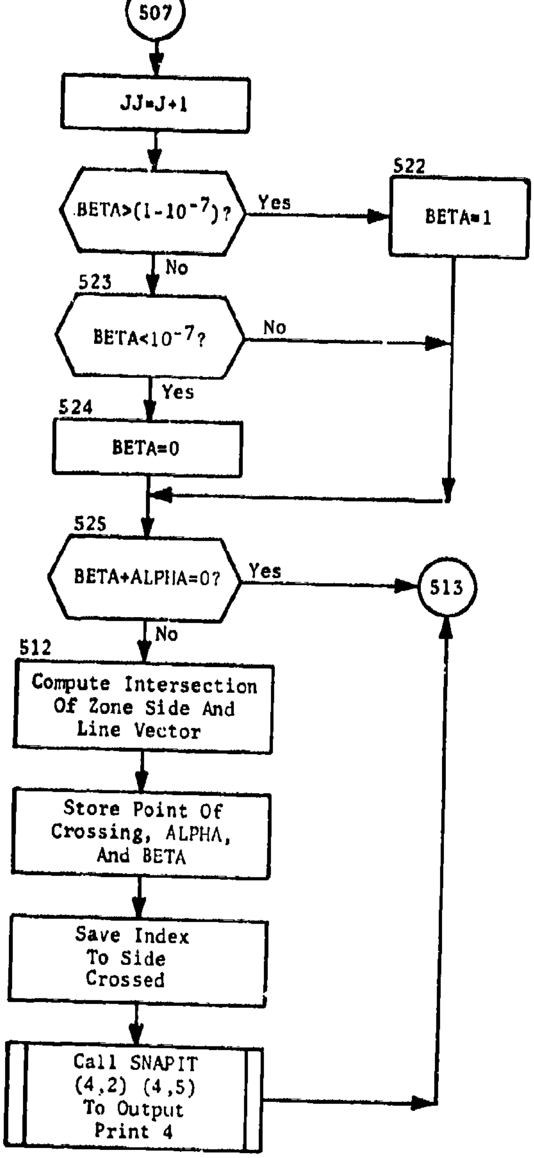


Fig. 180. (cont.)
(Sheet 7 of 7)

SUBROUTINE CHKSUM

PURPOSE:	To sum the contents of /DINDATA/ in fixed point.
ENTRY POINTS:	CHKSUM
FORMAL PARAMETERS:	I, where I = 0 means to clear, I - 1 to sum, and I = 2 to print
COMMON BLOCKS:	DINDATA
SUBROUTINES CALLED:	None
CALLED BY:	PLNTPLAN

Method

Subroutine CHKSUM will clear the checking sum, KSUM, to 0 (when the formal parameter I = 0), and print its contents (when I = 2). If I = 1, the contents of each word of the first seven arrays of common /DINDATA/ are added to KSUM.

Subroutine CHKSUM is illustrated in figure 181.

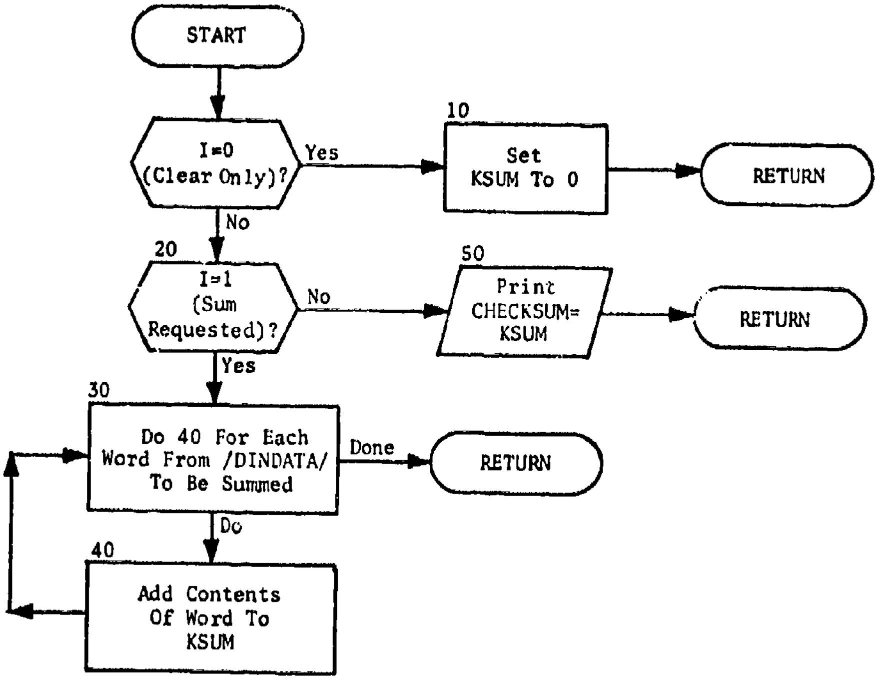


Fig. 181. Subroutine CHKSUM

SUBROUTINE CLINDATA

PURPOSE:	To initialize common /INDATA/ and common /DINDATA/
ENTRY POINTS:	CLINDATA
FORMAL PARAMETERS:	None
COMMON BLOCKS:	DINDATA, INDATA, KEYLENG
SUBROUTINES CALLED:	None
CALLED BY:	PLNTPLAN, PLANTANK

Method

Each word in common /INDATA/ and each word in /DINDATA/ is set to zero. IALT in /INDATA/ is initialized to 1.

Subroutine CLINDATA is illustrated in figure 182.

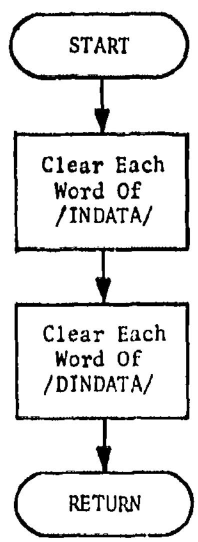


Fig. 182. Subroutine CLINDATA

SUBROUTINE DECOYADD

PURPOSE: To allocate the decoys carried by a bomber.

ENTRY POINTS: DECOYADD

FORMAL PARAMETERS: None

COMMON BLOCKS: DINDATA, DISTC, EVENTS, IOUTOLD, RL, 1, 9

SUBROUTINES CALLED: ORDER, REORDER

CALLED BY: PLNTPLAN

Method

As each bomber plan is processed by the main program, any flight situation which could use a decoy launch (see table 49) is flagged by storing the event number (MHT) of the event following the launch in array LMHT in common /1/. An associated launch priority is stored in the corresponding word of array LPRIORITY, and, for situations resulting in decoy coverage over a variable distance, the index to the array DELDIS, which contains the distance to be covered, is stored in the corresponding word of array NDCYRQ. Subroutine DECOYADD orders these arrays according to priority and allocates the available decoys in the order of this priority. The subroutine will determine only the location of decoy launches (except for launches of priority 1 or 6, for which the location must be determined as the events are processed for termination). Terminations are calculated in the main program as the decoy events are inserted into the detailed History table.

Three arrays in common /9/ are used to communicate decoy launch information to the main program. Array ILAUNDEC contains the number of decoys launched; array TIMELAUN contains the time interval between the decoy launch and the event preceding it; array DISTORE contains the distance to be covered by each decoy launch event. If this distance is greater than the range of one decoy, DECOYADD allocates sufficient decoys to cover the entire distance. It is assumed that another decoy is launched as soon as the previous decoy terminates. However, only one launch event is posted for the entire coverage distance.

Since the bomber must launch all decoys, more than one decoy may be launched at a time if the priority list has been satisfied before all decoys have been allocated. In the case of area coverage, there may not be sufficient decoys remaining to cover the distance of the first allocation. Hence, an entry is made in array DISTORE each time a decoy launch event occurs over the same area. If the last decoy does not cover the same distance as the previous decoy(s), two decoy termination events must be posted for the one launch event.

Each time a decoy is allocated, the index to the detailed History table (MHT) is incremented to reserve a line for each event generated by the launch. Since a decoy launched at low altitude (priorities 1 and 6) will always terminate at its target, no termination event is necessary. Hence space is reserved only for the launch event. (This situation is communicated to the termination section by storing the number of decoys launched as a negative number.) For high-altitude launches, either one or two termination events are required in addition to the launch event.

The decoys are allocated by processing each entry in the priority array in order. Since the calculation of timing and distance information differs according to the launch situation, branches are made to various sections of the program according to priority. It should be noted that since the priority 3 launch information is sent to the subroutine as the first instance of a priority 5 launch, the priority 4 launch will be encountered before the priority 3 decoy has, in fact, been allocated. Thus, before the section for priority 4 is processed, a check is made to insure that more than one decoy remains to be allocated. If only one remains, the priority 4 section is skipped, reserving that decoy for the first priority 5 launch (i.e., priority 3).

Since the priority 8 situation calls for the decoy(s) to be launched immediately after the priority 4 decoys to cover the high altitude flight until a go low or a depenetration, the launch event is omitted from the detailed History table and the distance to be covered by the decoys is added onto the distance to be covered by the priority 4 decoys. This merely moves the priority 4 termination event(s) to include the distance that would be covered by the priority 8 decoy(s).

If decoys remain after every entry in the priority array has been processed for the first time, the array will be reprocessed, in order, to provide double coverage. Since many of the values calculated on the first pass need not be recalculated, a different set of branches is taken, according, again, to priority. Up to six allocation passes will be made, as long as decoys remain. If more than six are required to allocate all the decoys, the error message

> NUMBER OF DECOY LAUNCHES EXCEEDS CAPACITY OF DECOY
> ALLOCATION

is generated. Whenever this occurs, or whenever no more decoys remain to be allocated, control returns to the main program. Subroutine DECOYADD is illustrated by figure 183.

462-548 O - 72 - 30

Table 49. Launch Priority

LAUNCH PRIORITY	CIRCUMSTANCE OF LAUNCH
1	R_L* miles before first low-altitude gravity bomb attack on a SAM-defended target
2	Immediately before changing from high to low altitude
3	Immediately before penetrating defended airspace if flying at high altitude
4	R_H** miles before first high altitude gravity bomb attack on a SAM-defended target
5	Coverage when flying at high altitude over defended airspace before priority 4 launch
6	R_L miles before subsequent low-altitude gravity bomb attacks on SAM-defended targets
7-8***	Coverage when flying at high altitude over defended airspace after priority 4 launch

*R_L= range of decoy at low altitude (data set to 200 nautical miles)

**R_H= range of decoy at high altitude (data set to 400 nautical miles)

***Priority 8 is used if the coverage is to begin at the point where the priority 4 decoy terminates. Priority 7 is used if the bomber has changed altitude between the priority 4 and the priority 7 launch.

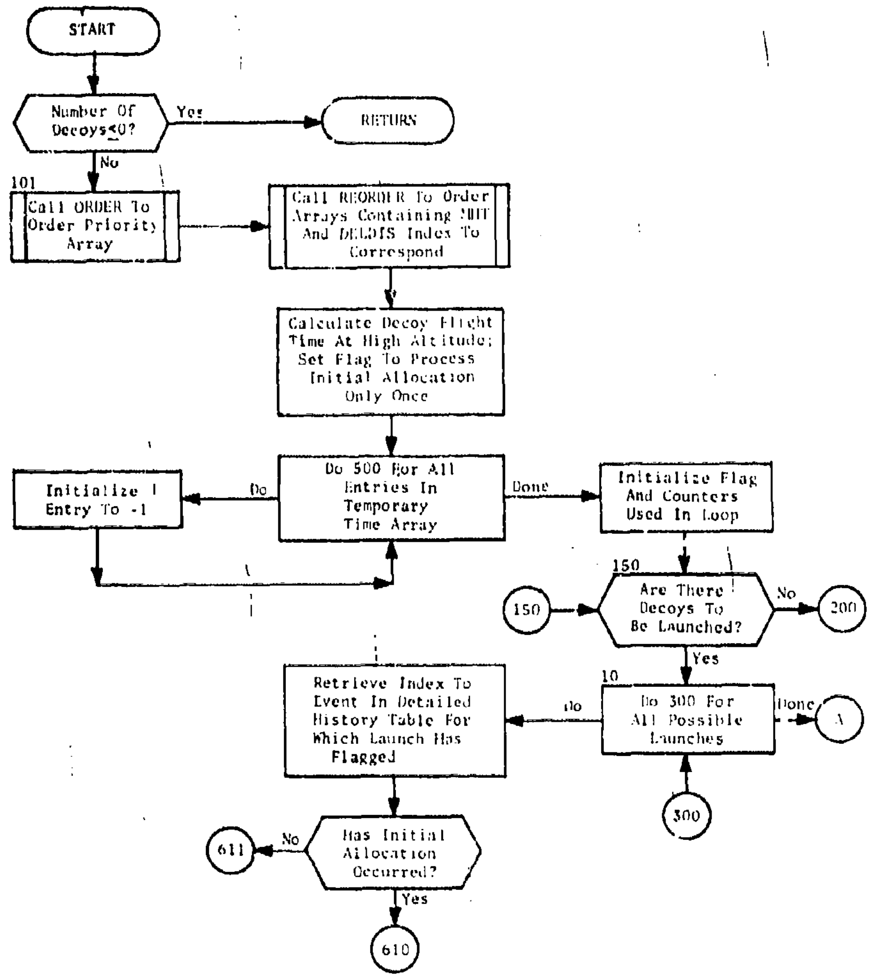


Fig. 183. Subroutine DECOYADD
(Sheet 1 of 7)

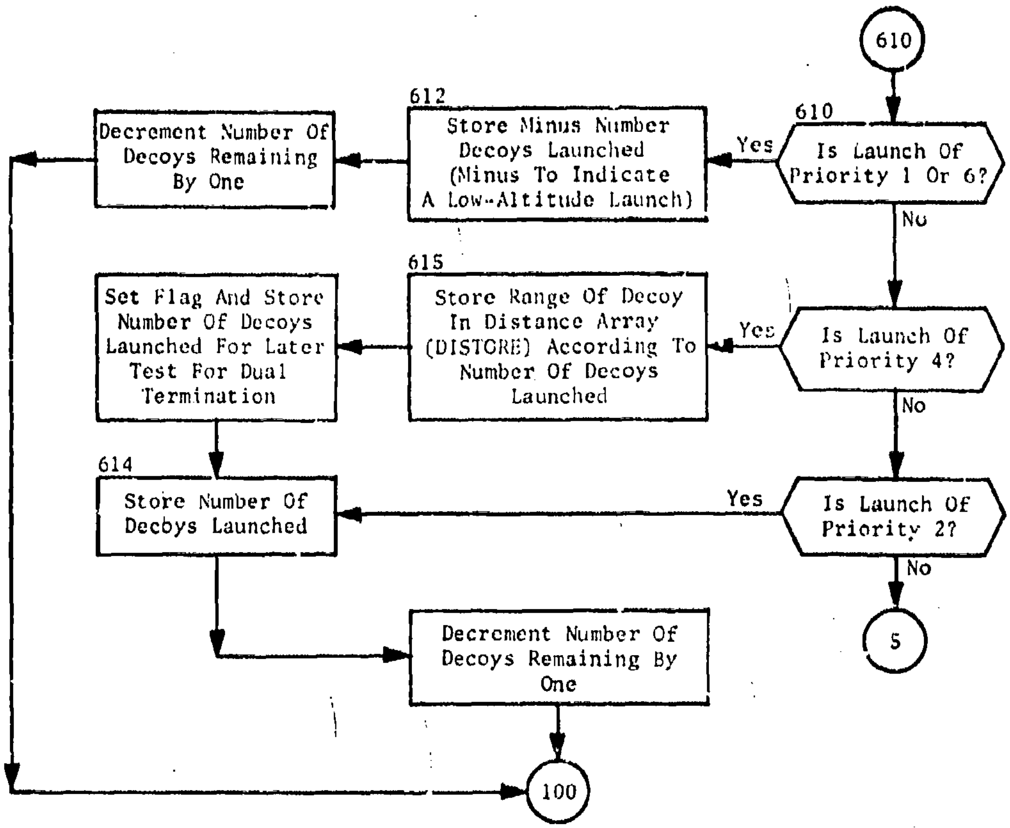


Fig. 183. (cont.)
(Sheet 2 of 7)

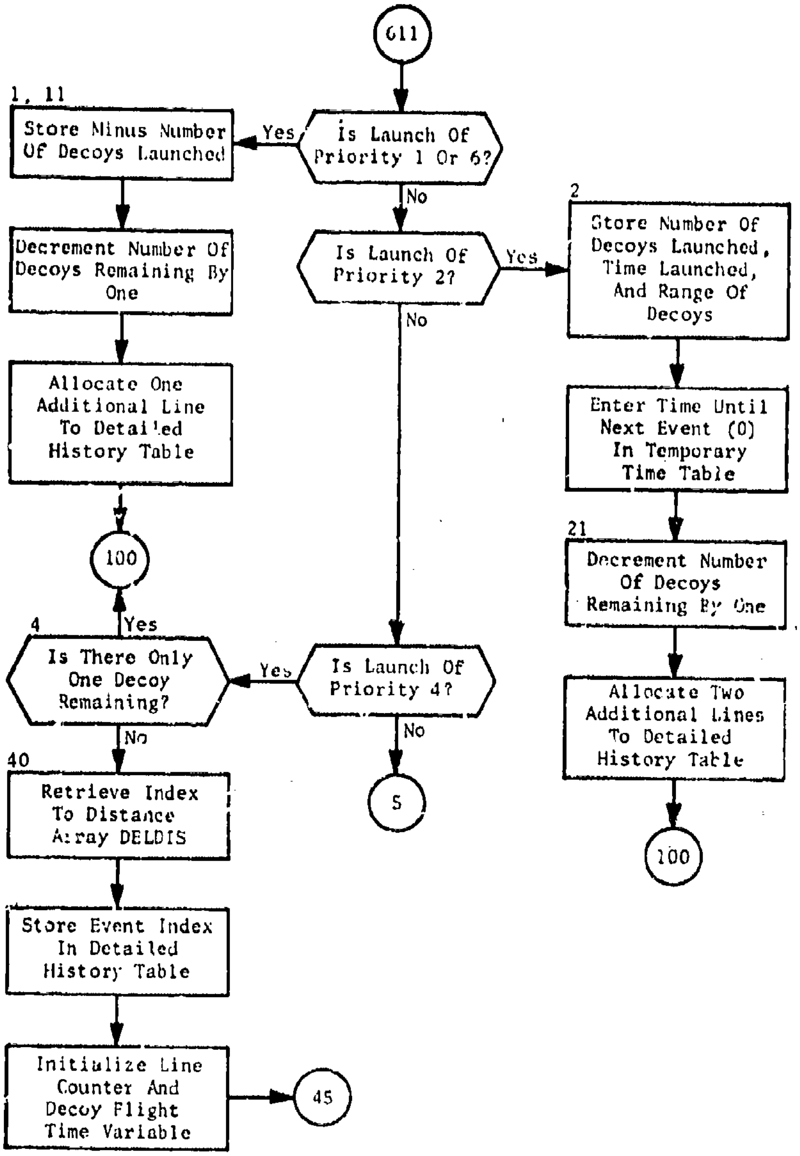


Fig. 183. (cont.)
(Sheet 3 of 7)

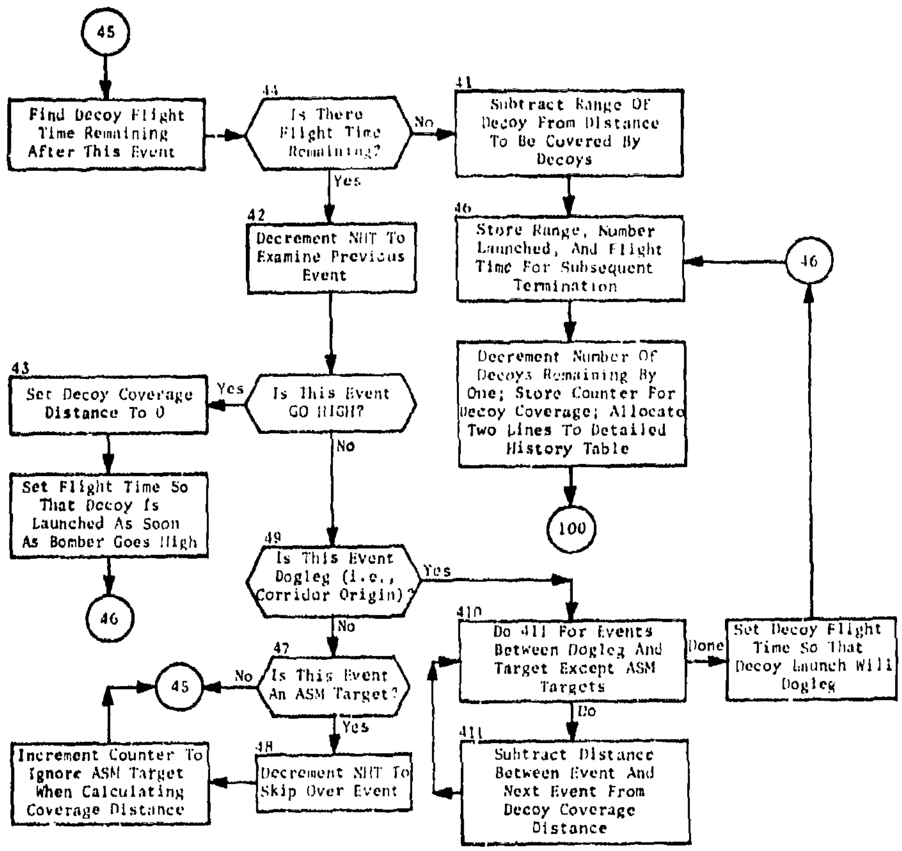


Fig. 183. (cont.)
(Sheet 4 of 7)

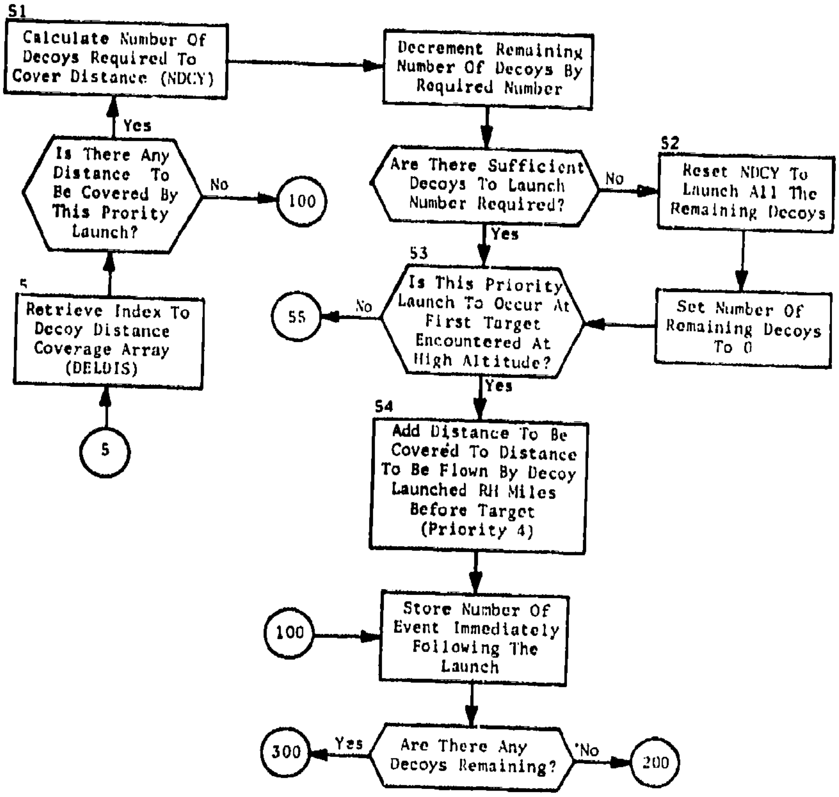


Fig. 183. (cont.)
(Sheet 5 of 7)

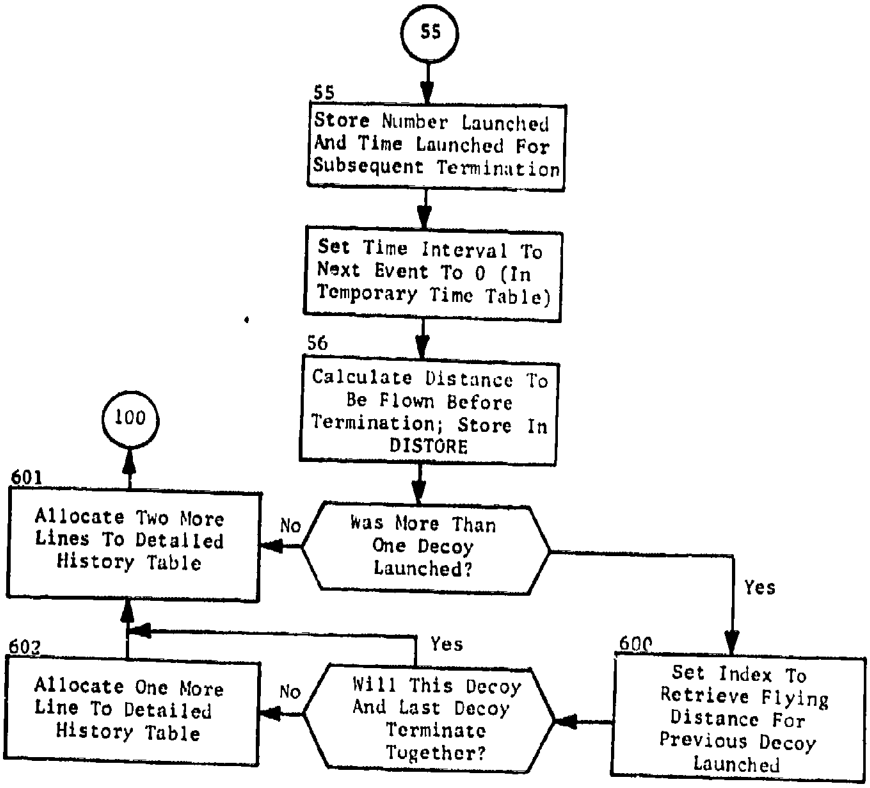


Fig. 183. (cont.)
(Sheet 6 of 7)

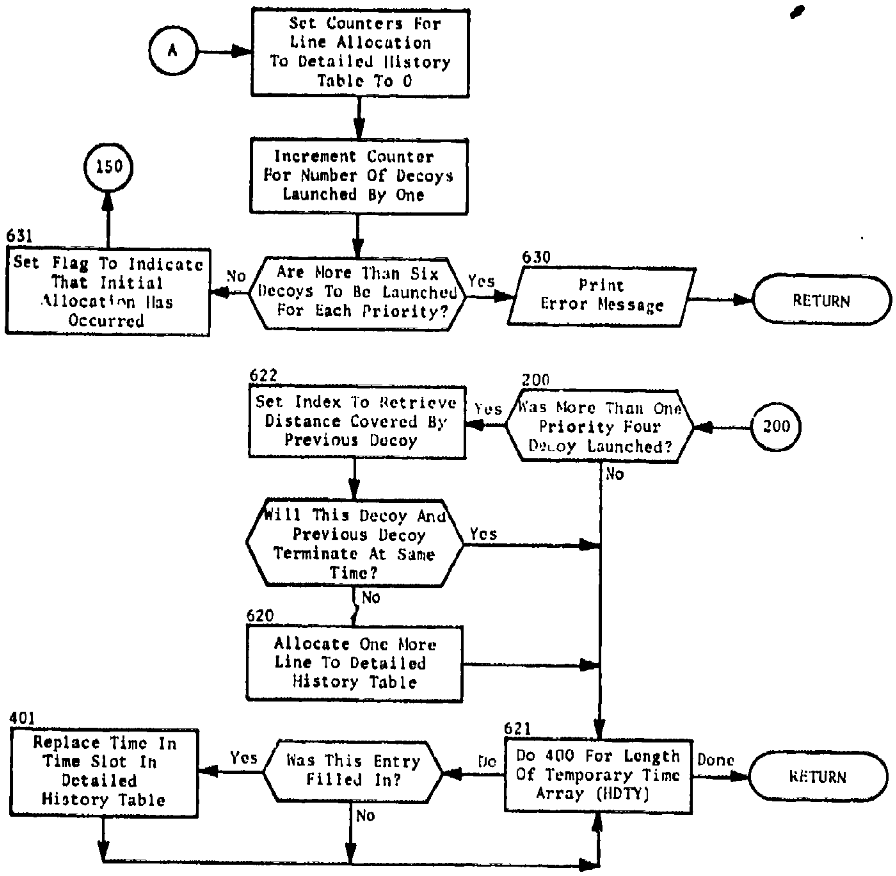


Fig. 183. (cont.)
(Sheet 7 of 7)

SUBROUTINE DISTIME

PURPOSE.	To compute distances between events and convert these distances into time increments.
ENTRY POINTS:	DISTIME
FORMAL PARAMETERS:	None
COMMON BLOCKS:	ASMTABLE, DINDATA, DINDATA2, EVENTS, IOUTOLD, OUTSRT, PAYLOAD, SPASM
SUBROUTINES CALLED:	DISTF
CALLED BY:	PLNTPLAN

Method

This subroutine is called immediately after a bomber plan or an alternate bomber plan has been processed. In either case, the plan is contained in the arrays of common /DINDATA/. The counter MHT is set to the number of lines contained in the plan. For primary plans, counter NPL is set to zero. An alternate plan is located beginning at cell MHT.

DISTIME uses the latitudes contained in array HLA and the longitudes contained in the array HLO, and records the resulting time increments in array HDT. For computing distances, the function DISTF is used. This computes great circle distances where the longitudinal difference is greater than 2.8 degrees, otherwise it assumes a Mercator projection. To convert distances to time, the speed SPDHI or SPDLO is used, depending on whether the bomber is at high or low altitude. For ASMs, the value in SPASM is used.

This computation is sufficient for all events except for zone crossings. This is because zone crossings are located or determined only approximately on a Mercator projection. The adjustment to the distances in the case of zone crossings may be described by the illustration in figure 184. This shows the two zone crossing events Z_1 and Z_2 located between events E_1 and E_2. The distances between events are d_1, d_2, and d_3 as indicated. The great circle distance between E_1 and E_2 is D. In this case the distance d_1 would be replaced by $d'_1 = d_1D'$ where $D' = D/(d_1 + d_2 + d_3)$. Similarly $d'_2 = d_2D'$ and $d'_3 = d_3D'$.

Subroutine DISTIME is illustrated in figure 185.

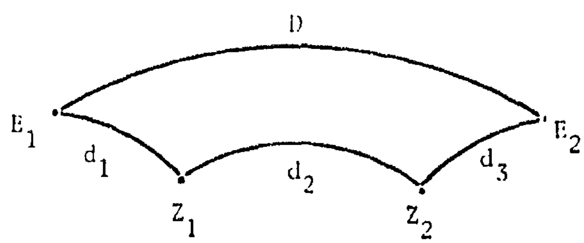


Fig. 184. Distance Adjustments For Zone Crossings

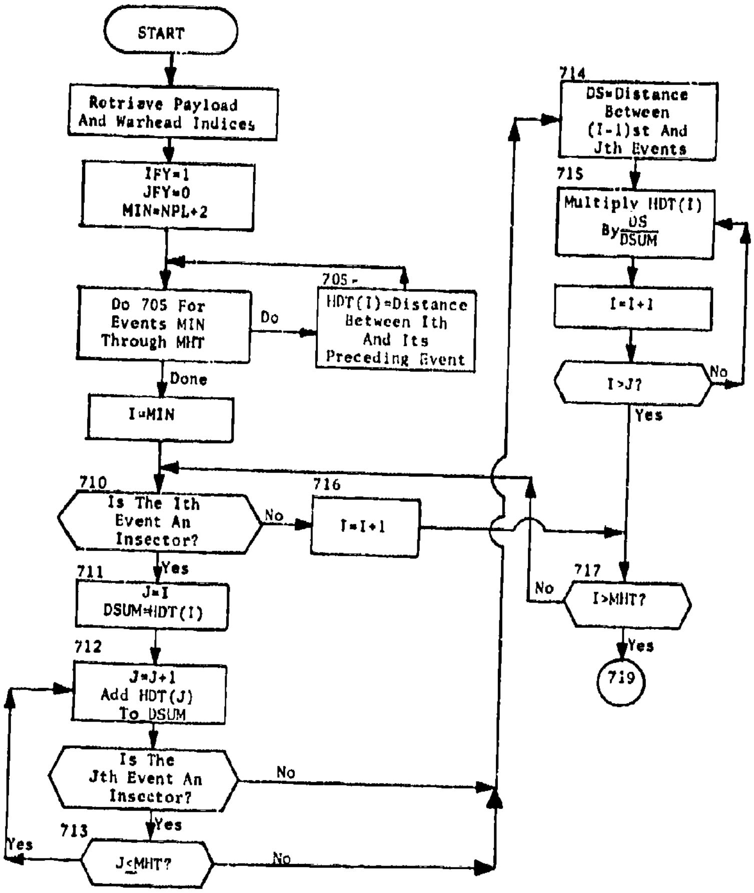


Fig. 185. Subroutine DISTIME
(Sheet 1 of 2)

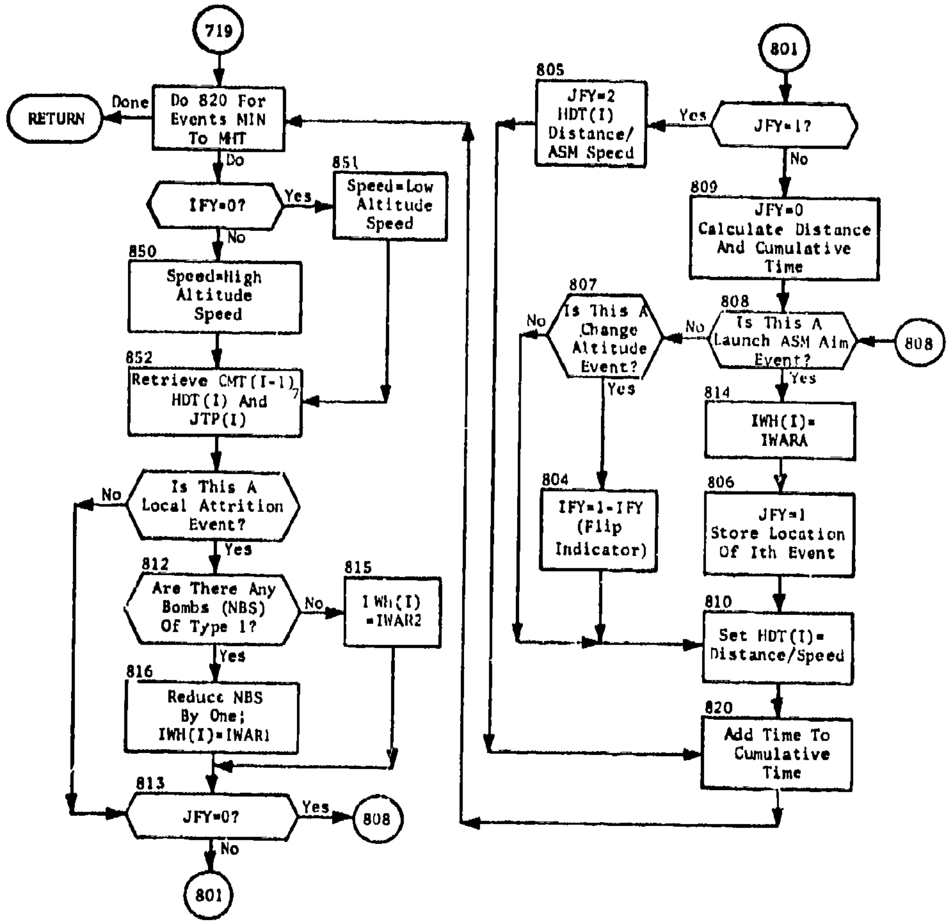


Fig. 185. (cont.)
(Sheet 2 of 2)

SUBROUTINE FINDZONE

PURPOSE:	To find the closed polygon (zone) that contains an arbitrary point (X, Y).
ENTRY POINTS:	FINDZONE
FORMAL PARAMETERS:	None
COMMON BLOCKS:	FINDZONE, ITAB, MASTER, 9
SUBROUTINES CALLED:	DIFFLONG, ORDER, REORDER
CALLED BY:	PLNTPLAN, ZONECROS

Method

The routine first sorts the zone data so that the zones are in numerical order. Then the point to be tested (FINDLON, FINDLAT), equivalenced to (X, Y), is checked to see whether it is inside any described zone. If it is, IFOUNDZN is returned with the zone number; if not, IFOUNDZN is returned with zero value.

The mathematical algorithm is based on the theorem that given any polygon and a point, if the sum of the directed angles between the point and successive vertices of the figure is zero the point is outside the polygon. If the sum is >0 the point is inside the figure. Since in this case the coordinates are described by longitude and latitude, adjustment must be made for the circular nature of the longitudinal scale. The first point of the zone is chosen as the reference point and the longitudinal difference between this point and every other point of the zone is used to determine the size and direction of the enclosed angles. This method assumes that no one zone has more than a 180° difference in longitude between any two points.

Subroutine FINDZONE is illustrated in figure 186.

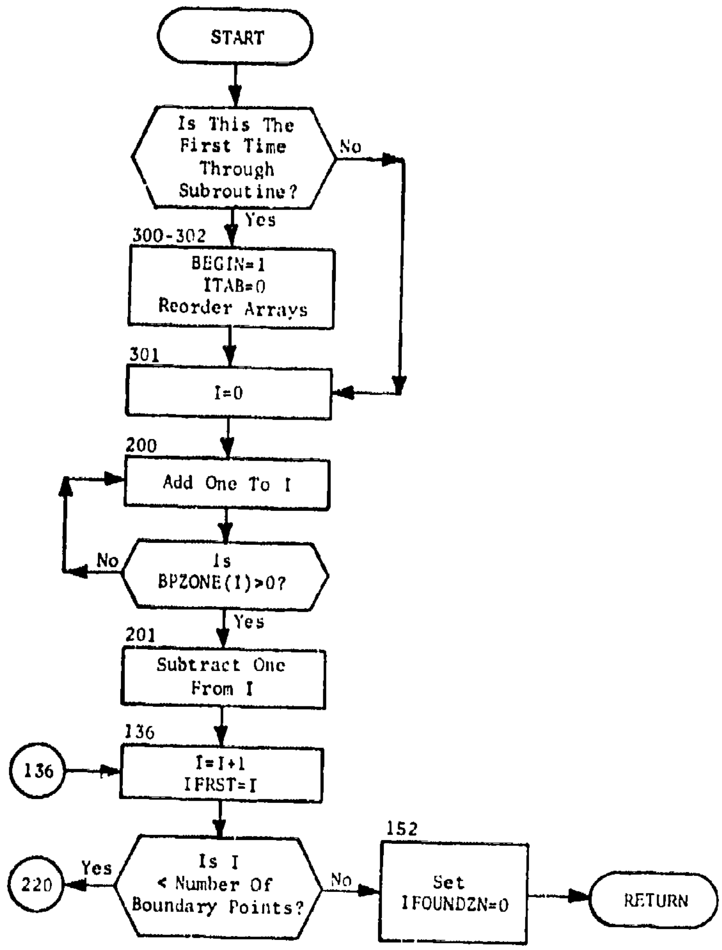


Fig. 186. Subroutine FINDZONE
(Sheet 1 of 3)

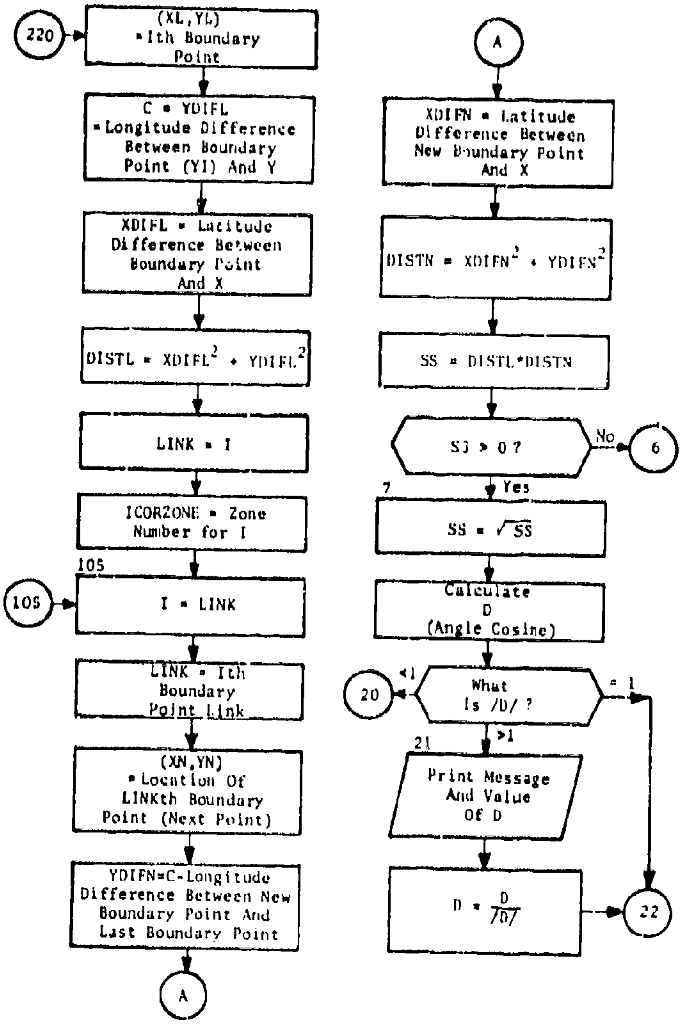


Fig. 186. (cont.)
(Sheet 2 of 3)

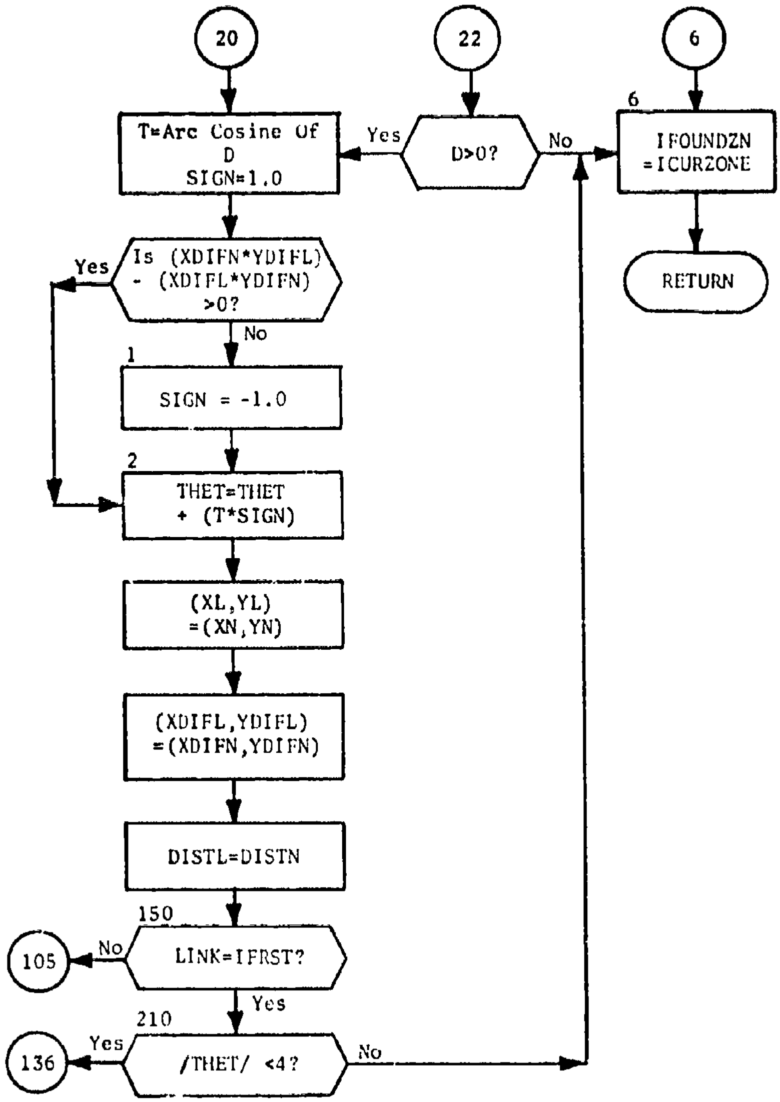


Fig. 186. (cont.)
(Sheet 3 of 3)

462-546 O - 72 - 31

SUBROUTINE FLYPOINT

PURPOSE:	FLYPOINT is an integral part of block 40 in PLNTPLAN which adjusts events for ASM launches.
ENTRY POINTS:	FLYPOINT, PREFLY1, PREFLY2, POSTFLY
FORMAL PARAMETERS:	None
COMMON BLOCKS:	ASMARRAY, LASM, OUTSRT
SUBROUTINES CALLED:	DISTF, LAUNCH
CALLED BY:	PLNTPLAN

Method

PREFLY1 determines the distance between the ASM target and the previous flypoint which was not an AIM ASM event.

PREFLY2 calculates the distance between the ASM target and the previous flypoint.

POSTFLY finds the next flypoint, and calls subroutine LAUNCH to compute the ASM launch point.

Subroutine FLYPOINT is illustrated in figure 187.

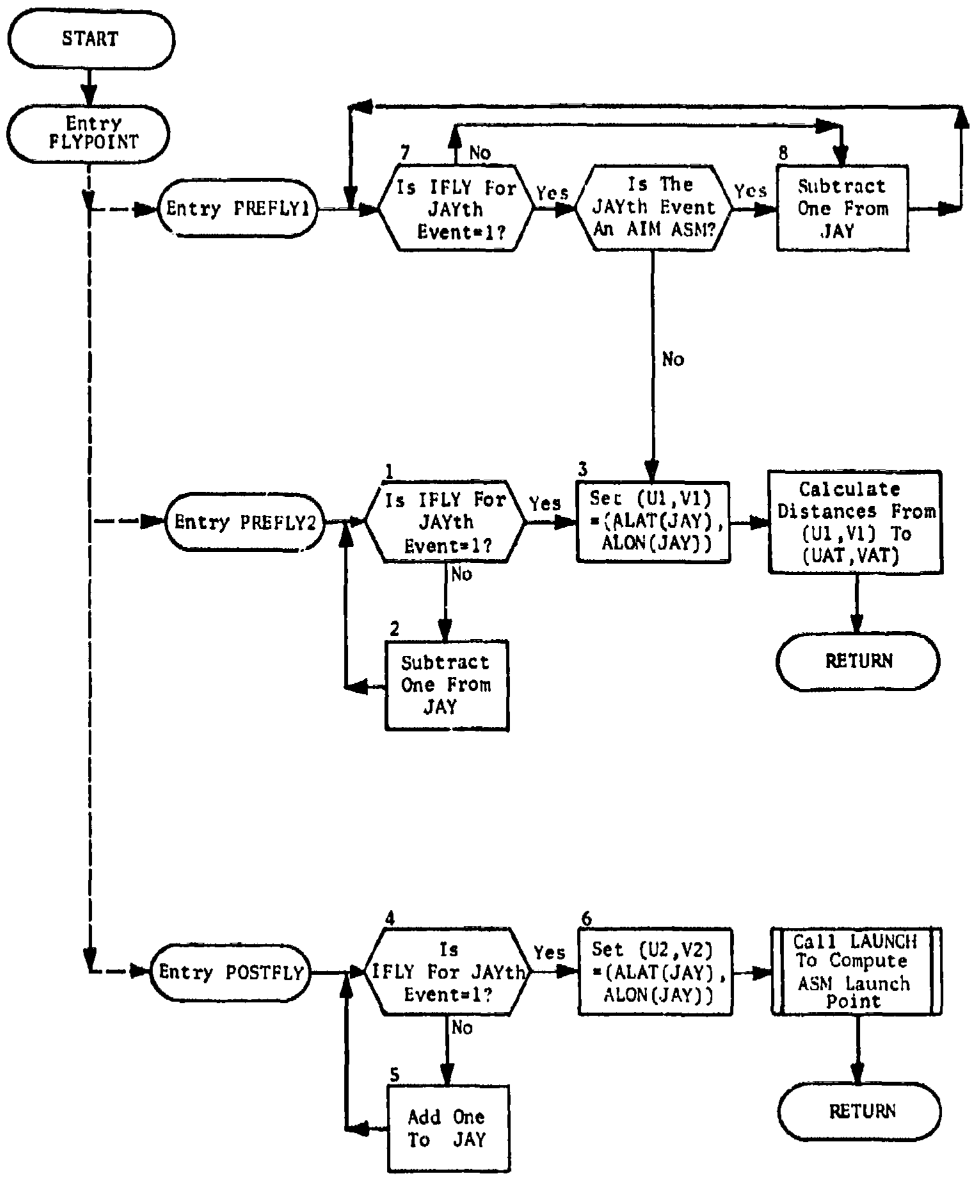


Fig. 187. Subroutine FLYPOINT

SUBROUTINE INITANK

PURPOSE: To read and store tanker data from the BASFILE.

ENTRY POINTS: INITANK

FORMAL PARAMETERS: None

COMMON BLOCKS: FILES, IRF, KEYLENG, MASTER, REF, TANKER, 7, 9

SUBROUTINES CALLED: RDARRAY

CALLED BY: PLNTPLAN

Method

After PLNTPLAN has read and stored all other necessary information from the BASFILE, it calls INITANK to complete the task by reading each tanker base record and storing the elements required by PLNTPLAN.

Subroutine INITANK is illustrated in figure 188.

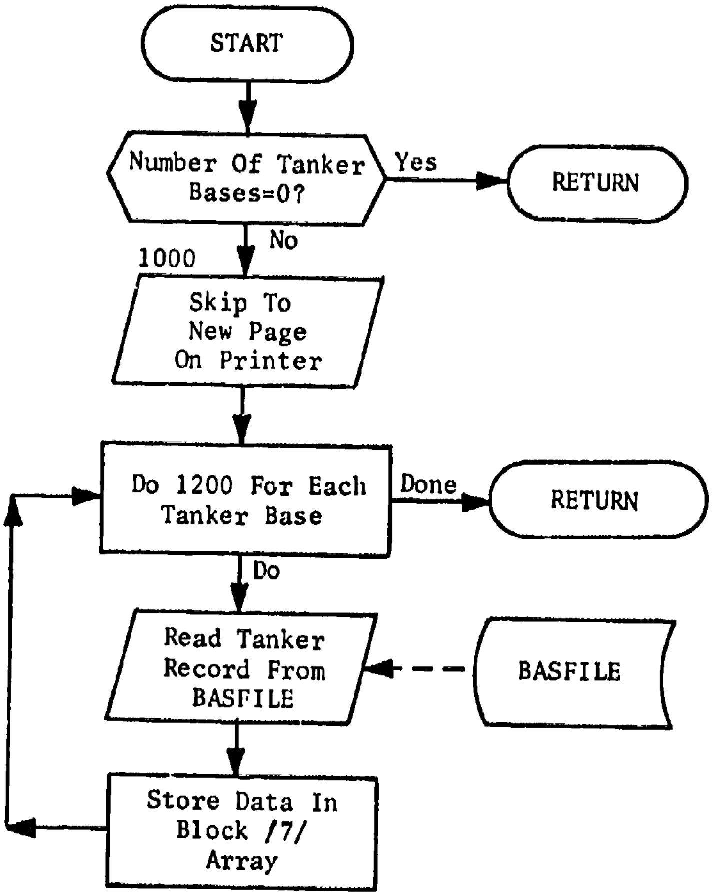


Fig. 188. Subroutine INITANK

SUBROUTINE LAUNCH

PURPOSE:	To determine the aim point, or launch point, at which an ASM is to be fired.
ENTRY POINTS:	LAUNCH
FORMAL PARAMETERS:	None
COMMON BLOCKS:	LASM, LAUNSNAP
SUBROUTINES CALLED:	DIFFLONG, SNAPIT
CALLED BY:	FLYPOINT (entry POSTFLY)

Method

This subroutine is called by the subroutine FLYPOINT whenever the aim point for an ASM is required. The inputs and outputs to this subroutine are all contained in /LASM/ whose variables are the following: U1, V1, U2, V2, UAT, VAT, RASM, RLAT, and RLONG. The subroutine is given that a bomber is flying from the point (U1, V1) to the point (U2, V2) and that it is to fire an ASM of range R at the target (UAT, VAT) during this flight, at maximum range if possible. It determines the point (RLAT, RLONG) at which the ASM is to be fired. The value for R is stored in RASM.

This description refers to figure 189. Here again the bomber is assumed to fly from point (U1, V1) to point (U2, V2) and the point (RLAT, RLONG) at which the ASM is to be fired from maximum range R to a target located at (UAT, VAT) is to be determined. Two cases occur. In the first and simpler of the two, the range of the ASM is sufficient so that it may be launched while the bomber is proceeding in a straight line path from (U1, V1) to (U2, V2). This would be the case if the range of the ASM were R' shown in figure 189. The ASM target is said to be "in range". The ASM could be launched at maximum range from either point p or point p' shown in the figure. Of course, point p would be chosen. Since point p is a point en route, it is not considered to be a FLYPOINT. The second and more interesting case occurs when the range of the ASM is equal to R as shown. Here the bomber must deviate from its course and fly to the point p" to fire the ASM. The ASM target is said to be "out of range", and the point p" is now a fly point. A solution in this case is to divide the angle $\theta = \theta_1 + \theta_2$ into its components parts in the same proportion as D1 and D2 shown in the figure, that is:

$$\theta_1 = \theta D_1/D_1+D_2) \qquad \theta_2 = \theta D_2/(D_1+D_2)$$

This was found to give a good solution in most cases.. However, a better solution is obtained by using a circle with radius .75 x R instead of R, and is described below.

The procedure carried out by the LAUNCH subroutine is outlined with the help of figure 190. The origin of the coordinate system is taken at (U1, V1), and a Mercator projection is used with the longitude corrected to the ASM target located at the point (B1, A1), by multiplying it by α=cos B1. The distance R1 to the ASM target from this origin is first checked against the range R of the ASM. If it is less the ASM is fired from the origin. This occurs if the ASM target is in the circular region about the origin. Otherwise the distance P1 is computed as

$$P1 = B1 * \cos\alpha - A1 * \sin\alpha = (B1 * A - A1 *B)/P$$

and compared against R. If P1 < R then the ASM target is in range of the (possibly projected) flight path. Then, provided it is also in the half-circular region about the point (U2, V2), the aim point is on the line from the origin to this point. The distance F is obtained by first obtaining

$$P2 = A1*\cos\alpha + B1 * \sin\alpha = (A1 *A+ B1*B)/P$$

and subtracting from it the distance x which is obtained by quadratic solution.

If the ASM target is not in range of the flight path, the aim point is computed as follows (see figure 191). First, the point (BT, AT) is computed by:

$$BT = B*D1/(D1 + D2)$$

$$AT = A*D1/(D1 + D2).$$

Then DIST is computed by:

$$DIST^2 = (AT - A1)^2 + (BT - B1)^2 .$$

This yields the desired point (BF, AF) relative to (U1, V1) as is:

$$BF = B1 + r(BT - B1)/DIST$$

$$AF = A1 + R(AT - A1 /DIST$$

from which are obtained RLAT = U1 + BF and RLONG = V1 + AF/α.

The flowchart for LAUNCH is given in figure 192. In comparing it with the above description, it is useful to note that, in the program, quantities are squared for comparison purposes. Thus $(R1)^2$ = R1SQ, $(R2)^2$ = R2SQ, p^2= PATHSQ, and $(P1)^2$ = B1SQ.

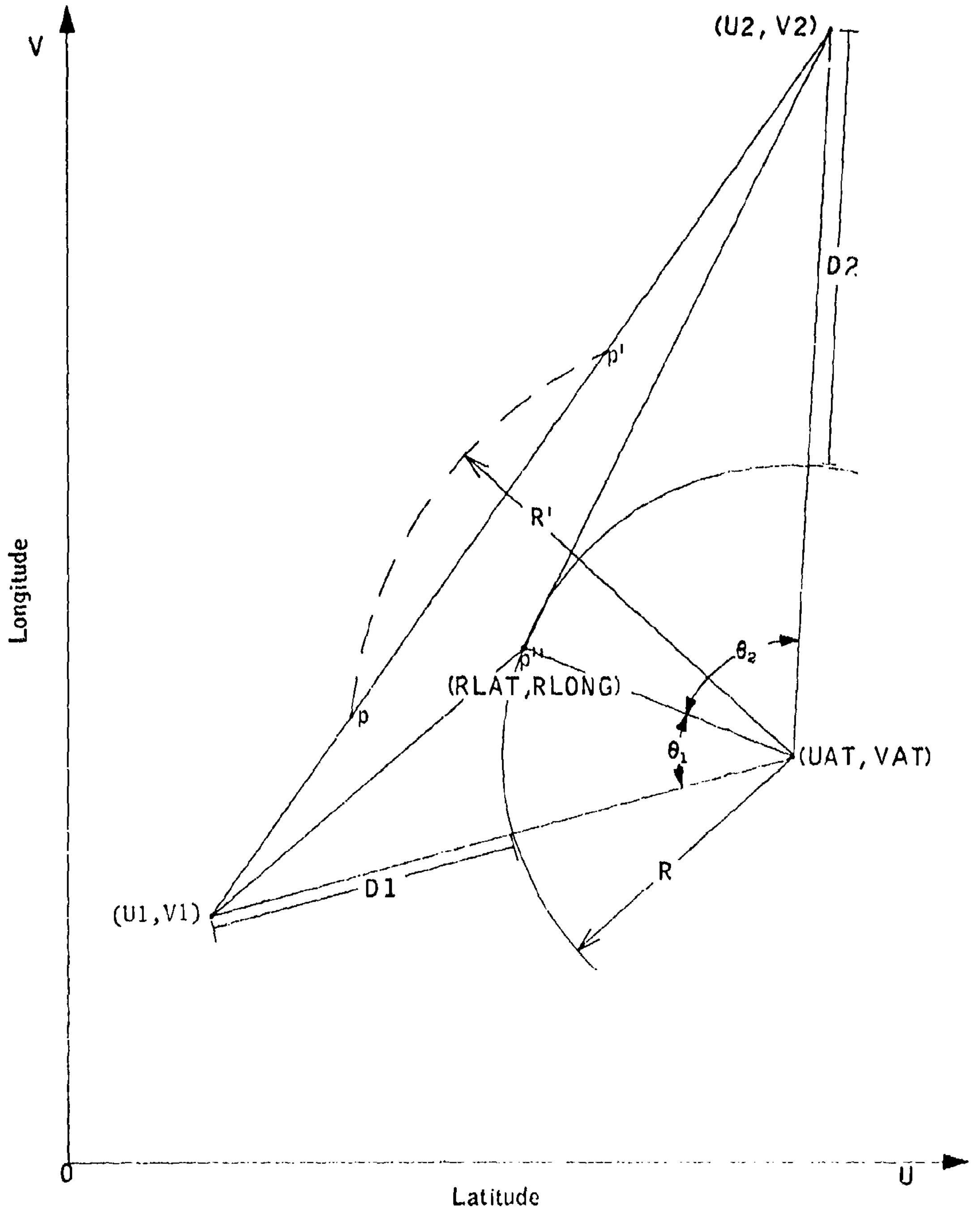


Fig. 189. Determination of ASM Aim Point

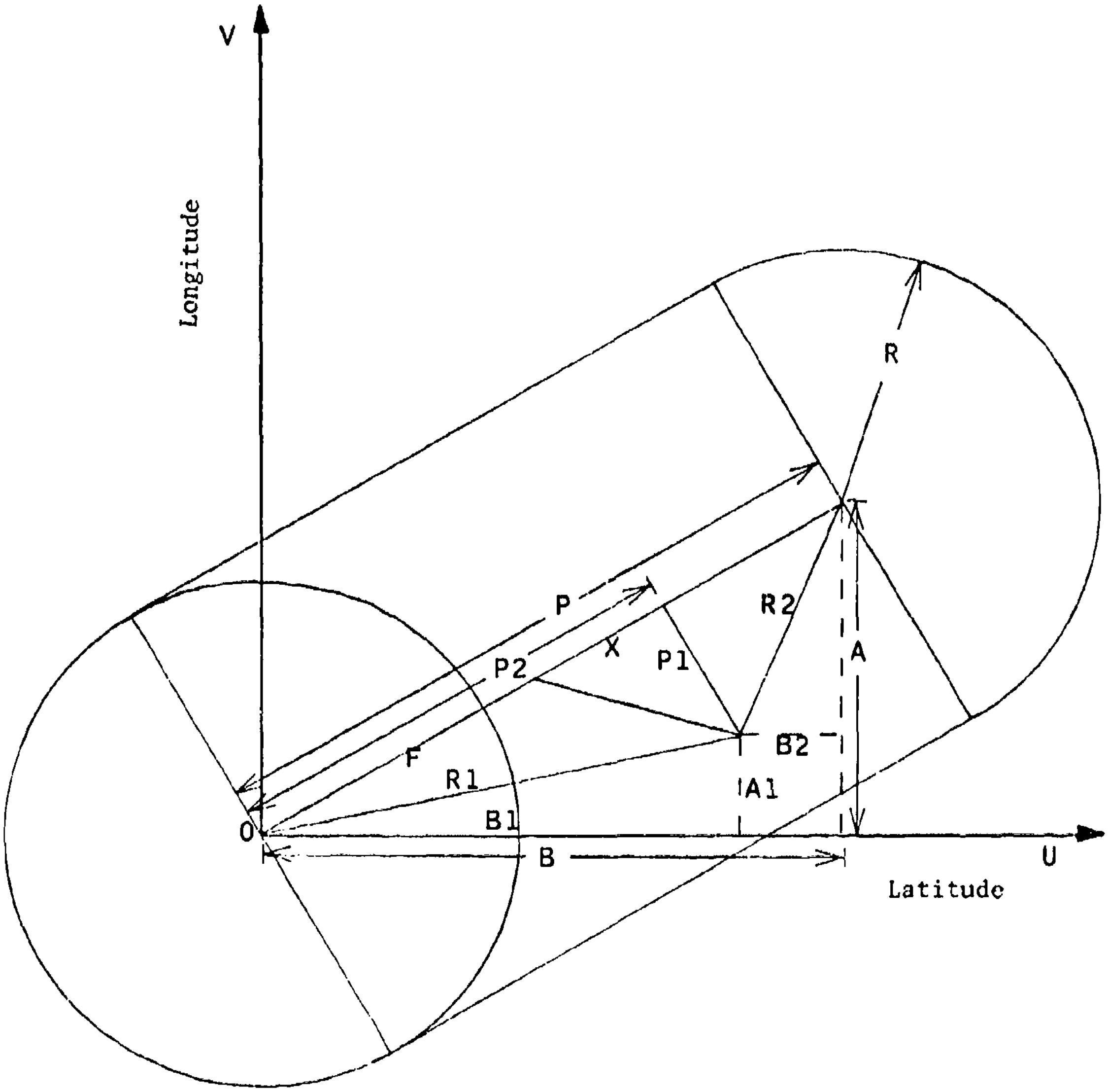


Fig. 190. LAUNCH Procedure Outline

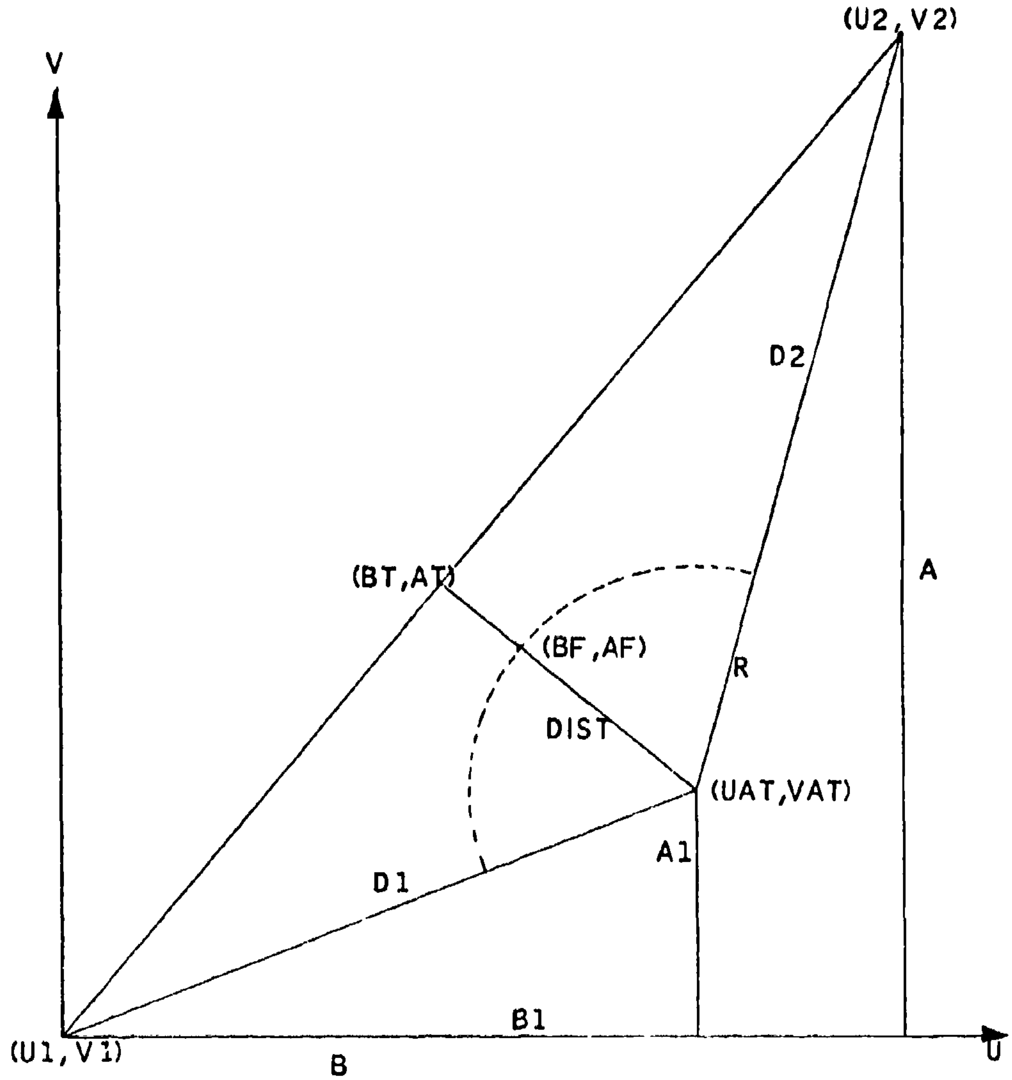


Fig. 191. Computation of Flight Path Aim Point

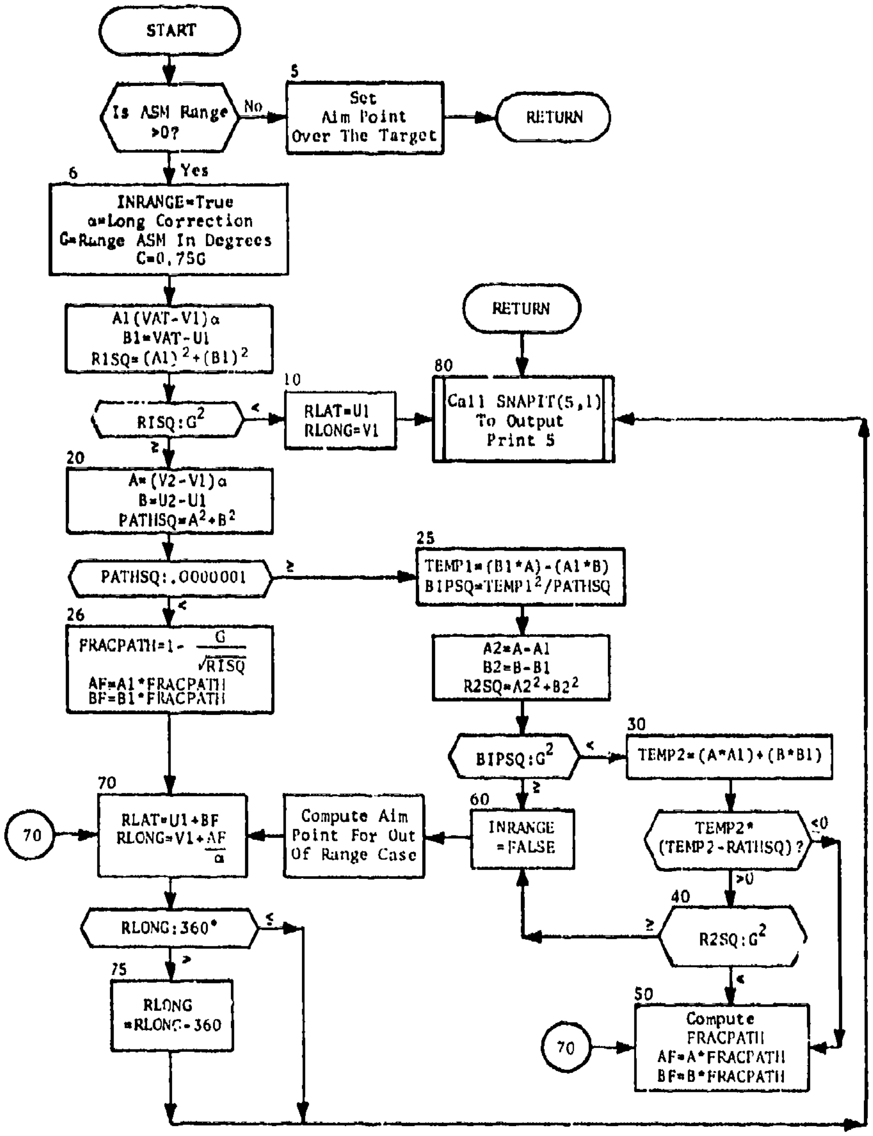


Fig. 192. Subroutine LAUNCH

SUBROUTINE LNCHDATA

PURPOSE:	To read the user card input missile timing data, and to prepare it for use by subroutine TIMELNCH.
ENTRY POINTS:	LNCHDATA
FORMAL PARAMETERS:	None
COMMON BLOCKS:	BLOCK, FILES, ITP, MASTER, MYIDENT, TIMELINE
SUBROUTINES CALLED:	DISTF, NUMGET, RDARRAY, SETREAD
CALLED BY:	PLNTPLAN

Method

Subroutine LNCHDATA reads the user data cards which contain missile timing information. It first reads the latitudes and longitudes of any missile timing lines, doing preliminary calculations on the line data to save time in subroutine TIMELNCH. Then, the CORMSL data is read and stored. FLIGHT CORMSLs, LINE CORMSLs, and FLTMIN parameters are discussed with subroutine TIMELNCH.

Since some of the timing data is contained on the MSLTIME file, LNCHDATA initializes the file for use by subroutine PLANTMIS. Finally, LNCHDATA prints the timing line and CORMSL data.

Subroutine LNCHDATA is illustrated by figure 193.

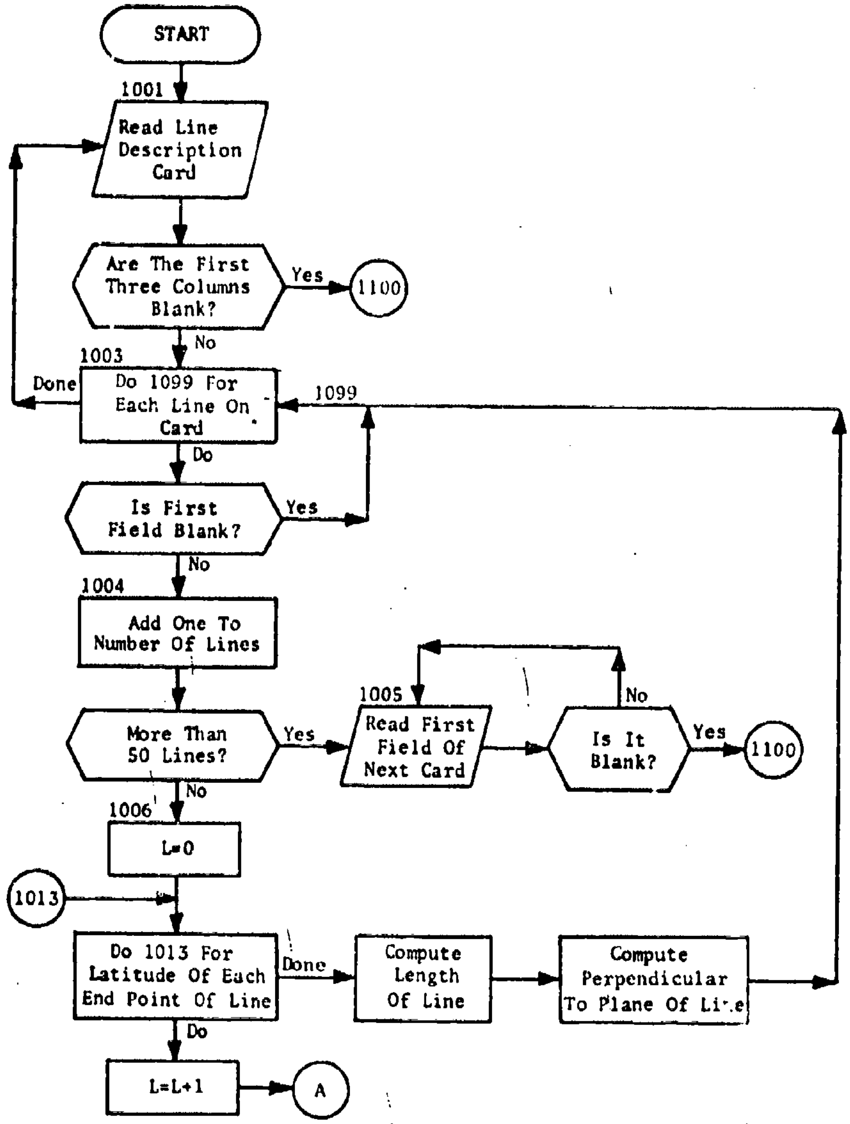


Fig. 193. Subroutine LNCHDATA
(Sheet 1 of 4)

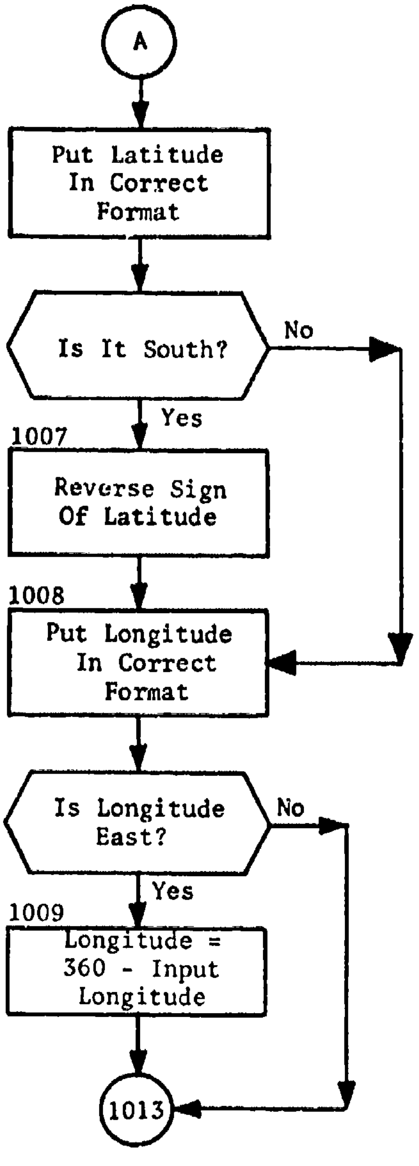


Fig. 193. (cont.)
(Sheet 2 of 4)

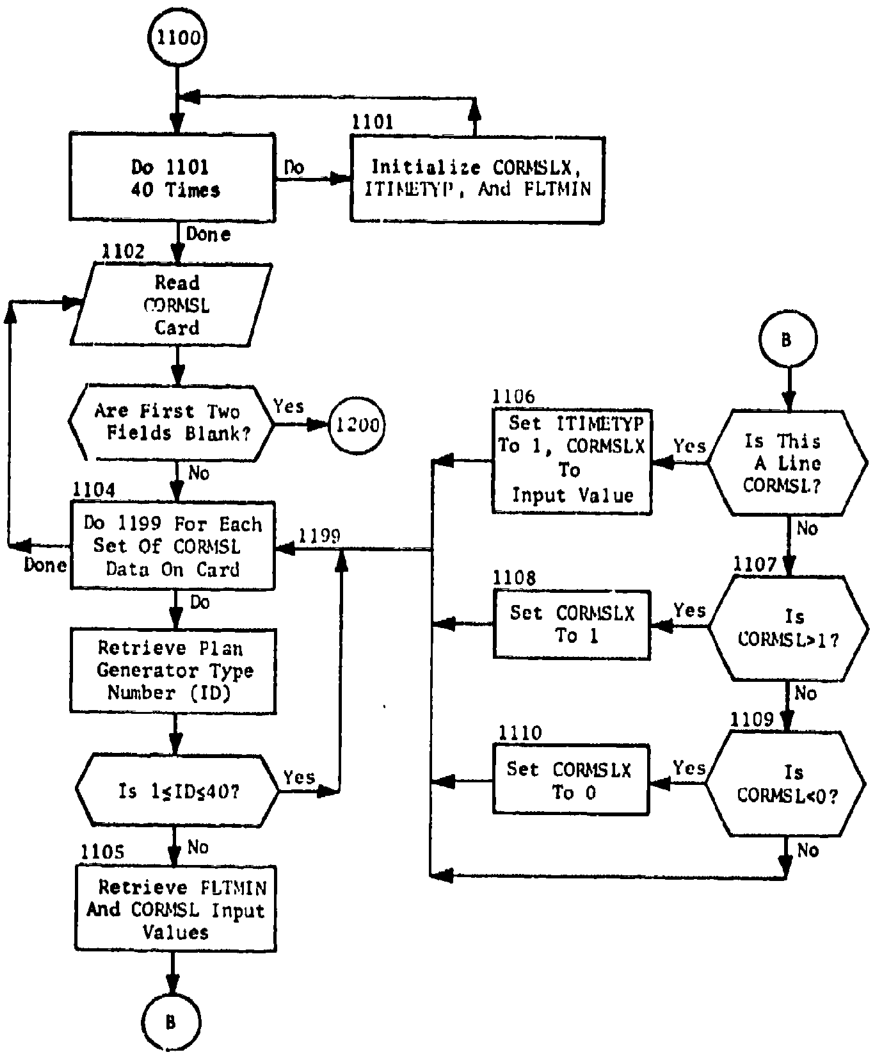


Fig. 193. (cont.)
(Sheet 3 of 4)

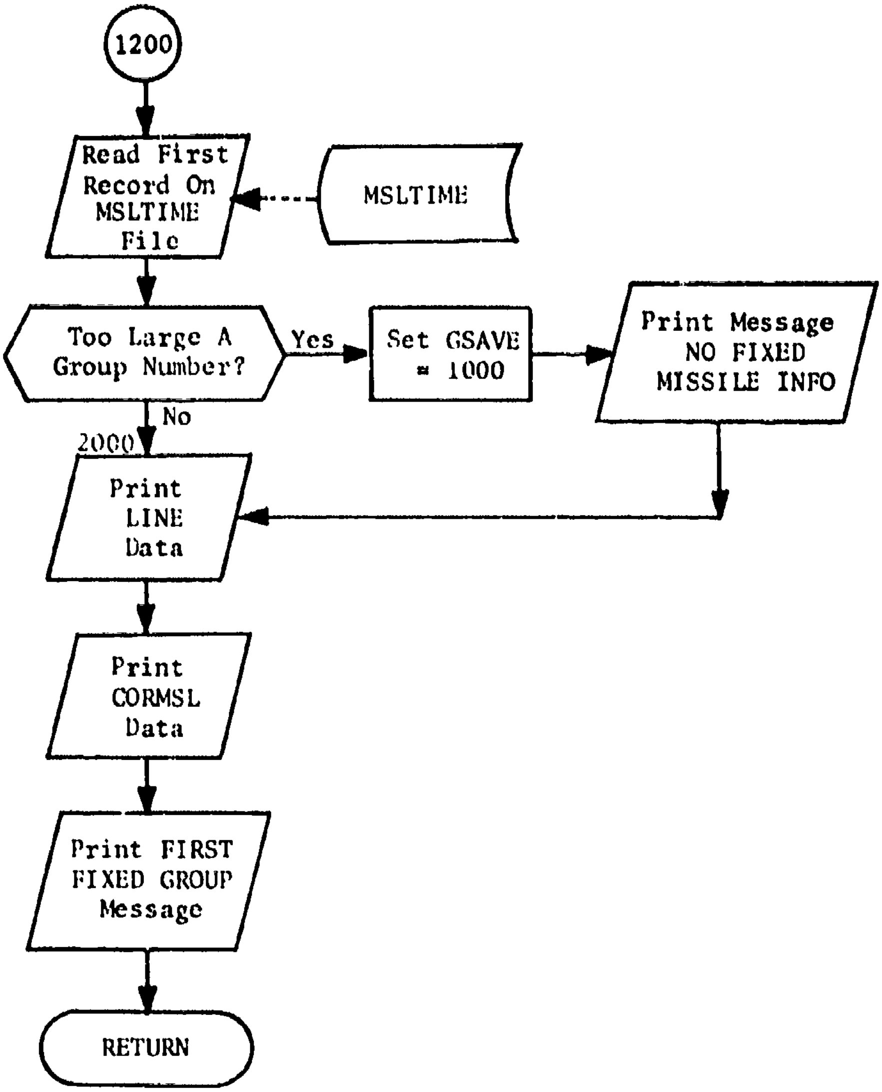


Fig. 193. (cont.)
(Sheet 4 of 4)

482 546 O - 72 - 37

SUBROUTINE PLANTANK

PURPOSE:	To process the tanker records originally contained on the BASFILE, and to generate tanker plans which correspond to them for inclusion on the EVENTAPE and/or PLANTAPE.
ENTRY POINTS:	PLANTANK
FORMAL PARAMETERS:	None
COMMON BLOCKS:	ARTIME, CONTROL, DINDATA, DINATA2, EVENTS, FILES, ICLASS, INDATA, IOUTOLD, IRF, IRFTK, ITP, KEYLENG, KEYS, MASTER, MAX, MYIDENT, OUTSRT, PLANTYPE, REF, SNAPON, TANKER, TWORD, 7, 8, 9, 10
SUBROUTINES CALLED:	CLINDATA, DISTF, IPUT, ORDER, SNAPCON, SNAPIT, VAM, WRWORD, WRARRAY
CALLED BY:	PLNTPLAN

Method

The input data for the generation of tanker plans are contained on the BASFILE in the form of records whose contents are listed in table 50. All of these records are read from the BASFILE early in the program by subroutine INITANK and stored in common /7/ for use by PLNTPLAN and subroutine PLANTANK. The number NTANKBAS of these input records is supplied to PLNTPLAN through common /MASTER/. A record is supplied for each tanker base. A separate plan is generated for each tanker on the base; i.e., N_t plans are generated for each input record, where N_t is the number of tankers on the given base.

Each tanker plan generated consists of seven events: (1) Launch event, (2) Enter Refuel Area, (3) Leave Refuel Area, with (4), (5), (6), and (7) as alternate Recovery events, as shown in table 51.

Tanker plans are generated in the following operation. As each input record is read in, N_t plans are generated: N_a plans for alert tankers, and then N_t - N_a for nonalert tankers.

Each tanker base is first inspected to determine if its tankers are to be automatically allocated. Then, after all bases have been inspected, PLANTANK fills common block /9/ with required data and calls subroutine

VAM to allocate those tankers to specific refuel areas in such a way as to minimize the total miles flown by them while servicing all bomber requests.

When VAM has returned its solution, PLANTANK allocates any extra tankers and then proceeds to calculate the time schedules for individual flights.

In the second-strike case, all tankers are sent to their assigned refuel areas at the earliest possible moment, considering delays before launch due to alert or nonalert status as well as the travel time required between base and refuel area.

In the first-strike case, however, they are scheduled as follows. Bombers have been scheduled by PLNTPLAN to arrive at specific refuel areas over a period of time (which may be several hours) so as to satisfy the requirements associated with the CORBOMB input parameter. These bomber refuels have been posted in the matrix IARVLS/ARVLS (I,J) where J indicates data for the Jth refuel to be scheduled by PLNTPLAN, I=1 contains the scheduled time of the Jth refuel, I=2 contains the assigned refuel area.

As each tanker is processed, the IARVLS array is searched for the first unserviced bomber refuel which is to occur at the refuel area to which the tanker has been assigned by subroutine VAM. When found, the bomber time of arrival is retrieved, the tanker is scheduled to launch so as to arrive at the refuel area .1 hour prior to the bomber, and the IARVLS entry is set to zero to indicate that the bomber has been serviced. If the search finds no unserviced bomber at the refuel area, the tanker is extra, thus PLANTANK schedules it to arrive .1 hour before the earliest bomber at the area (stored in array ARTIME).

After scheduling has beem completed, distances from refuel area to recovery bases are calculated for each tanker, the recovery events are ordered by ascending distance, and EVENTAPE, PLANTAPE, and printed reports are output with tanker plans according to user options.

Figure 194 illustrates subroutine PLANTANK.

Table 50. Tanker Input Record

ITEM	Fortan Name	Symbolic Name
Tanker base index	INDEXTK	
Base latitude	TKLAT	
Base longitude	TKLONG	
Refuel area	IREFTK	
Number of tankers per squadron	NPSQNTK	N_t
Number of tankers on alert per squadron	NALRTK	N_a
Tanker speed	SPEEDTK	V_t
Alert delay	DLYALTK	D_a
Nonalert delay	DLYALTK	D_n
Total time on station	TTOS	
Tanker type	ITYPETK	
Tanker range	TANKRNGE	

Table 51. Tanker Plan

Event Type	Time	Place
Launch	Delay	INDEXTK
Enter Refuel Area	$DIST/V_t$	IREFTK as set by PLANTANK
Leave Refuel Area	TTOS	IREFTK as set by PLANTANK
$Recover_1$	DI_1/V	$(RCBLAT, RCBLONG)_1$
$Recover_2$	DI_2/V_t	$(RCBLAT, RCBLONG)_2$
$Recover_3$	DI_3/V_t	$(RCBLAT, RCBLONG)_3$
$Recover_4$	DI_4/V_t	$(RCBLAT, RCBLONG)_4$

Where DIST = Distance from tanker base to refuel area
DI_x = Distance from refuel area to recovery $base_x$

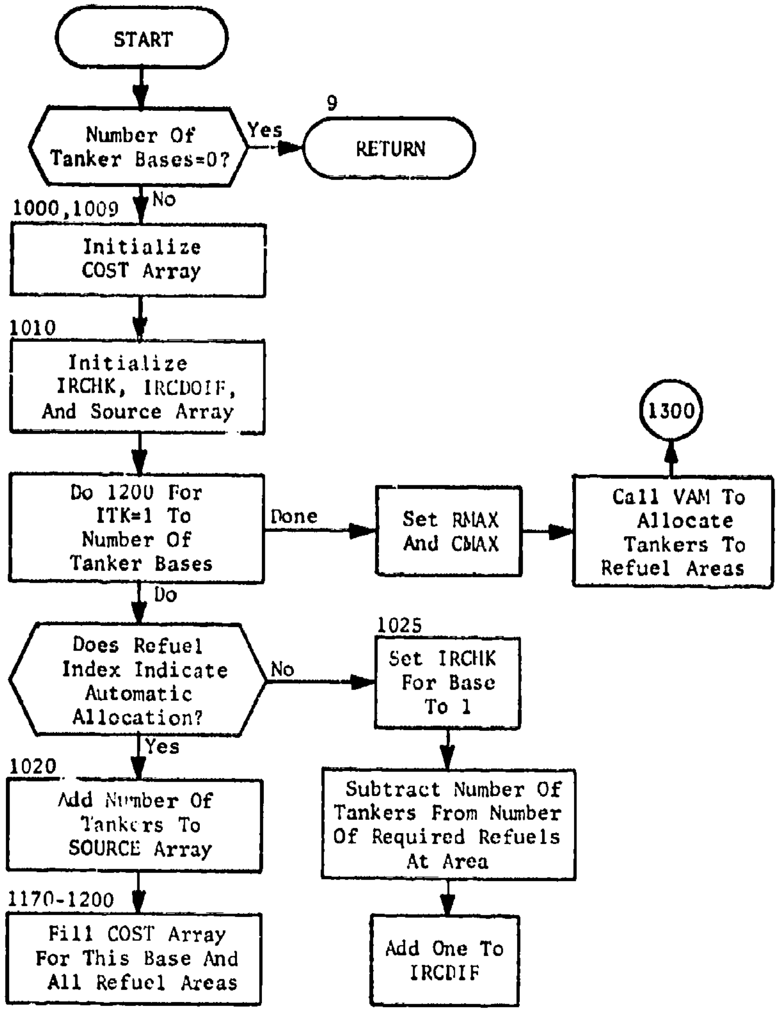


Fig. 194. Subroutine PLANTANK
(Sheet 1 of 4)

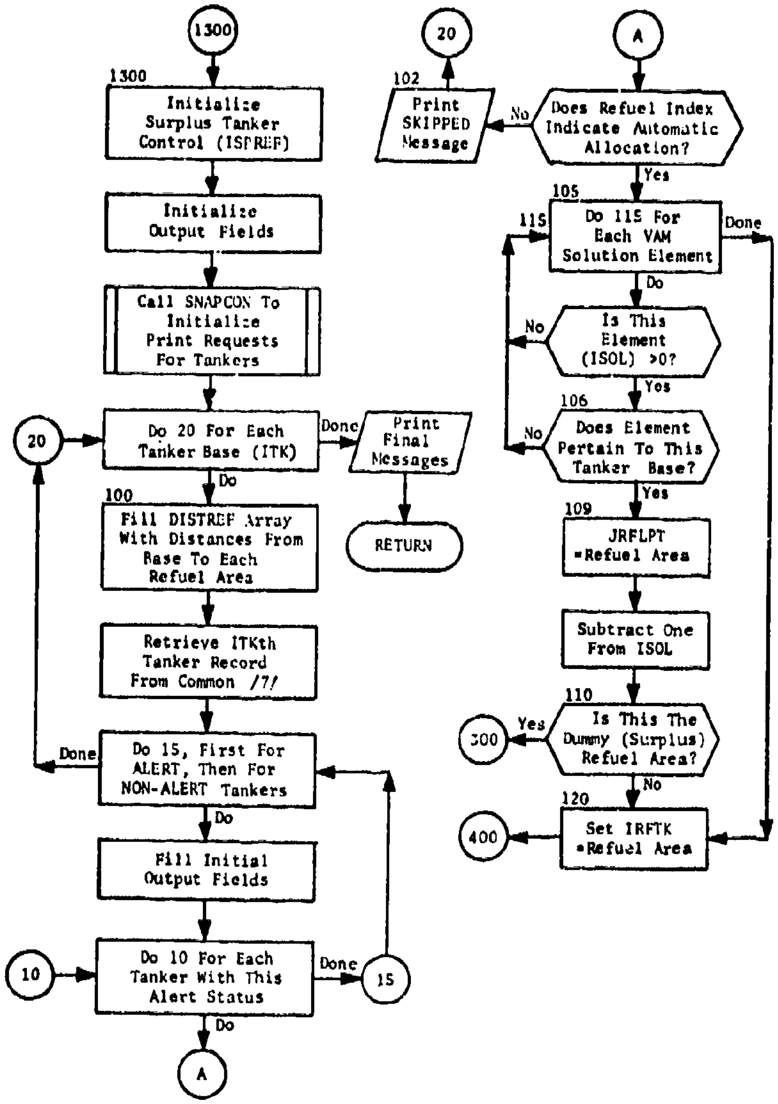


Fig. 194. (cont.)
(Sheet 2 of 4)

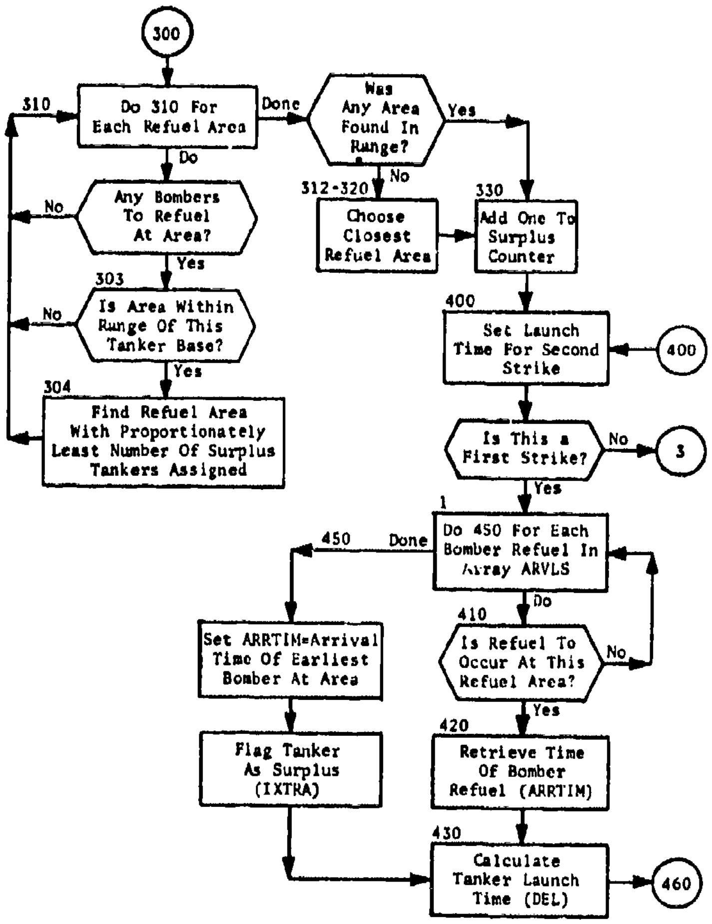


Fig. 194. (cont.)
(Sheet 3 of 4)

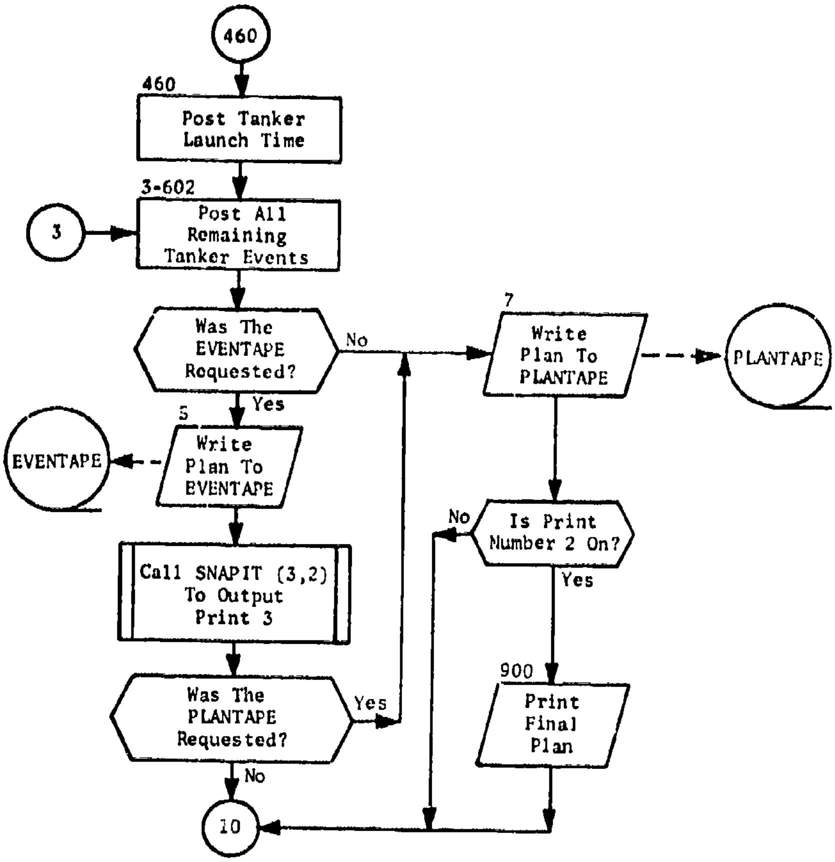


Fig. 194. (cont.)
(Sheet 4 of 4)

SUBROUTINE PLANTMIS

PURPOSE: To control the processing of all missile plans input from the STRKFILE.

ENTRY POINTS: PLANTMIS

FORMAL PARAMETERS: None

COMMON BLOCKS: BLOCK, CONTROL FILES, ITP, KEYLENG, MAX, MISCT, MRVFLG, NAVAL, OUTSRT, PAYLOAD, SNAPON, TIMELINE, TWORD, WPNGRPX, WPNTYPEX

SUBROUTINES CALLED: ABORT, RDARRAY, SNAPIT, TIMELNCH, TIMEME, WRARRAY, WRWORD

CALLED BY: PLNTPLAN

Method

This subroutine reads and processes missile plans from the STRKFILE. It is called whenever PLNTPLAN reads a missile record.

The STRKFILE record is first moved from /OUTSRT/ to /BLOCK/. To facilitate processing, most of the data is then transferred to the TDATA array in common /TIMELINE/. Table 52 shows the variable placement in arrays BLOCK and TDATA.

The remainder of PLANTMIS merely outputs missile plans in the correct format onto the PLANTAPE and EVENTAPE. If the weapon group under consideration has a time-dependent DBL probability, the subroutine precedes the missile launch events for each base on the EVENTAPE with a naval DBL destruct event. The last words in each EVENTAPE missile launch event record are target data words which occur in pairs, one pair for each target. The second word is the time of flight from launch to target. The format of the first word, along with that of the DBL destruct event, is shown in table 42 under PLNTPLAN Output Files.

When PLANTMIS has completed processing a missile record, it reads the next plan from the STRKFILE. If the information read is another missile record, PLANTMIS processes it without returning to PLNTPLAN; otherwise, it returns.

All processing of missile plans is done in subroutine PLANTMIS, TIMELNCH, and LNCHDATA.

Figure 195 illustrates subroutine PLANTMIS.

Table 52. Arrays TDATA/ITDATA and BLOCK/LOCK Used in PLANTMIS and TIMELNCH

TDATA/ITDATA INDEX	ATTRIBUTE	BLOCK/LOCK INDEX
---	STRKFILE or EVENTAPE Record words 1-18	1-18
1-18	Missile indices	19-36
19-36	Site indices (from data base)	37-54
37-54	Target indices (from data base)	55-72
55-72	Offset latitude (DLAT)	73-90
73-90	Offset longitude (DLONG)	91-108
91-108	Flight Time (hours)	109-126
109-126	Weapon site latitude	127-144
127=144	Weapon site longitude	145-162
145-162	Target latitude	163-180
163-180	Target longitude	181-198
181-198	Target designator	199-216
199-216	Target task	217-234
217-234	Target country	235-252
235-252	Target flag	253-270

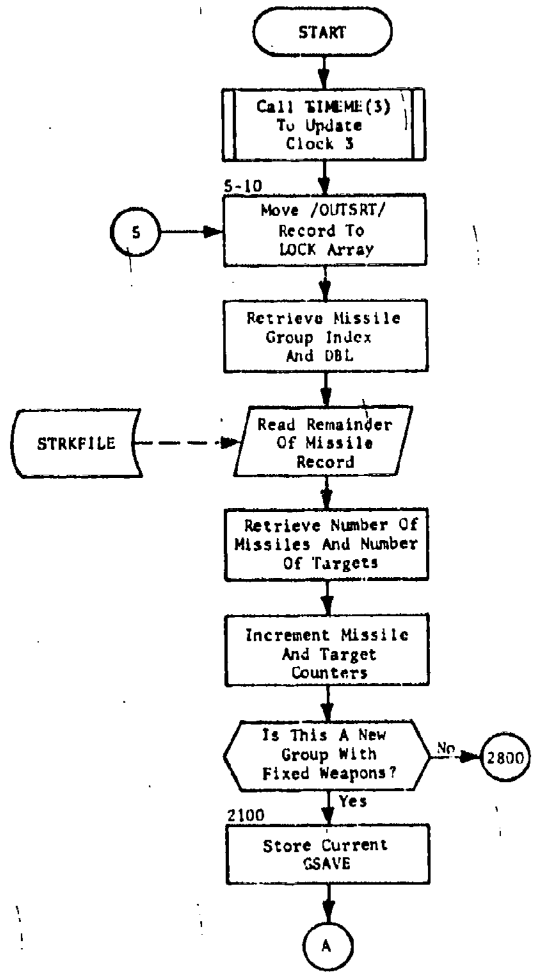


Fig. 195. Subroutine PLANTMIS
(Sheet 1 of 4)

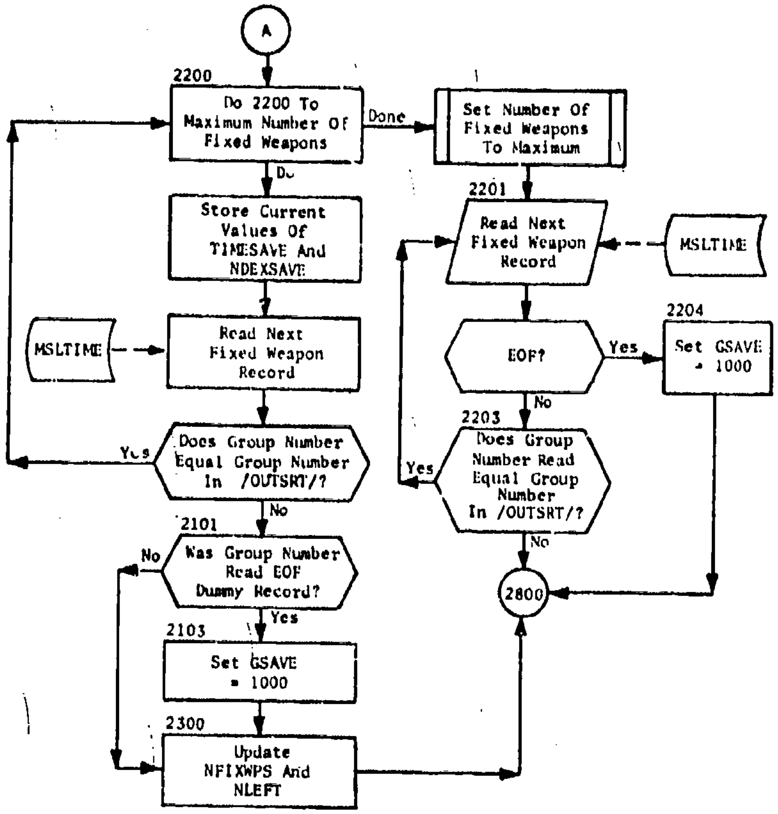


Fig. 195. (cont.)
(Sheet 2 of 4)

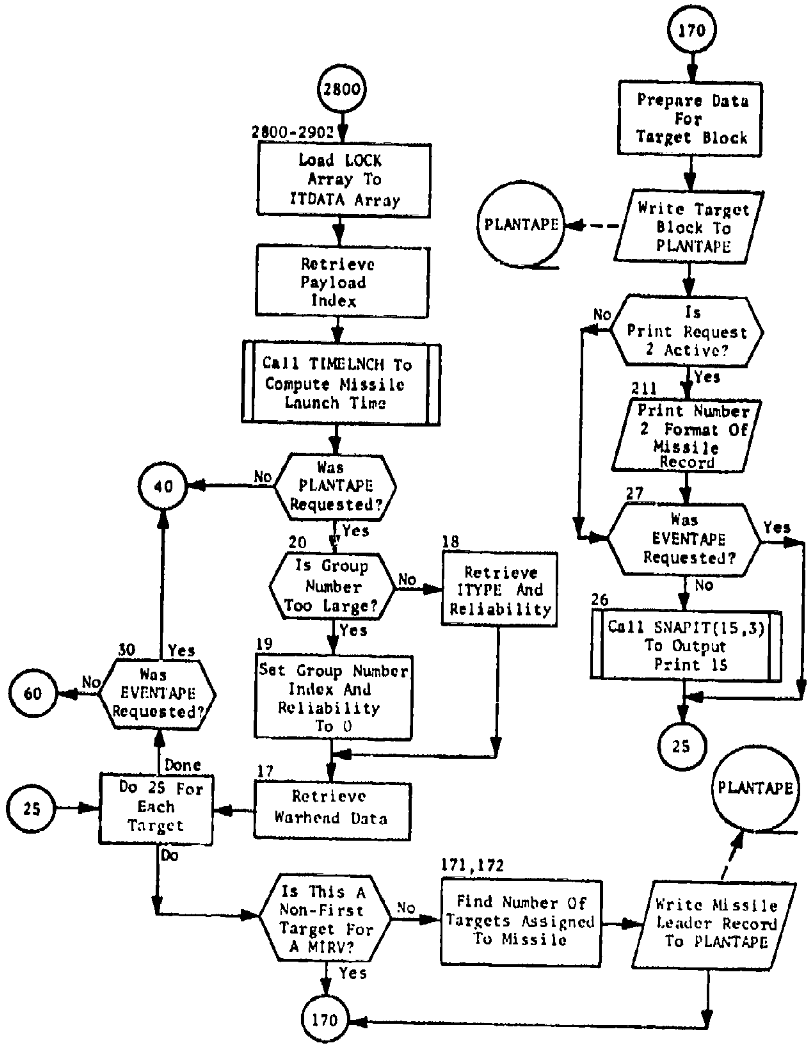


Fig. 195. (cont.)
(Sheet 3 of 4)

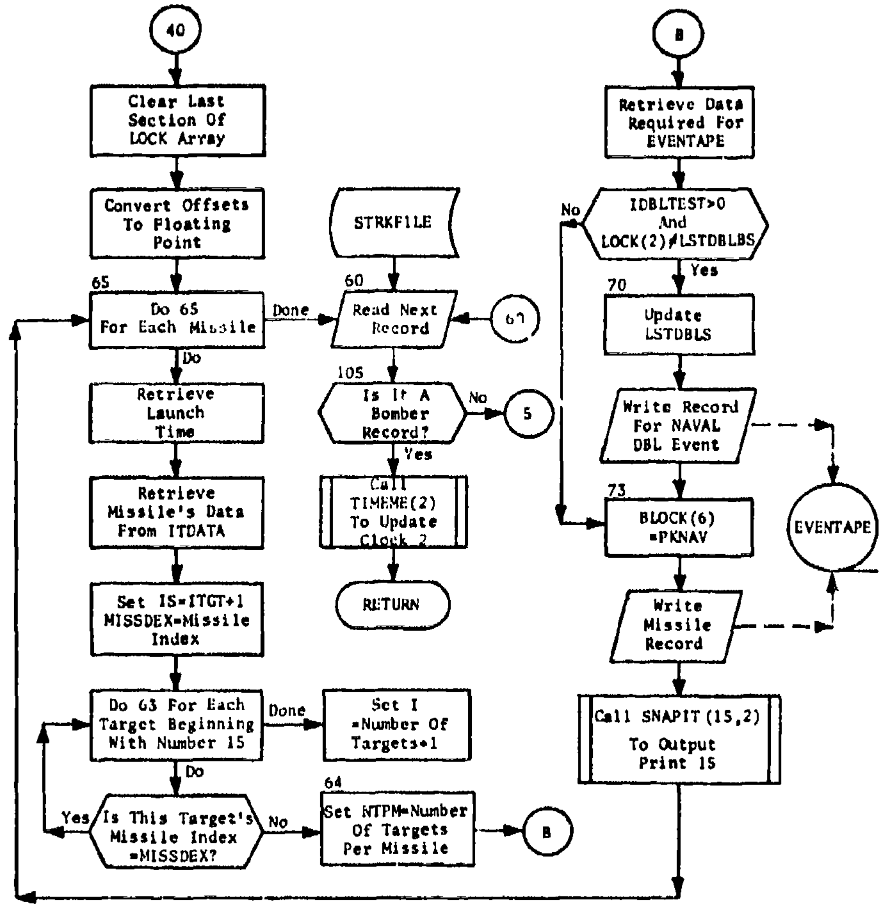


Fig. 195. (cont.)
(Sheet 4 of 4)

SUBROUTINE POST

PURPOSE:	To enter the event type, event location, and place code to the arrays in common /DINDATA/.
ENTRY POINTS:	POST POST4 (Refuel event) POST5 (Enter Zone event) POST8 (Local Attrition event) POST15 (formerly, Launch Decoy event; now inactive) POST17 (Change Altitude event)
FORMAL PARAMETERS:	A = Latitude of event B = Longitude of event I = Place code for event
COMMON BLOCKS:	DINDATA, EVENTS, FILES
SUBROUTINES CALLED:	None
CALLED BY:	PLNTPLAN, ZONECROS

Method

The event type is determined by the entry point used. Subroutine POST increments the /DINDATA/ line counter (MHT) by 1, then enters the event type code in JTP (MHT), the latitude of the event in HLA (MHT), the longitude in HLO (MHT) and the place index in KPL (MHT).

Subroutine POST is illustrated in figure 196.

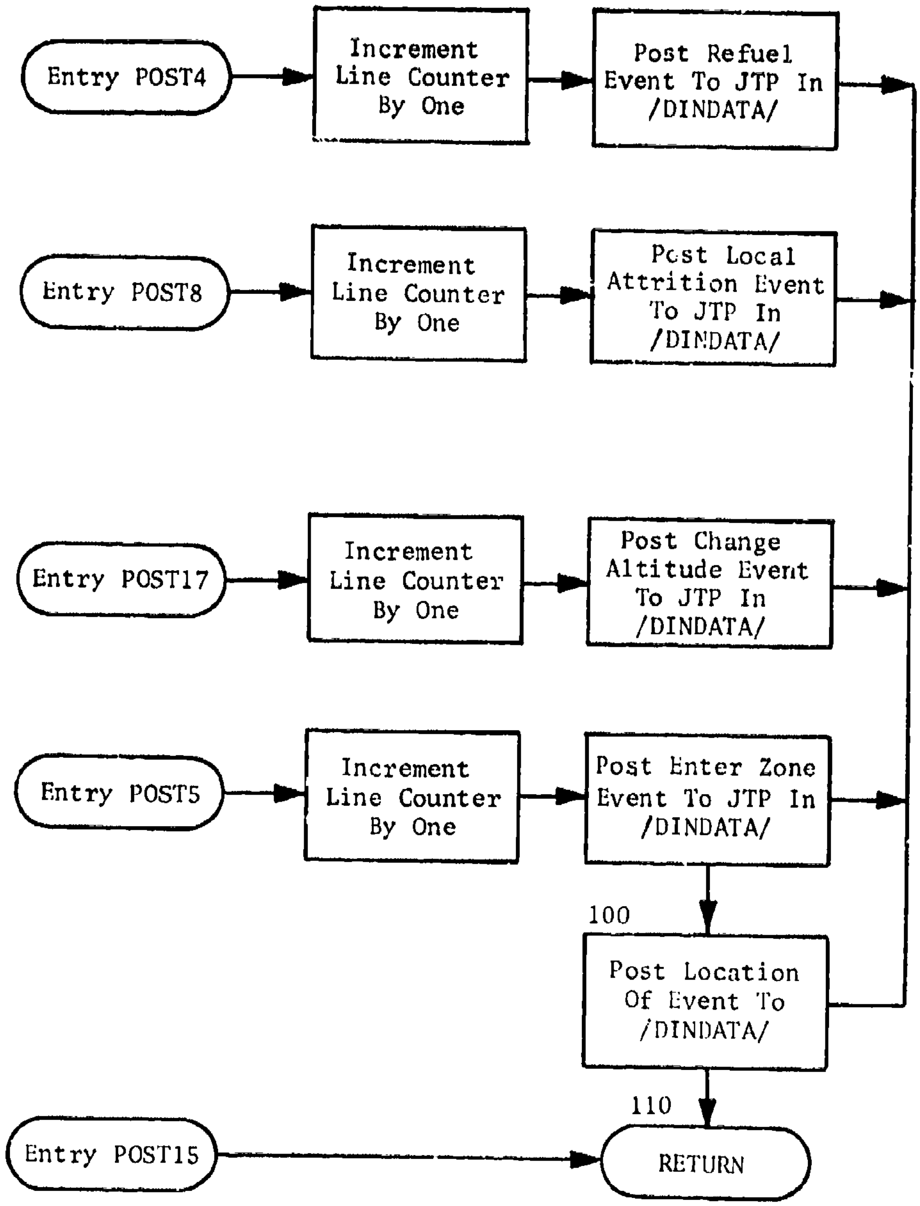


Fig. 196. Subroutine POST

SUBROUTINE POSTLAUN

PURPOSE:	To add information pertaining to possible decoy launches to the arrays examined by subroutine DECOYADD.
ENTRY POINTS:	POSTLAUN
FORMAL PARAMETERS:	LPR, LHT, LDST
COMMON BLOCKS:	1
SUBROUTINES CALLED:	None
CALLED BY:	PLNTPLAN, ZONECROS

Method

POSTLAUN increments the counter for the number of possible decoy launches (NPSLN) and stores the information sent through the calling parameters in the appropriate arrays, indexed by NPSLN. Parameter LPR contains the priority of the possible launch; parameter LHT contains the number of the event following the possible launch; and parameter LDST contains the index to the word in DELDIS in which the decoy coverage distance is stored. Parameter LDST will be meaningful only for launches of priority 5, 7, or 8.

Subroutine POSTLAUN is illustrated in figure 197.

482-548 O - 72 - 33

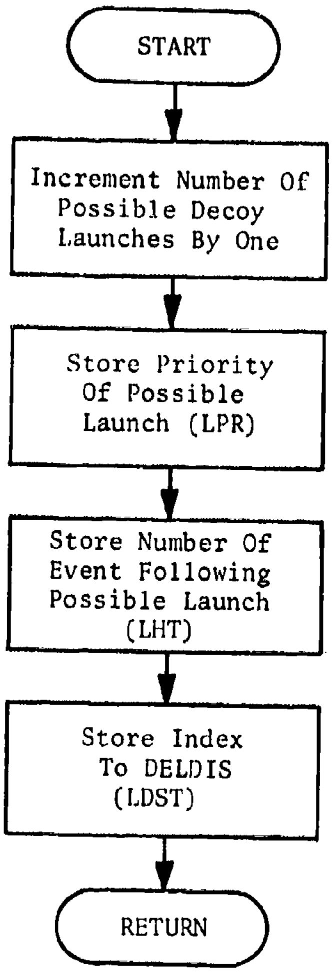


Fig. 197. Subroutine POSTLAUN

SUBROUTINE PRNTAB

PURPOSE: To print the final tanker allocation tables.

ENTRY POINTS: PRNTAB

FORMAL PARAMETERS: None

COMMON BLOCKS: ARTIME, IRF, MASTER, REF

SUBROUTINES CALLED: LATCV, LONCV

CALLED BY: PLNTPLAN

Method

Subroutine PRNTAB outputs tanker allocation information, first for the user-assigned refuel areas, then, after skipping a line, for the refuel areas calculated by PLNTPLAN.

Subroutine PRNTAB is illustrated in figure 198.

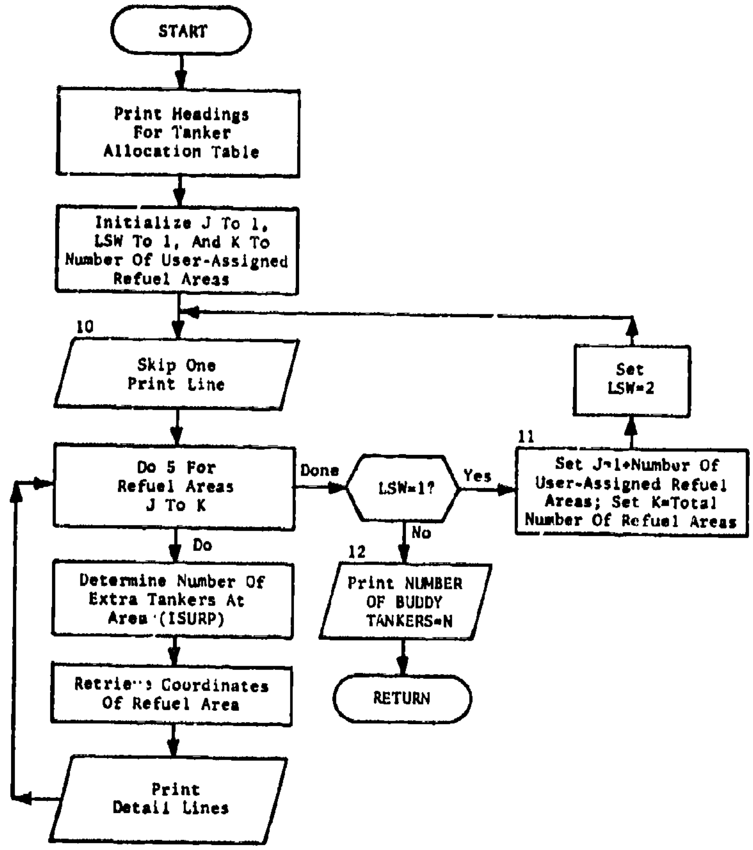


Fig. 198. Subroutine PRNTAB

SUBROUTINE SNAPCON

PURPOSE:	To control the activation of optional prints within PLNTPLAN.
ENTRY POINTS:	SNAPCON
FORMAL PARAMETERS:	None
COMMON BLOCKS:	IFTPRNT, OUTSRT, SNAPON
SUBROUTINES CALLED:	None
CALLED BY:	PLNTPLAN

Method

This subroutine with SNAPIT and SNAPOUT provides the capability for optional printing within PLNTPLAN. SNAPCON reads the print request cards initially, and uses this information to control the activation of prints during PLNTPLAN processing. SNAPIT is called wherever an optional print is to be issued; SNAPIT, in turn, calls SNAPOUT to do the actual printing.

For each input record from STRKFILE, SNAPCON checks the print request list with a particular value of a print control parameter, say group, to determine which prints are to be activated. Let x be the value of the print control parameter; e.g., the group number on the current input record. Suppose that a = the starting group and b = the finishing group as specified on the print request card. Then x is checked to determine whether it is in the interval a to b. Either a or b, or both, may be blank or zero on the control card. Table 53 lists the possible values of a and b, and the value that x should assume if the print is to be active in each of these cases. Let a_m be the minimum value x can have and let b_m be its maximum value. Then the following single test of x suffices to determine if x is such that the print is active:

$$a' + a_m L(a') \leq x \leq b' + a'L(b') + b_m L(a')$$

where L(x) = 1 if x = 0 and is 0 otherwise.

For each print number (1 to 15) which is to be active for the current plan, SNAPCON sets the corresponding element(s) in the NAP array to 3.

Subroutine SNAPCON is illustrated in figure 199.

Table 53. Possible Values of a and b

a	b	VALUE OF x FOR ACTIVE PRINT
0	0	any value
a'	0	$x = a'$
0	b'	any value
a'	b'	$a' \leq x \leq b'$

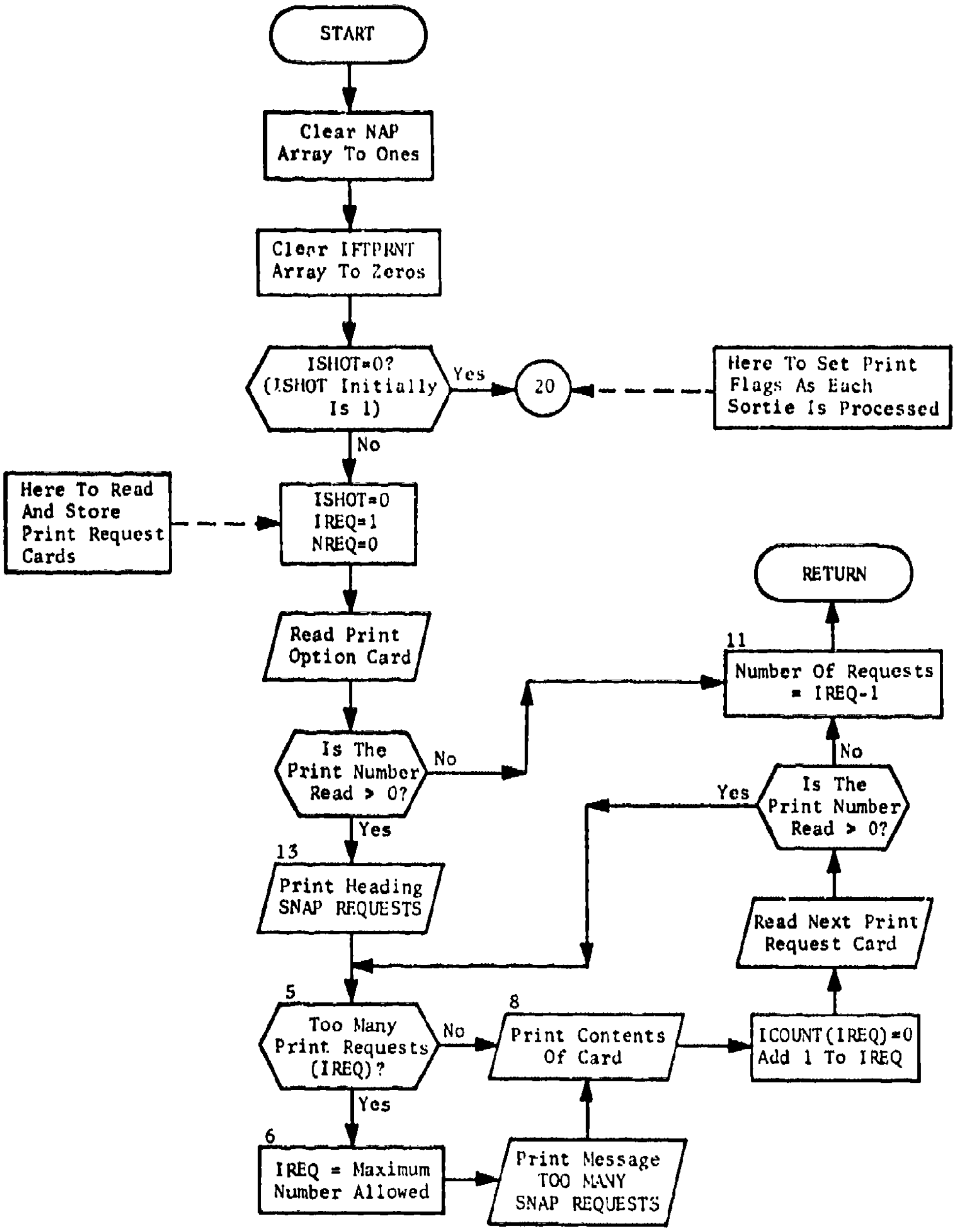


Fig. 199. Subroutine SNAPCON (Sheet 1 of 2)

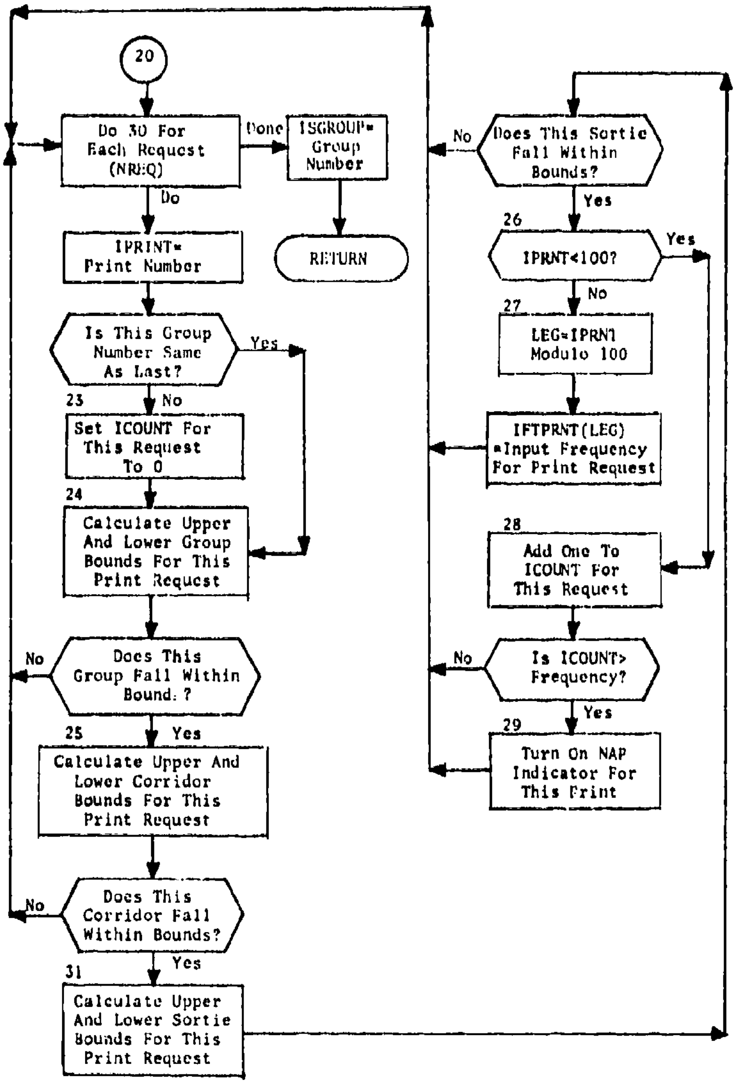


Fig. 199. (cont.)
(Sheet 2 of 2)

SUBROUTINE SNAPIT

PURPOSE:	SNAPIT is called whenever a print is to be issued. It determines whether the print is active and, if so, calls SNAPOUT to issue the print.
ENTRY POINTS:	SNAPIT
FORMAL PARAMETERS:	IO = Print request number NO = Print option code
COMMON BLOCKS:	SNAPON
SUBROUTINES CALLED:	SNAPOUT, TIMEME
CALLED BY:	PLNTPLAN, BOUNDARY, LAUNCH, PLANTANK, PLANTMIS

Method

When the user's print option cards are read at the beginning of PLNTPLAN execution, the encountering of each unique print request number (numbered 1 through 15) causes the corresponding element of the array NAP in block SNAPON to be set to 3. Otherwise, the element will contain 1.

During processing of PLNTPLAN, subroutine SNAPIT is called and given the applicable print number along with a print option code whenever a print is to be issued. If the NAP flag for the requested print is not equal to 3, SNAPIT simply returns. Otherwise, the print option code is checked for the value 1, SNAPOUT is called with both IO and NO parameters, and SNAPIT returns.

A print option code of 1 causes SNAPIT to print the message "PRINT NUMBER _, and if it is print 4, the message BOUNDARY SNAP before calling subroutine SNAPOUT. This option code, regardless of its value, is passed on to SNAPOUT where it has significance for prints 4, 11, and 15.

Subroutine SNAPIT is illustrated in figure 200.

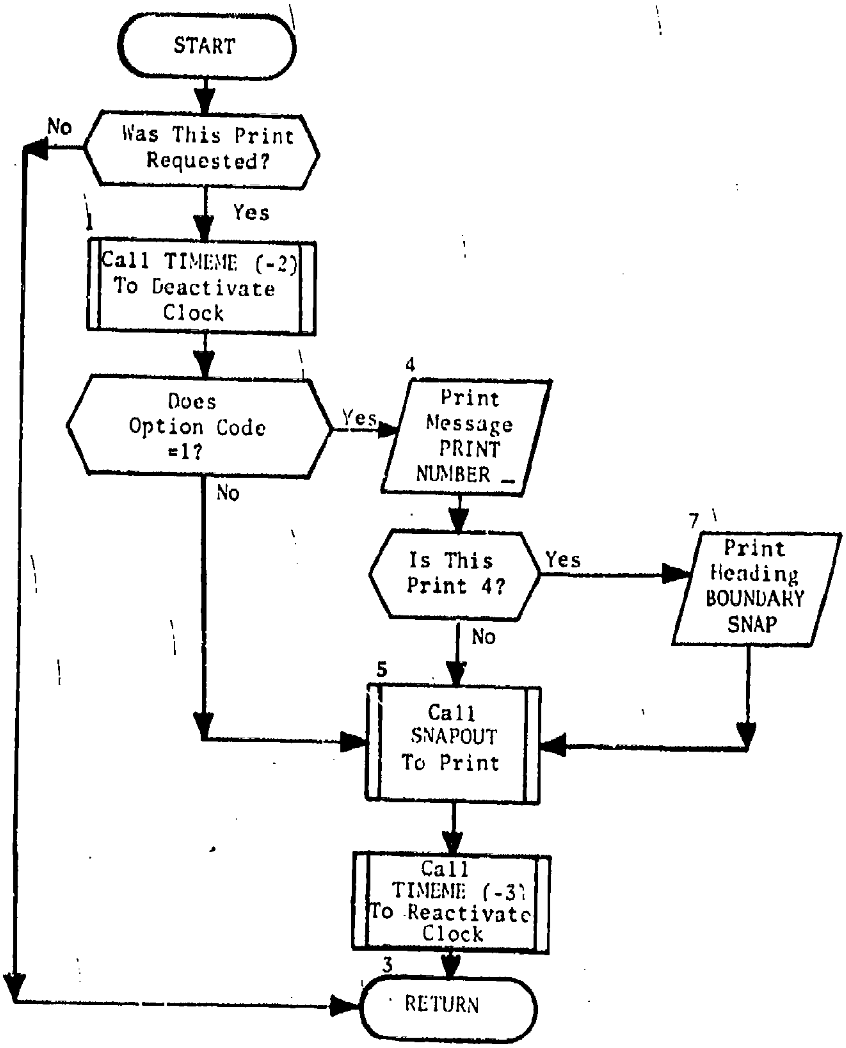


Fig. 200. Subroutine SNAPIT

SUBROUTINE SNAPOUT

PURPOSE: To perform all optional printing within PLNTPLAN.

ENTRY POINTS: SNAPOUT

FORMAL PARAMETERS: ILK = Print request number
MLK = Print option code

COMMON BLOCKS: ASMARRAY, ASMTABLE, BLOCK, BOUND, BOUNDPT, CONTROL, CORCOUNT, CORRCHAR, DINDATA, DINDATA2, DISTC, FILES, HILO, IDP, INDATA, IOUTOLD, IRF, IRFTK, ITP, KEYLENG, LASM, LAUNSNAP, MASTER, MH, MH2, OUTSRT, PAYLOAD, PLANTYPE, SPASM, TANKER, WPNGRPX, WPNTYPEX, 9

SUBROUTINES CALLED: DISTF, LATCV, LONCV, TIMEME

CALLED BY: SNAPIT

Method

The User's Manual describes the optional prints available in PLNTPLAN (numbered 1 through 15), and the data card format to be used when requesting them. The cards are read initially by subroutine SNAPCON.

Then during PLNTPLAN processing, subroutine SNAPCON is entered as each new STRKFILE bomber record is read in to be processed. SNAPCON scans the list of requests and determines, by matching the group, corridor, and sortie numbers of the incoming record against the request list, which prints are to be activated and which are to be inactive during the processing of the record. It communicates this information to subroutine SNAPIT via common block /SNAPON/. This contains a fifteen-word array NAP, one cell for each print request. The cell is set to 3 for active requests; otherwise it is set to one.

The subroutine SNAPIT is called during the PLNTPLAN program or its subroutines wherever a particular print might possibly be issued. For example, SNAPIT is called upon to print the detailed plan immediately after this plan has been completed. SNAPIT then checks cell 3 of NAP and issues the print only if the cell is set to 3. It calls on subroutine SNAPOUT to do the actual printing. This separation of subroutines is made because the resulting FORTRAN-produced program is more efficient; SNAPIT and SNAPOUT might logically be treated as one subroutine.

SNAPOUT itself contains only printing routines. In some instances, the print option code (MLK), passed with the print request number, may select differing print options within the given print number.

Subroutine SNAPOUT is illustrated in figure 201.

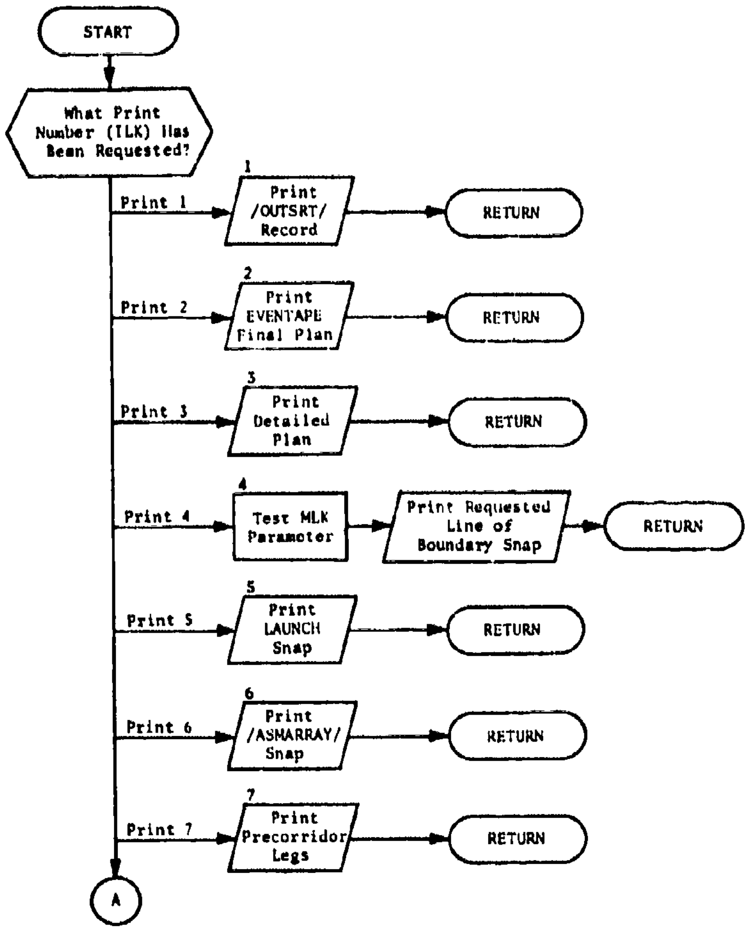


Fig. 201. Subroutine SNAPOUT
(Sheet 1 of 2)

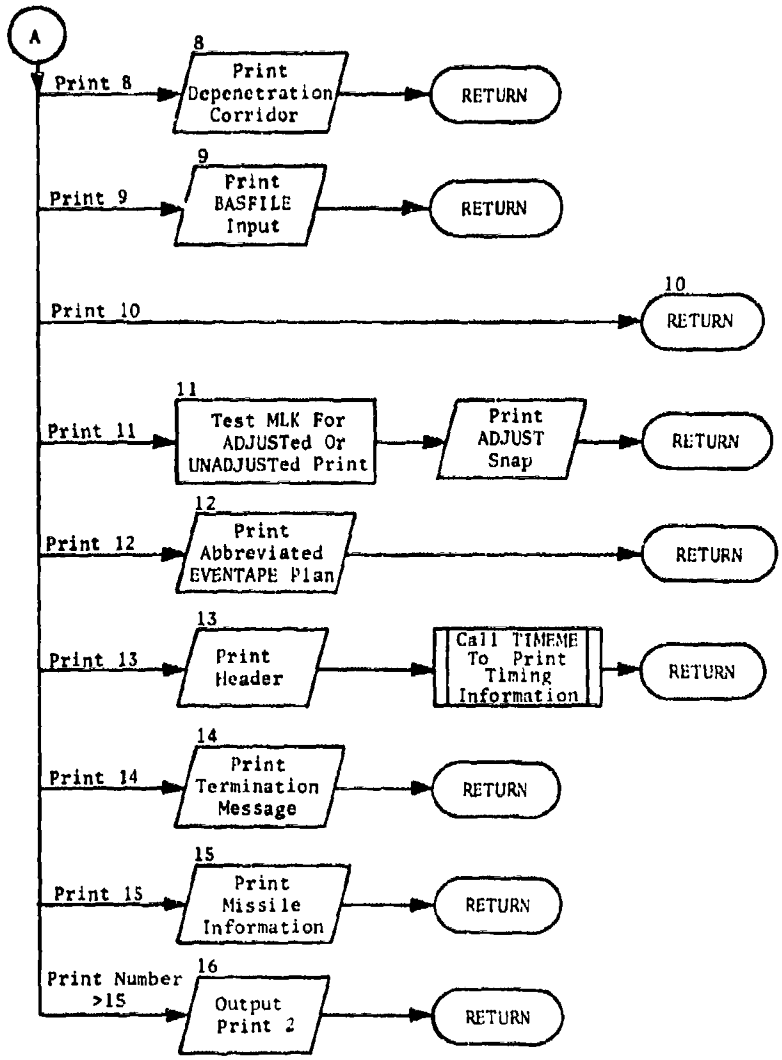


Fig. 201. (cont.)
(Sheet 2 of 2)

SUBROUTINE SWTCHALT

PURPOSE:	To convert each bomber plan's change altitude events to go high or go low events.
ENTRY POINTS:	SWTCHALT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	DINDATA, EVENTS
SUBROUTINES CALLED:	None
CALLED BY:	PLNTPLAN

Method

SWTCHALT examines the plan contained in common block /DINDATA/, and replaces the event code for each odd-numbered Change Altitude event with a go low event code. The even-numbered Change Altitude events become go high's.

Subroutine SWTCHALT is illustrated in figure 202.

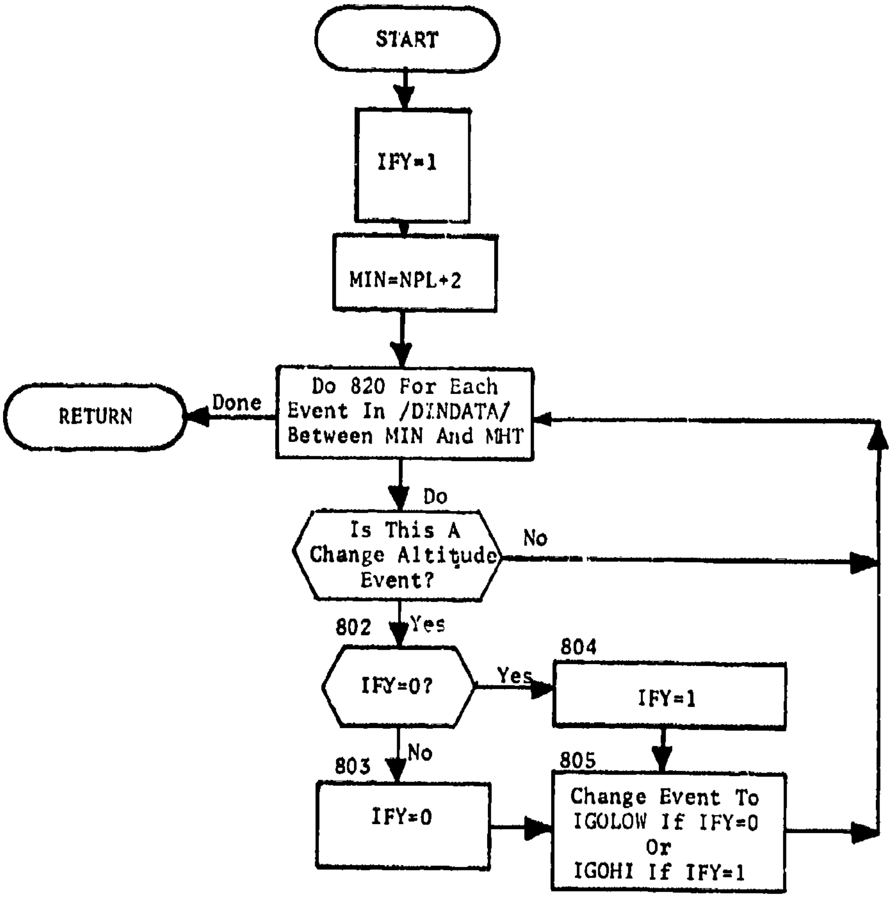


Fig. 202. Subroutine SWTCHALT

SUBROUTINE TIMELNCH

PURPOSE:	To compute launch times for missiles, and to assign specific fixed weapons to specific targets.
ENTRY POINTS:	TIMELNCH
FORMAL PARAMETERS:	None
COMMON BLOCKS:	BLOCK, PLANTYPE, TIMELINE, WPNGRPX
SUBROUTINES CALLED:	ATN2PI, DISTF
CALLED BY:	PLANTMIS

Method

For missile plans, all data except launch time and flight time are copied from the STRKFILE. Subroutine TIMELNCH computes the actual launch times and flight times, using the card data read in by subroutine LNCHDATA and the fixed assignment information on the MSLTIME file.

The main user card input parameters for missile timing are the minimum flight time (FLTMIN) and the coordination time for missiles (CORMSL). FLTMIN is the minimum flight time for a missile type. All flight times less than FLTMIN will be raised to FLTMIN before any timing calculations are made.

A CORMSL may also be specified for each missile type. This parameter will control the launch timing. There are two kinds of CORMSL: a "FLIGHT" CORMSL and a "LINE" CORMSL. A "FLIGHT" CORMSL is the fraction of the missile's flight which is completed at time 0.0. Clearly, such a CORMSL must lie between 0 and 1. If it is 0 the missile is launched at t = 0. If it is 1.0 the missile impacts at t = 0. The "LINE" CORMSL requires another user input. The user specifies a sequence of straight line segments (not necessarily connected). The "LINE" CORMSL is then the time at which the missile first crosses any line. If the flight path does not cross any line, then the missile will impact at time t = 0.0. These CORMSL features are operative only for initial strikes (INITSTRK = 1).

462-546 O - 72 - 34

Subroutine TIMELNCH, then, executes as follows.

First, it raises all flight times which are below the specified minimum flight time (FLTMIN) to the minimum. The subroutine then determines whether the next target processed is the first target assigned to a missile or a later target assigned to a MIRV payload. This information is needed since, if there are several targets assigned to a missile and more than one have fixed time assignments, only the first fixed time assignment encountered will be considered. Thus, if a previous fixed time assignment has determined the launch time for the missile, no further calculations need be done to compute the launch time for later re-entry vehicles on the missile. If there are no fixed assignments (with timing) on a missile with MIRV payload, the launch time is computed by considering only the data for the target assigned to the first re-entry vehicle on the booster.

The program then checks the missile to see if it is a weapon that was fixed with timing in program ALOC (statements 1000-1100). If so, the launch time specified in ALOC is used (statement 1300).

If the weapon is not fixed, TIMELNCH checks the plan type. If the strike is retaliatory (INITSTRK = 2) the complicated time plan is ignored and the launch time is the time specified by POSTALOC. If INITSTRK = 1 there are two options. If the missile type has a FLIGHT CORMSL the launch time is computed in statement 3000 so that the fraction of the flight specified by CORMSL is completed at time zero. If the missile type has a LINE CORMSL the situation is more complex.

The 2300 block of statements calculates whether the missile flight path crosses one of the timing lines input in subroutine LNCHDATA. The method for determining the intersection is explained in the Analytical Manual for the Plan Generation subsystem. The 2300 block computes vector cross products to find possible intersections.

If the missile crosses a line, the launch time is computed so that the missile crosses the timing line at time equal to CORMSL. If the missile fails to cross any line, the launch time is chosen so that the missile will impact at time zero.

For a description of the variables contained in arrays BLOCK/LOCK and TDATA/ITDATA, refer to discussion of subroutine PLANTMIS.

Subroutine TIMELNCH is illustrated in figure 203.

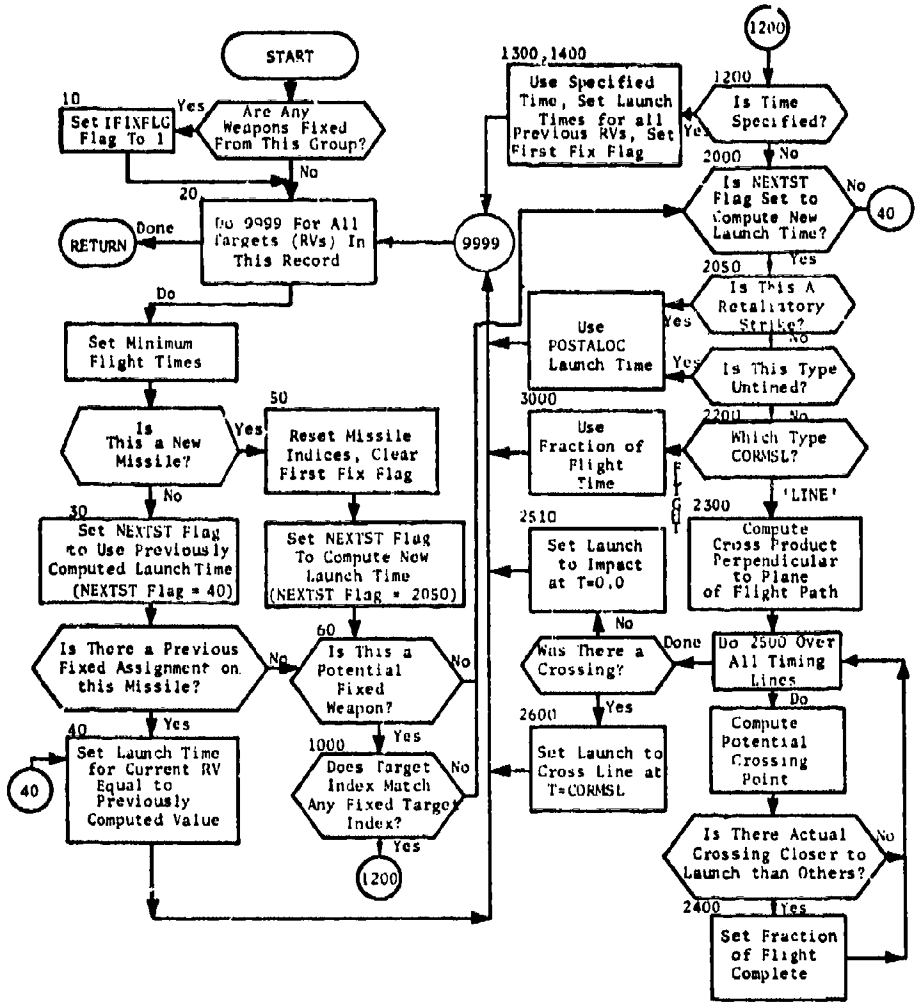


Fig. 203. Subroutine TIMELNCH

SUBROUTINE VAM

<u>PURPOSE:</u> To solve the transportation problem of allocating available tankers to bomber refuels so as to minimize the total tanker miles flown.

<u>ENTRY POINTS:</u> VAM

<u>FORMAL PARAMETERS:</u> None

<u>COMMON BLOCKS:</u> ARTIME, 9

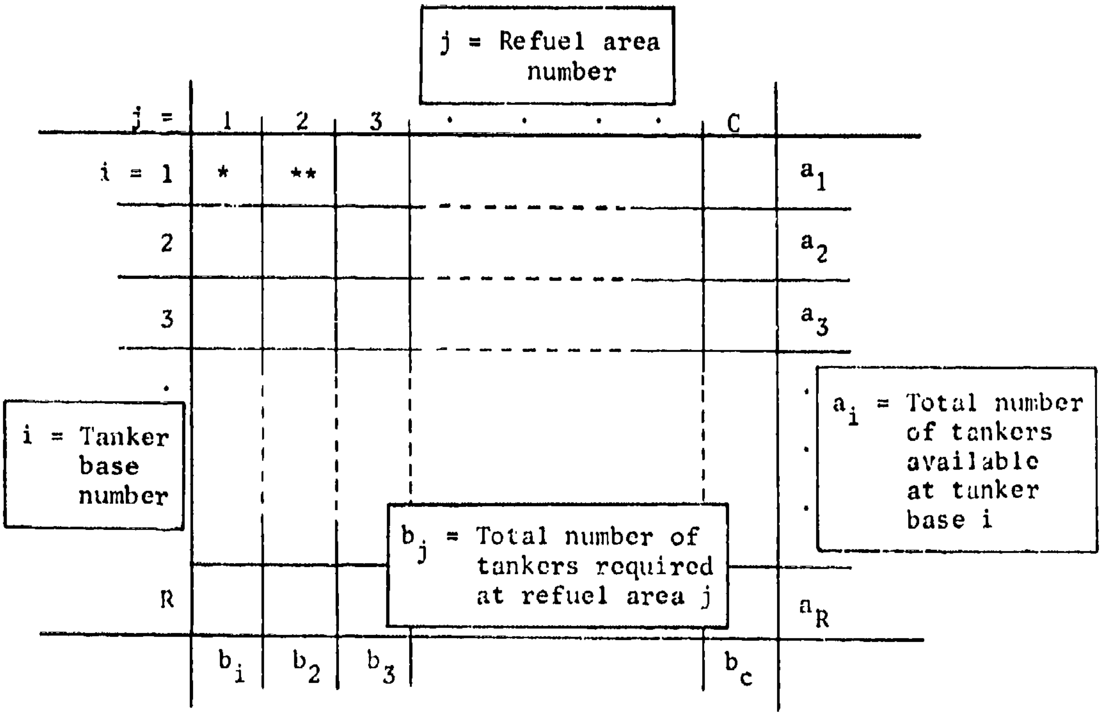
<u>SUBROUTINES CALLED:</u> None

<u>CALLED BY:</u> PLANTANK

Method

The task of allocating tankers to refuel areas in such a way as to service all bomber refuel requirements is considered by subroutine PLANTANK to be a form of the classic transportation problem. Hence, it structures the data as such in common /9/, and calls subroutine VAM to apply Vogel's Approximation Method to obtain a solution.

A general statement of the problem involves the following variables:



* COST(1,1) / X(1,1)   ** COST(1,2) / X(1,2)

Each cell in the above table has two entries associated with it:

1. COST (i,j) = distance from base i to refuel area j + safety factor of .5 miles.
2. X (i,j) = number of tankers at base i to be assigned to refuel area j.

The cost matrix is an input to the algorithm; the X matrix is its solution.

The statement of the transportation problem to be solved is:

<u>Given</u>: all i, j, a_i, b_j, and COST (i,j),

<u>Find</u>: all X (i,j) such that the total number of tanker miles flown

$$\left(\sum_{i=1}^{R} \sum_{j=1}^{C} \left[\mathrm{COST}(i,j) * X(i,j) \right] \right)$$

is minimized, subject to the constraints that

1. the total number of tankers assigned from base i must equal the total number of tankers available at base i

$$\left(\sum_{j=1}^{C} X(i,j) = a_i \text{ for } 1 \leq i \leq R\right)$$

2. the total number of tankers assigned to refuel area j must equal the total number required at refuel area j

$$\left(\sum_{i=1}^{R} X(i,j) = b_j \text{ for } 1 \leq j \leq C\right)$$

A dummy refuel area is created to handle extra tankers, which are later reassigned by subroutine PLANTANK.

The FORTRAN labels used in VAM rather than the symbols above are:

Table Symbol	FORTRAN Name
C	CMAX
R	RMAX
a	SOURCE(I)
b	SINK(J)
COST(i,j)	COST(I,J)
X(i,j)	ISOL(K), where X(RBASLOC(I),CBASLOC(J))=ISOL(K)

The solution is found using Vogel's Approximation Method, a standard operations research technique. The steps of the procedure are:

1. For each row and column in the COST matrix, calculate the difference between the smallest and next-smallest entry (row and column penalties).
2. Select the row or column with the largest difference.
3. Allocate as many tankers as possible to the smallest COST cell in that row at column.
4. Allocate zero elsewhere in the row or column where the supply (tankers) or demand (refuel requests) has been exhausted.
5. Make the only feasible allocation in any rows or columns having only one cell without an allocation of tankers.
6. Eliminate all fully allocated rows and columns from further consideration. Stop if no rows or columns remain. Otherwise,
7. Begin again, using the modified COST matrix.

A nondegenerate basic feasible solution will have (CMAX+RMAX-1) non-zero allocations in the ISOL array.

Subroutine VAM is illustrated in figure 204.

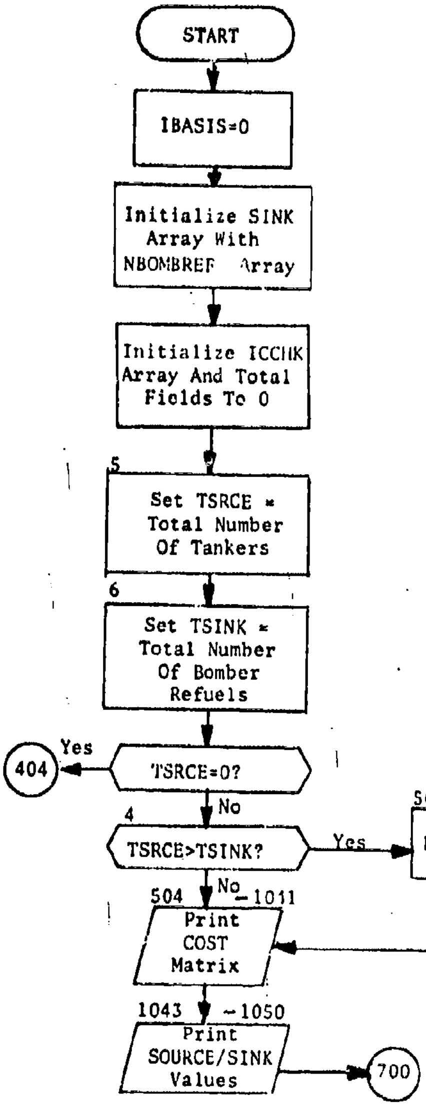


Fig. 204. Subroutine VAM
(Sheet 1 of 8)

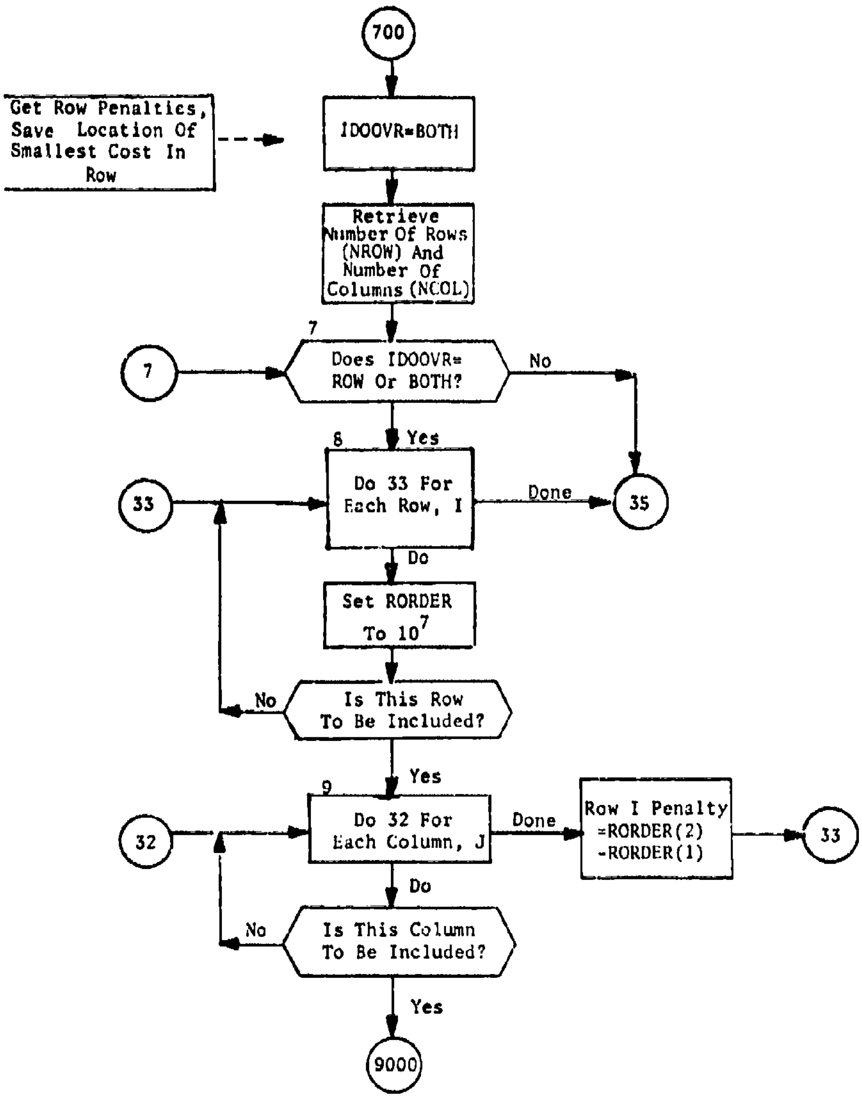


Fig. 204. (cont.)
(Sheet 2 of 8)

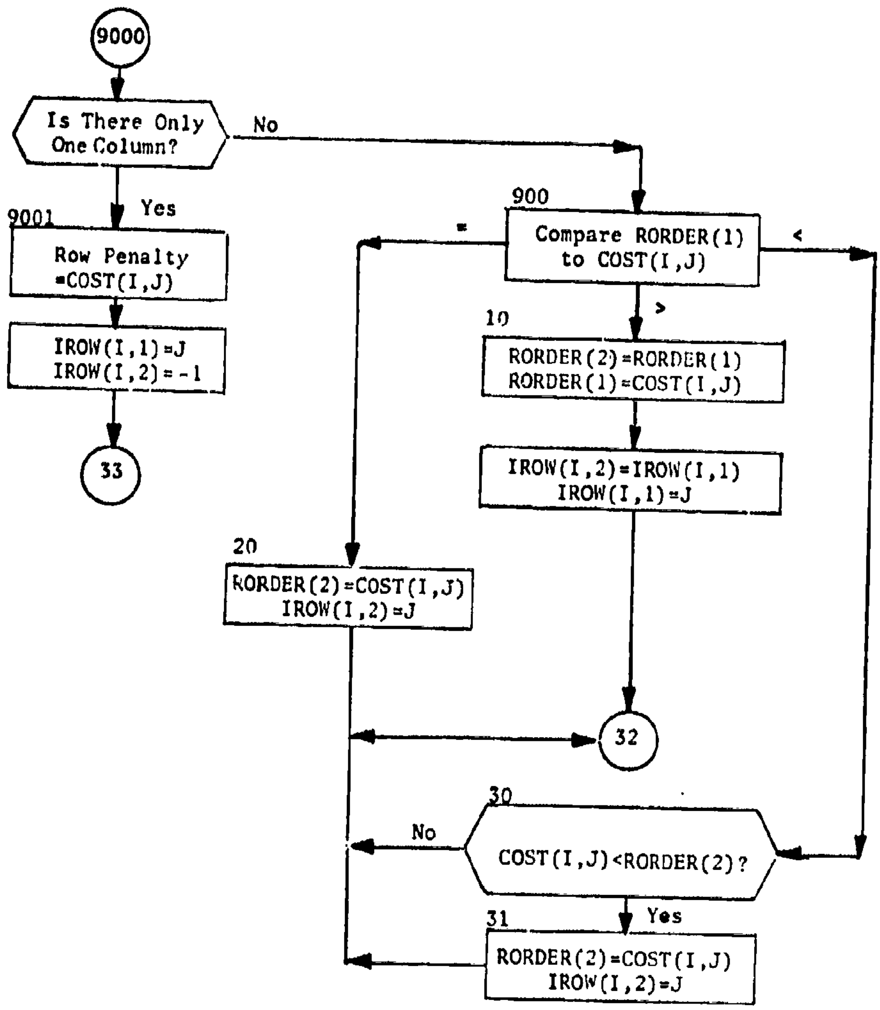


Fig. 204. (cont.)
(Sheet 3 of 8)

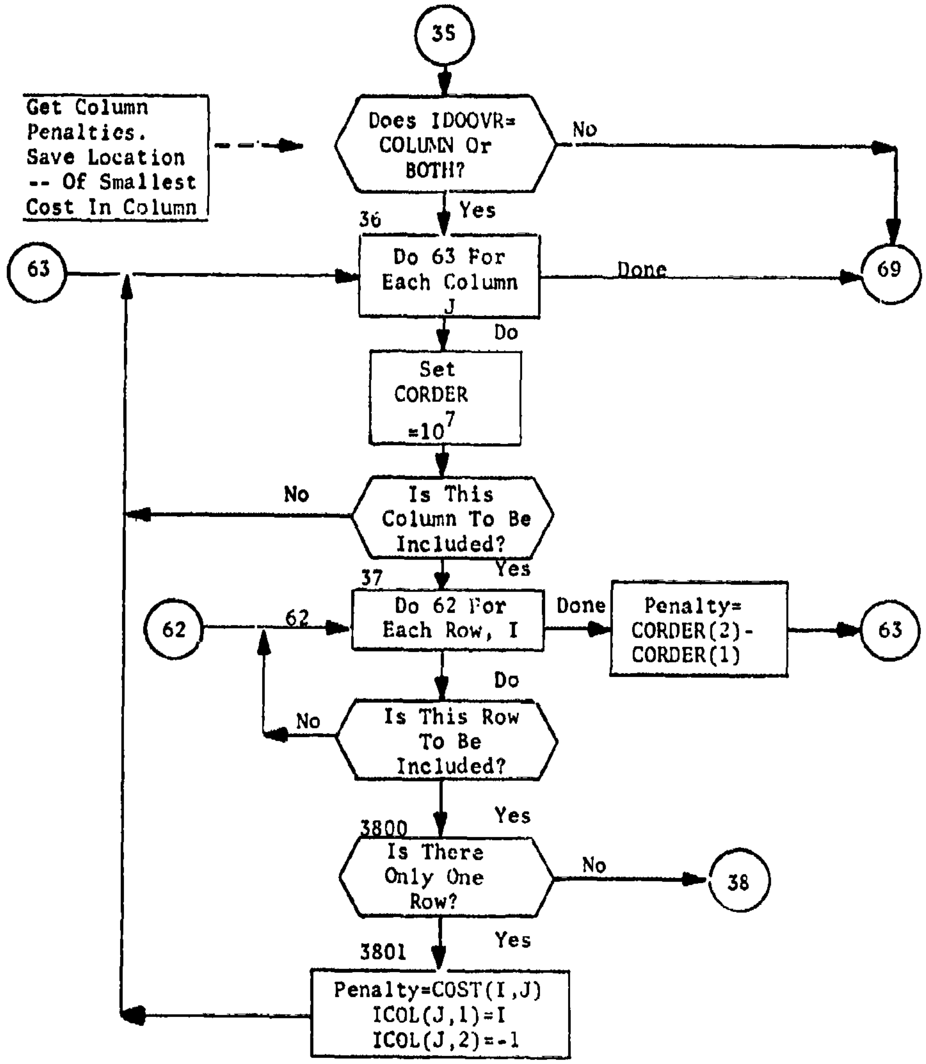


Fig. 204. (cont.)
(Sheet 4 of 8)

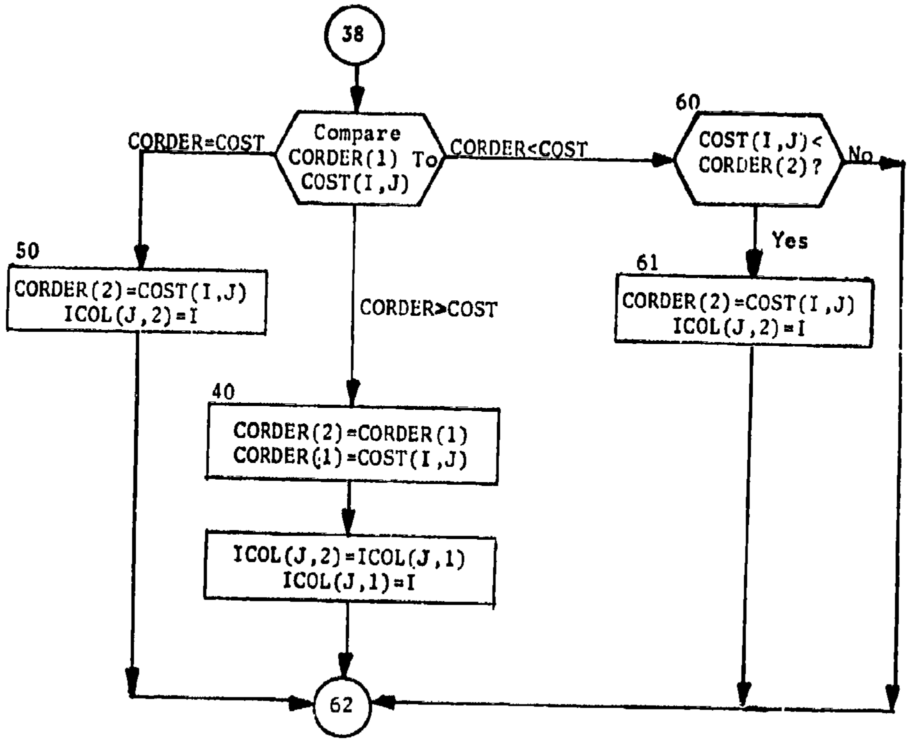


Fig. 204. (cont.)
(Sheet 5 of 8)

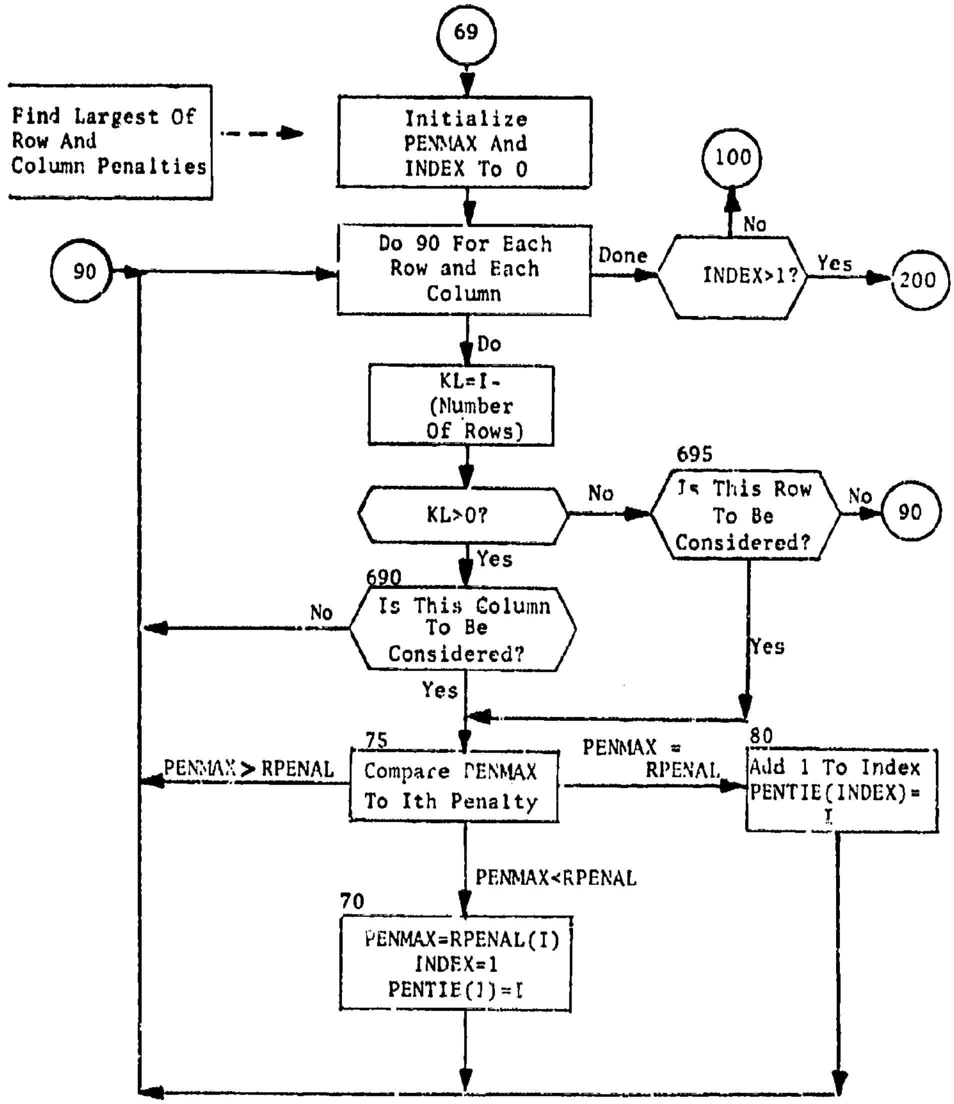


Fig. 204. (cont.)
(Sheet 6 of 8)

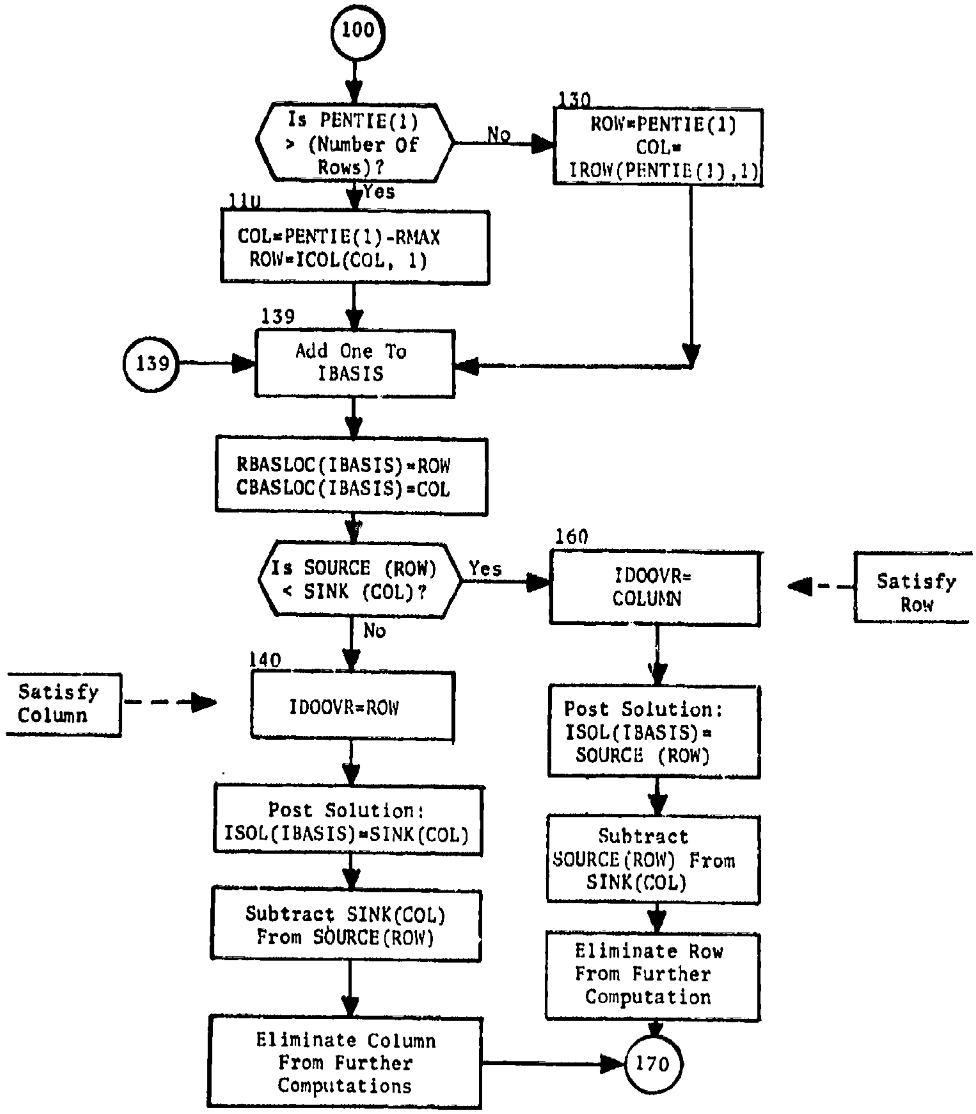


Fig. 204. (cont.)
(Sheet 7 of 8)

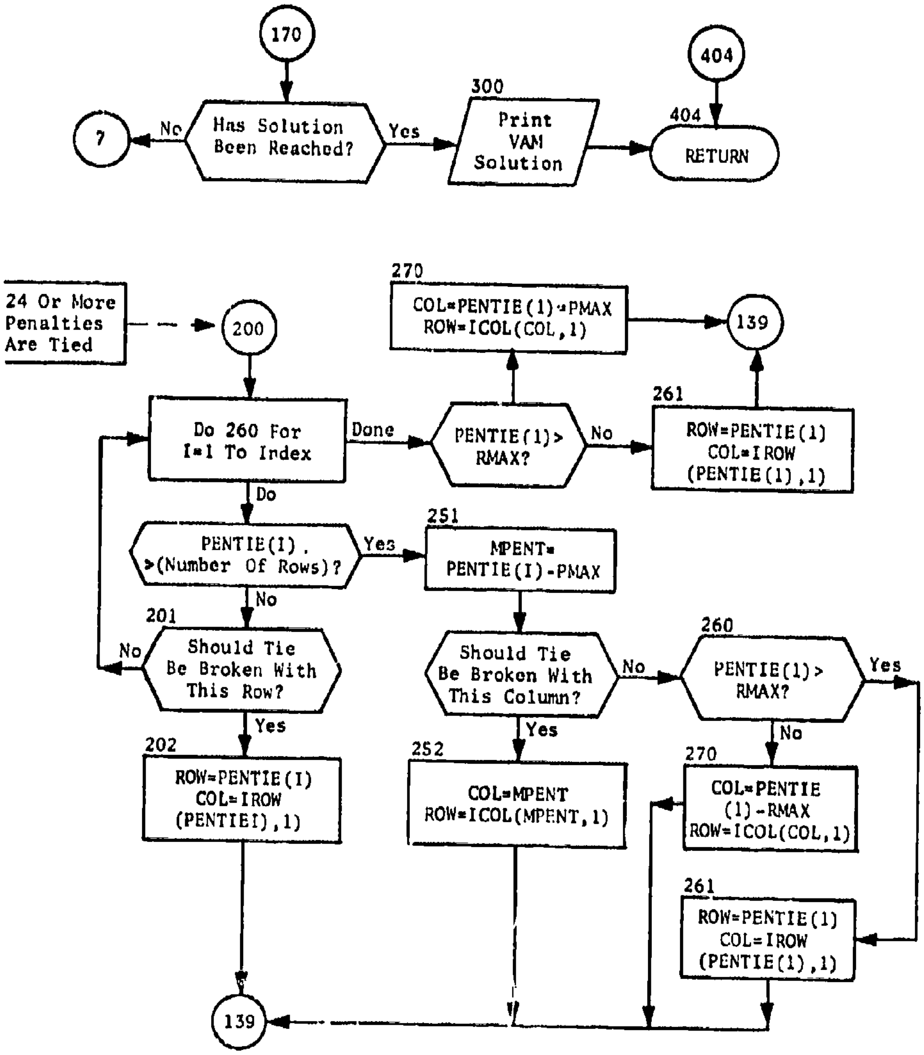


Fig. 204. (cont.)
(Sheet 8 of 8)

SUBROUTINE ZONECROS

PURPOSE:	To obtain all zone crossings which intercept a given directed line segment and post the corresponding INSECTOR events.
ENTRY POINTS:	ZONECROS
FORMAL PARAMETERS:	None
COMMON BLOCKS:	BOUND, CONTROL, DINDATA, EVENTS, FINDZONE, IFLGDPEN
SUBROUTINES CALLED:	BOUNDARY, FINDZONE, POST, POSTLAUN
CALLED BY:	PLNTPLAN

Method

Subroutine ZONECROS accepts as input the line segment from (X1,Y1), to (X2,Y2) specified in common /BOUND/ and calls on subroutine BOUNDARY to obtain the first crossing (XR,YR). It then checks to find if there is a crossing (IZIT ≠ 0). If there is one, it posts it, records this crossing as the beginning of the next line segment, and begins again by calling BOUNDARY. This is repeated until no crossing is found. Special handling is required, however, when both the zone number and depenetration flag (IFLGDPEN) are zero.

Note that the defense zone in which the corridor origin is located is specified to PLNTPLAN in common /CORRCHAR/. Consequently, the bomber path is processed both backward from this point to the corridor entry using subroutine BOUNDARY directly, and forward to the recovery base using ZONECROS.

Subroutine ZONECROS is illustrated in figure 205.

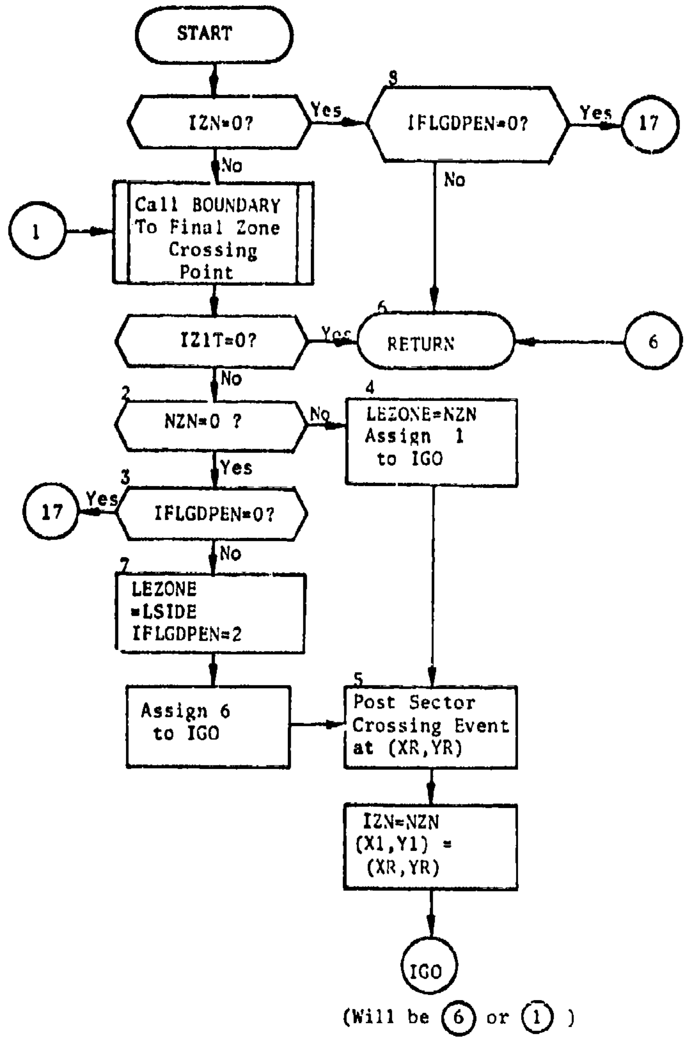


Fig. 205. Subroutine ZONECROS
(Sheet 1 of 3)

462-546 O - 72 - 35

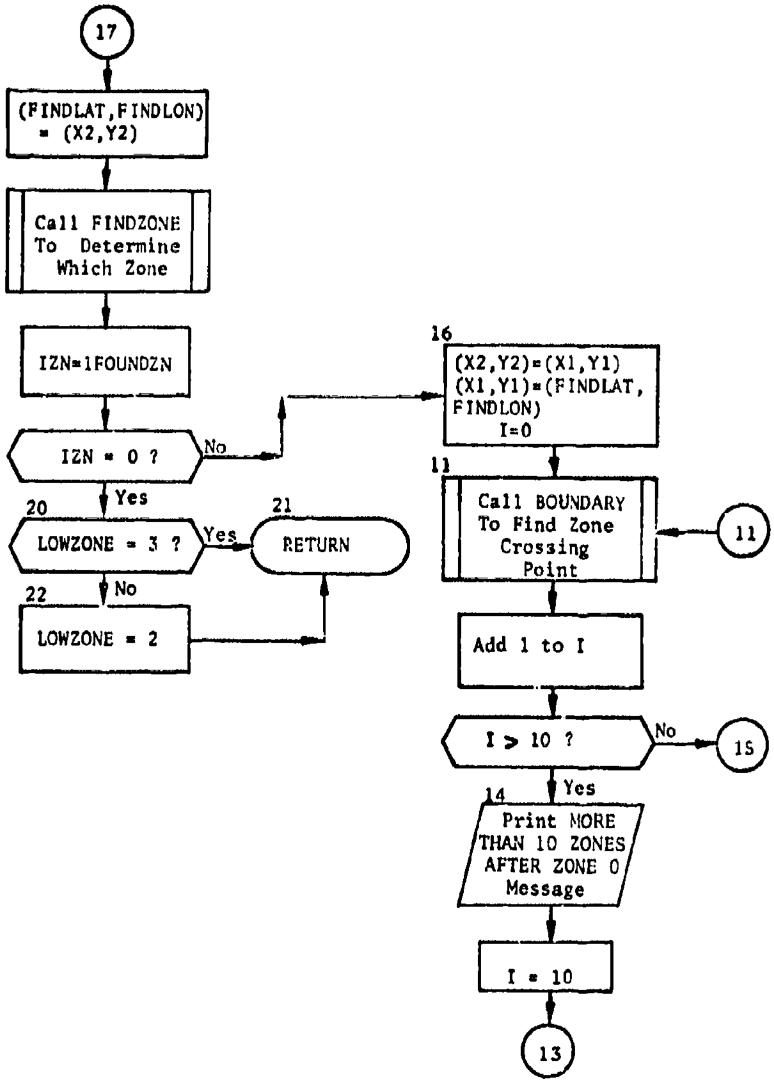


Fig. 205. (cont.)
(Sheet 2 of 3)

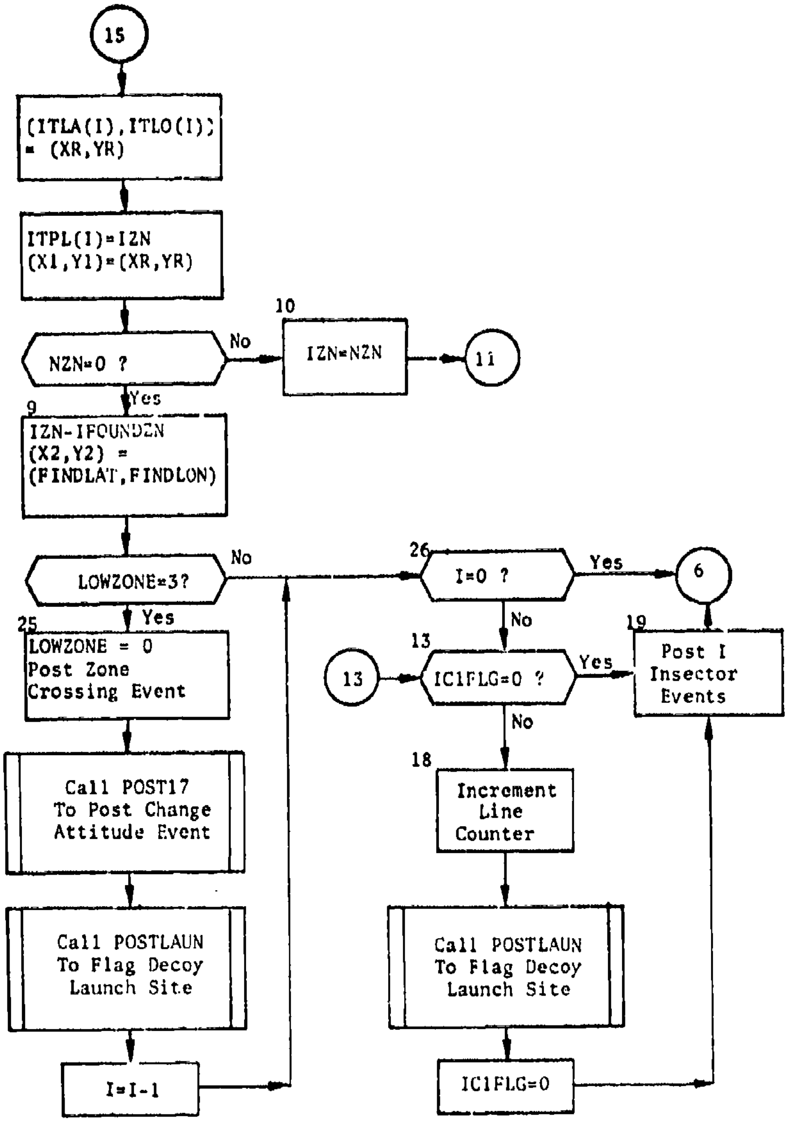


Fig. 205. (cont.)
(Sheet 3 of 3)

CHAPTER 9
PROGRAM EVALALOC

PURPOSE

The purpose of program EVALALOC is to summarize the planned allocation of weapons to targets and provide an expected-value estimate of the results. Provision is also included to evaluate the allocation for variations in the values assigned selected parameters (planning factors) associated with the weapons and targets. EVALALOC may be run at two stages of plan development, either before program ALOCOUT or after program PLNTPLAN. If run prior to the selection of desired ground zeros (DGZs) for complex targets (accomplished in ALOCOUT), the analysis of aim point offsets is not included. In this case, the results produced by EVALALOC represent an upper limit estimate which assumes that each target element in a complex is directly targeted. When EVALALOC is run after program PLNTPLAN, the weapon aim points offsets are available and are included in the expected-value computations.

INPUT FILES

When program EVALALOC is run before program ALOCOUT, the input files are the BASFILE prepared by program PREPALOC and the ALOCTAR file prepared by program ALOC. When run after program PLNTPLAN, the PLANTAPE produced by PLNTPLAN is also required as input.

OUTPUT FILE

Program EVALALOC does not produce an output file for use by later processors; its sole output is a set of summaries which present the expected-value results of the planned weapon allocation. A detailed description of these summaries is contained in the EVALALOC section of Chapter 3, Plan Generation Subsystem, User's Manual, Volume II.

CONCEPT OF OPERATION

Program EVALALOC processes the targets one at a time. For each target (or target element of a complex target), the weapons assigned are read in and ordered by time of arrival. Surviving target values are calculated, utilizing the same damage functions used in program ALOC (subroutine WAD), except that correlations are ignored. After the survival probability of each target is computed, the target and the assigned weapons are classified for summarization purposes. When all targets have been processed, the expected-value results are summarized and printed.

The initial pass over the target system always produces an evaluation based on the same weapon and target parameters used in program ALOC. Subsequent passes may be made to investigate the sensitivity of the results to changes in these weapon and/or target parameters.

When program EVALALOC is run prior to program ALOCOUT, the weapon allocation data are obtained directly from the ALOCTAR file (ALOCTAR reflects the allocation by target). When EVALALOC is run in the post-PLNTPLAN mode, the weapon allocation data are obtained from the PLANTAPE, but cannot be used directly. The PLANTAPE reflects the allocation by sortie; i.e., each block of data describes a delivery vehicle which transports warheads to one or more targets. Consequently, in this mode of operation, EVALALOC must first process the PLANTAPE and construct from it a file which reflects the allocation by target.

The program consists of the main executive program EVALALOC, a summarizing, data handling, and print subroutine EVAL2, and a computing subroutine EVALPLAN. EVALALOC includes provisions for exploring the sensitivity of the results to the assumed or calculated values of some of the weapon or target planning parameters. The program can be recycled and these parameter values can be varied using subroutines WPNMODIF and TGTMODIF.

As indicated above, when EVALALOC is run after PLNTPLAN, it is necessary to prepare a new file on which the weapon-to-target allocation is target oriented. Because of the large amount of data which must be stored to describe the allocation, it is necessary to pack several items of information in each word. This is done by the four packing subroutines, PACK, BOMRPAKR, MISLPAKR, and UNPACKER. In the event that the pre-PLNTPLAN operation is prescribed, the packing routines are never called and the allocations are read from ALOCTAR.

Program EVALALOC (figure 206) reads the user's general control data card and stores the input parameters ITGTMAX, JOPT, and PREFABRT in common block /OPT/, the parameter PKTX in common block /MIS/, and the parameter LAW(1), LAW(2) in common block /LAW/. If ITGTMAX is negative, it stops the run. Otherwise, it initializes the filehandler and reads the

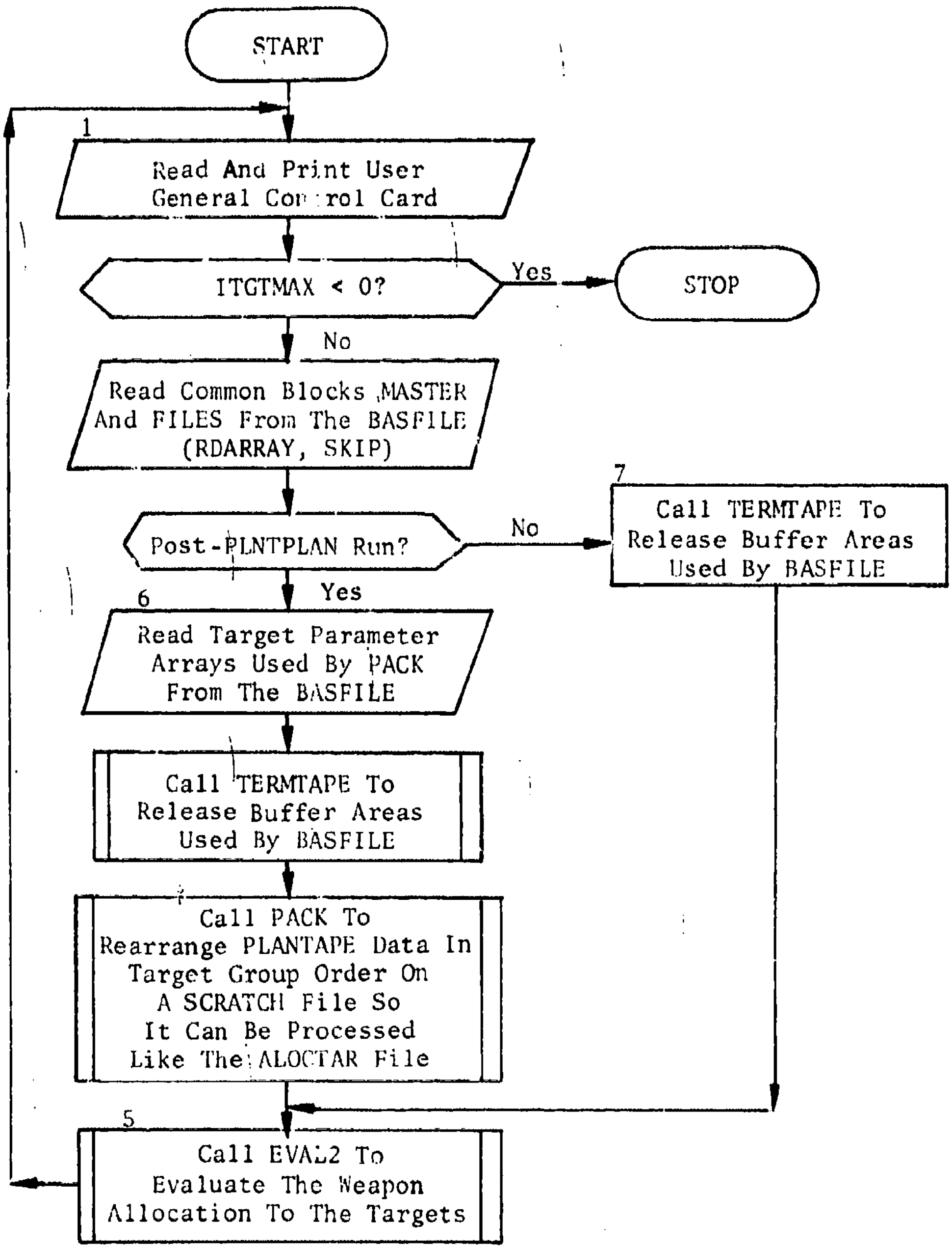


Fig. 206. Program EVALALOC

/FILES/ and /MASTER/ common blocks from the BASFILE. If the run is post-PLNTPLAN it also reads the REL, ITYPE, and DBL arrays into common block /WPNMOD/ so they can be used by subroutine UNPACKER to modify the values of the weapon penetration probability obtained from the PLANTAPE. In this mode of operation, EVALPLAN next calls subroutine PACK to reorder the weapon allocation data from the PLANTAPE. Then it calls EVAL2 to evaluate the allocation and summarize the results. In the pre-ALOCOUT mode of processing, subroutine PACK and its associated subroutines are not used, and EVALALOC proceeds directly to the EVAL2 call. After EVAL2 completes its processing, EVALALOC reads the next user general control data card and repeats the process described here until it reads a card where ITGTMAX is negative.

Subroutine PACK is used only in post-PLNTPLAN operation of EVALALOC. PACK determines the order of the targets on the ALOCTAR file and reorders the weapon-to-target allocation data on the PLANTAPE in that target order. To do this, it must pack the weapon allocation data using subroutines MISLPAKR or BOMRPAKR (depending on the weapon type). At the time of packing, BOMRPAKR and MISLPAKR use subroutine SEARCH to locate the ALOCTAR targer number associated with the target index number, INDEXNO, contained on the PLANTAPE. This number is also packed with the weapon data. When all the weapon allocation data from the PLANTAPE have been packed and written onto a scratch file in target number order by PACK, the ALOCTAR file is read again, one target at a time, and PACK uses UNPACKER to locate and unpack the weapon allocation data associated with the targets. Since these data are in target order on the SCRATCH file, the matching process is straightforward. After the allocation data is unpacked, PACK writes the target data from the ALOCTAR file and allocation data (obtained from the PLANTAPE) onto a new scratch file in a format identical with the ALOCTAR file. Hence in post-PLNTPLAN processing, this scratch file is used in place of the ALOCTAR file.

Subroutine EVAL2 controls the evaluation process in EVALALOC. It first calls subroutine WPNMODIF to modify the BASFILE weapon parameters specified by the user if this is not the first pass through EVAL2. (On the first pass no weapon or target parameter modifications are done.) Then it reads the target and weapon allocation data from the ALOCTAR file (or from the ALOCTAR-like scratch file in post-PLNTPLAN operation) one target at a time. If the target is multiple or complex, it also reads the associated target element data from the BASFILE. For each target, or complex target component, if this is not the first pass, EVAL2 calls TGTMODIF to perform user-specified modifications on target parameters; for all passes it then calls EVALPLAN to determine the amount of damage done to the target and to store the results in the various arrays in common blocks /LITTLE/ or /2/ to be used later in EVAL2 when it produces the printed output summaries. EVALPLAN uses the functions SSKPC, SSPCALC, and DIST to determine the probability of kill for a target, the target survival probability given a multiple weapon attack, and the distance between the target centroid and the point of weapon delivery. All three of these factors affect the expected-damage calculation for the target. (Note that in post-PLNTPLAN operation, TGTMODIF and EVALPLAN are called once for each multiple target element.) When all the targets and associated weapon allocation data have been read and evaluated, EVAL2 summarizes and prints the results in tables. This completes one pass through EVAL2. The next pass through is initiated with another call to WPNMODIF to see if the user wanted to test the sensitivity of the results to other weapon parameter modifications. If he did, another pass through EVAL2 is accomplished. This process continues until EVAL2 encounters the word RETURN as the weapon parameter to be modified. Then control is returned to the main executive program, EVALALOC.

COMMON BLOCK DEFINITION

External Common Blocks

The format and content of the external common blocks used in EVALALOC is presented in table 54. The use which the program makes of each of the blocks is set forth briefly below:

1. /FILES/: This common block contains information about the BASFILE, ALOCTAR file, PLANTAPE, and SCRATCH files (which are equivalenced to the STRKFILE which is needed to use the file-handler subroutines).

2. /1/: This common block is filled from the PLANTAPE by subroutines PACK and is used by subroutines BOMRPAKR and MISLPAKR to obtain the data which they pack for each target event.

3. /TGTMOD/: In post-PLNTPLAN operation, this block is first filled by PACK when it reads the ALOCTAR file and then is filled by UNPACKER when it unpacks data for the ALOCTAR-like SCRATCH file. Then, in all cases, it is filled by EVAL2 when it reads the ALOCTAR file or ALOCTAR-like SCRATCH file. The data are modified by subroutine TGTMODIF and used during EVALPLAN evaluation.

4. /WPNMOD/: In post-PLNTPLAN operation, this block is first filled from the BASFILE by EVALALOC so the weapon parameters can be used by PACK to modify the weapon penetration

probabilities from the PLANTAPE. In all operations it is filled again from the BASFILE by subroutine EVAL2, modified by WPNMODIF, and used during EVALPLAN evaluation.

5. /PLANREC/: Is a convenient storage place for data read from the PLANTAPE header portion of each record and is used only by PACK.

6. /MIS/: This block contains data about terminal ballistic missile interceptors and is filled by EVALALOC (PKTX) and EVAL2. It is used by EVAL2 during the calculation of expected target damage when the target has such interceptors.

Internal Common Blocks

The format and content of the internal common blocks encountered in EVALALOC are presented in table 55. The use of these blocks by the program is set forth briefly below:

1. /LATLON/: This block is used by subroutines PACK and UNPACKER during post-PLNTPLAN operation to transfer target and weapon delivery location coordinates back and forth. It is filled in all EVALALOC runs by EVAL2, and EVALPLAN uses it during its calculation of expected target damage.

2. /LITTLE/: This block is filled by subroutine EVALPLAN which uses it to pass some weapons allocated to one target to EVAL2. EVAL2 updates the arrays used to produce the printed summaries from these data.

3. /OPT/: This block contains the user's options as contained on the general data control card. It is used in various ways by nearly all EVALALOC subroutines to carry out the functions specified by the user.

4. /LAW/: This block is filled from the general control data card by program EVALALOC and is used by function SSPCALC when it determines target survival probability for EVALPLAN.

5. /BINSCH/: This common block is used by PACK, BOMRPAKR, MISLPAKR, and SEARCH and contains parameters related to the binary search done by SEARCH to determine the target number corresponding to a given target index number.

6. /2/: As used by PACK and its associated subroutines, this block contains the packed data from the PLANTAPE. It is used by EVAL2 to store weapon data (CCREL, IREG, IALERT, IPAY, and YIELD) from the BASFILE for use by EVALPLAN to hold the arrays in which the summary data for the prints produced by EVAL2 are stored.

Table 54. Program EVALALOC External Common Blocks
(Sheet 1 of 5)

INPUT FROM BASFILE

BLOCK	VARIABLE OR ARRAY*	DESCRIPTION
FILES	TGTFILE(2)**	Target data file
	BASFILE(2)	Data base information file
	MASLTIME(2)	Fixed missile timing file
	ALOCTAR(2)	Weapon allocation by targets file
	TMPALOC(2)	Temporary allocation file
	ALOCGRP(2)	Allocation by group file
	STRKFIL(2)	Strike file
	EVENTAPE***	Simulator events tape
	PLANTAPE***	Detailed plans tape

INPUT FROM PLANTAPE

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/1/	LGRP	Weapon group index
	KORD	Corridor index number
	TMLAUN	Missile time of launch
	HDT(90)	Missile flight time or bomber/tanker time since last event
	KPL(90)	Place or site index
	JTP(90)	Missile index or event tape

*Parenthetical values indicate array dimensions. All other elements are single word variables.

**In two word arrays, first word is logical unit number; second word is maximum file length in words. Single variables are logical unit numbers.

***These files are output on magnetic tape.

Table 54. (cont.)
(Sheet 2 of 5)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/1/ (cont.)	HLA(90)	Event latitude
	HLO(90)	Event longitude
	TZT(90)	Missile weapon site latitude or bomber weapon delivery offset latitude
	TZN(90)	Missile weapon site longitude or bomber weapon delivery offset longitude
	IWH(90)	Warhead type or index
	PA(90)	Reliability/damage expectancy
	ICMT(90)	Bomber/tanker cumulative time to event
	ITGTX(90)	Missile/bomber target index number

INPUT FROM ALOCTAR

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
TGTMOD	INDEXNO	Index number of target
	VTO	Original target value
	M	Number of hardness components (≤ 2)
	H(2)	Hardness of each component
	VO(2)	Original value of each component
	NK	Number of time periods (≤ 3)
	FVAL(3)	Fraction value escaping in each period
	TAU(3)	Time ending each period
	NUM	Number of weapons assigned

Table 54. (cont.)
(Sheet 3 of 5)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
TGTMOD (cont.)	IG(30)	Group number of assigned weapons
	KORR(30)	Weapon penetration corridor
	RVAL(30)	Relative value of weapon allocation
	PEN(30)	Weapon penetration probability
	TOA(30)	Weapon time of arrival on target
*	DE(30)	Survival probability of target after arrival of weapon I
	TIMEVAL(30)	Surivival probability of time-dependent target value after arrival of weapon I
	NTYPE	Target type name (same as IHTYPE on ALOCTAR file)
	INDTYPE	Index of target type used for summarization
	VTOC	Total value of all components of complex target
	ITGTINDX	Number of targets being processed; in UNPACKER, number of target. for which PLANTAPE weapons must be found
	IBGINDX	Index number of target for which PLANTAPE weapons must be found by UNPACKER.

*The remaining segment of /TGTMOD/ is used for internal processing.

Table 54. (cont.)
(Sheet 4 of 5)

INPUT FROM BASFILE

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
WPNMOD	NTYPES	Number of weapon types
	CEP(80)	Weapon CEP (nautical miles)
	ICLASS(80)	Weapon class index (1 for missile, 2 for bombers)
	REL(80)	Weapon reliability
	NAMEWPN(80)	Weapon type name
	FUNC(80)	Weapon function code
	ITYPE(200)	Weapon type index
	DBL(200)	Probability that the weapon survives before launch
	NPASS	Number of times the program runs
MIS	NWHDS(40)	Number of type 1 bombs
	NDECOYS(40)	Number of terminal decoys
	MISDEF	Number of terminal ballistic missile interceptors (minus the number of interceptors if a DEFALOC allocation) (ALOCTAR file)
	PKTX	Probability of kill of a terminal ballistic missile interceptor

INPUT FROM PLANTAPE

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
PLANREC	JSIDE	Side
	LGROUP	Group index
	LPEN	Penetration corridor number

Table 54. (cont.)
(Sheet 5 of 5)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
PLANREC (cont.)	JSORTIE	Missile record counter or bomber/tanker sortie number
	JUNIT	Zero or base index number
	JVEHIC	Zero or vehicle index number
	LCLAS	Weapon class (1, 2, or 3)
	JWPNTYPE	Weapon type index
	JREG	Zero or launch region
	JALERT	Alert status
	JPAYLOAD	Zero or payload index
	JDEPEN	Zero on depenetration corridor index
	LTOT	Number of missiles or total number of events in table
	LPLAN	Number of targets or total number of planned events
	G(1)	Missile time of launch
	G(2), G(3)	Not used
	MLO(2)	Lower plot markers for sortie
	MHI(2)	Upper plot markers for sortie
	JHTYPE	Warhead type name
	JPTYPE	Weapon (Plan Generator) type index
	JFUNC	Weapon function code
	JLAST	End sentinel or not used

Table 55. Program EVALALOC Internal Common Blocks
(Sheet 1 of 5)

BLOCK	VARIABLE OR ARRAY*	DESCRIPTION
LATLON	BLAT	Latitude of a burst event
	BLON	Longitude of a burst event
	TGTLAT	Target latitude
	TGTLON	Target longitude
LITTLE	INDCLAS	Summarization index used for target classes
	SURV	Survival probability of a target
	TGTRAD	Target radius
	TMULT	Original target multiplicity (for multiple targets)
	VALDES	Value of target destroyed
	VALESC	Value of target escaping during attack
	CTMULT	Current target multiplicity (for multiple target)
	PLANYLD	Total yield scheduled to a target
	DELYLD	Total yield delivered to a target
	NOWPNS	Number of weapon categories
	VALFAC	Fractional value of a complex target component referred to total value of complex
OPT	JOPT(20)	Logical array which controls printing and type of operation
	ITGTMAX	Number of targets for which detailed print is given
	PREFABRT	Probability of refueling abort

*Parenthetical values indicate array dimensions. All other elements are single word variables.

Table 55. (cont.)
(Sheet 2 of 5)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
OPT (cont.)	JSKIP	Indicator which contains program control information depending on user's input option
LAW	LAW(2)	Hollerith name of damage function option
BINSCH	NMTGTS	Total number of targets for binary search
	JREAD	Indicator equal to 0 on first call to SEARCH, 1 thereafter
	NLOG	Maximum number of iterations in binary search to find target number
	INT	Initial length of binary search interval
	NOW	Initial index of entry in JORDER array to be tested: dummy search
	JORDER(6144)	Array containing words for binary search - each word contains a target index number and its associated target number
/2/		As used by PACK, BOMRPAKR, MISLPAKR, UNPACKER
	LINK	Index to next words in JLINK, KLINK, and LLINK arrays to be packed or unpacked
	ILINK(12000)	Equivalenced to JLINK(4000), KLINK(4000), and LLINK(4000)

462-548 O - 72 - 36

Table 55. (cont.)
(Sheet 3 of 5)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/2/ (cont.)	JLINK(4000)	Packed array; each word contains target index number, target number, weapon group index, and corridor index
	KLINK(4000)	Packed array; each word contains weapon penetration probability, and time of target event
	LLINK(4000)	Packed array; each word contains latitude and longitude of burst event
/2/		As used by EVAL2 and EVALPLAN
	CCREL(20)	Command and control reliability (from BASFILE)
	IREG(20)	Weapon region index (from BASFILE)
	IALERT(200)	Weapon alert status 1 for alert, 2 for nonalert; (from BASFILE)
	IPAY(200)	Refuel code for bombers or payload index for missiles (from BASFILE)
	YIELD(200)	Weapon yield in megatons (from BASFILE)
	CLASTYPE(200)	Summarization index for aggregating types into classes
	NALLTYPE(I,J)(7,200)	Number of weapons of category I scheduled against targets of type J
	SKDWPTYP(I,J)(7,200)	Number of weapons of category I scheduled against targets of type J

Table 55. (cont.)
(Sheet 4 of 5)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/2/ (cont.)	DELWPTYP(I,J)(7,200)	Yield from weapon of category I against targets of type J
	ALLTYPE(I,J)(7,200)	Number of weapons of category I delivered to targets of type J
	NALLCLAS(I,J)(7,50)	Number of weapons of category I scheduled against targets of class J
	SKDWPCL(I,J)(7,50)	Yield from weapon of category I scheduled against targets of class J
	DELWPCL(I,J)(7,50)	Yield from weapon of category I delivered against targets of class J
	ALLCLAS(I,J)(7,50)	Number of weapons of category I delivered against targets of class J
	NAMECLAS(I)(50)	Name of class I
	NOFCLAS(I)(50)	Number of targets in class I
	VOCLAS(I)(50)	Total target value for class I
	VDESCLAS(I)(50)	Time-dependent value of target destroyed in class I
	VESCCLAS(I)(50)	Time-dependent value of target escaped in class I
	VREMCLAS(I)(50)	Time-dependent value of target remaining in class I
	SURVCLAS(I)(50)	Percent of target value surviving in class I
	SKEDCLAS(I)(50)	Megatons scheduled for target class I
	DELCLAS(I)(50)	Megatons delivered to class I targets
	DESTNOCL(I)(50)	Value of target destroyed in class I

Table 55. (cont.)
(Sheet 5 of 5)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/2/ (cont.)	PCDESTCL(I)(50)	Percent of target value destroyed in class I
	NSUMCLAS(I)(50)	Number of weapons allocated to targers in class I
	SUMCLAS(I)(50)	Number of weapons delivered to targets in class I
	KCLSNGL(I)(50)	Number of targets in class I attacked alone
	KCLSCOMP(I)(50)	Number of targets in class I attacked as part of a complex
	SKEDALL(J)(7)	Total megatonnage scheduled for weapon category J
	DELALL(J)(7)	Total megatonnage delivered from weapon category J
	NAMETYPE(K)(200) . . . KTYPCOMP(K)(200)	The definitions of these arrays exactly parallel those of arrays NAMECLAS through KCLSCOMP except that they are for target type K (instead of class I)
	NWPNTYP(J)(7)	Total number of weapons scheduled from category J
	XWPNTYP(J)(7)	Total number of weapons delivered from category J

SUBROUTINE BOMRPAKR

PURPOSE:	To pack the weapon allocation data for each bomber target event into three words
ENTRY POINTS:	BOMRPAKR
FORMAL PARAMETERS:	LLL - Index of entry in KPL array which contains the target index number
COMMON BLOCKS:	1, 2, OPT
SUBROUTINES CALLED:	SEARCH
CALLED BY:	PACK

Method

BOMRPAKR sets INDX equal to the target number which is KPL(LLL), and uses SEARCH to find the target number ITGT on the ALOCTAR file to which this index number corresponds. Then it determines the index, I, of the next free words in the JLINK, KLINK, AND LLINK arrays where the packed weapon allocation data are stored. Finally, it packs ITGT, INDX, weapon group number (LGRP), and corridor number (KORD) into JLINK(I), the penetration probability (IPEN), and time of weapon arrival (ITIME) into KLINK(I), and the weapon delivery latitude (LAT) and longitude (LON) into LLINK(I). Note that all of these weapon parameters have been integerized before packing. The locations of the packed values are illustrated in figure 207.

If the user has set the debug print option (JOPT(3)) to 1, BOMRPAKR prints all the parameters being packed as well as the packed words, KLINK(I) and LLINK(I).

Subroutine BOMRPAKR is illustrated in figure 208.

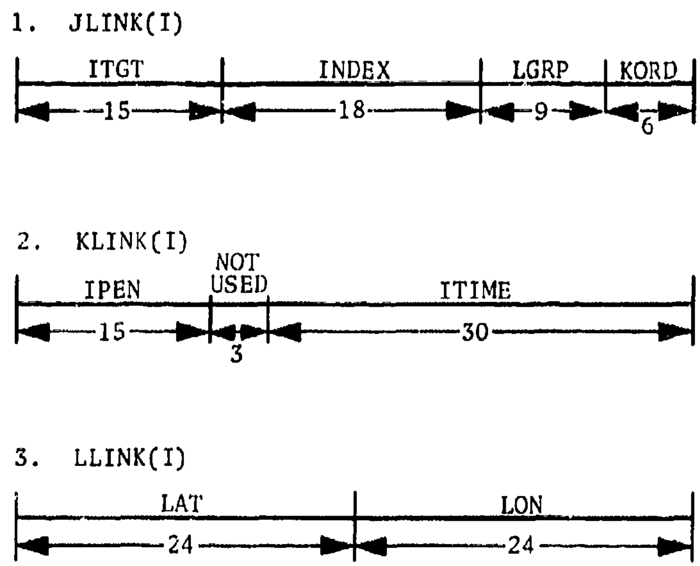


Fig. 207. Location of Packed Values in Subroutine BOMRPAKR

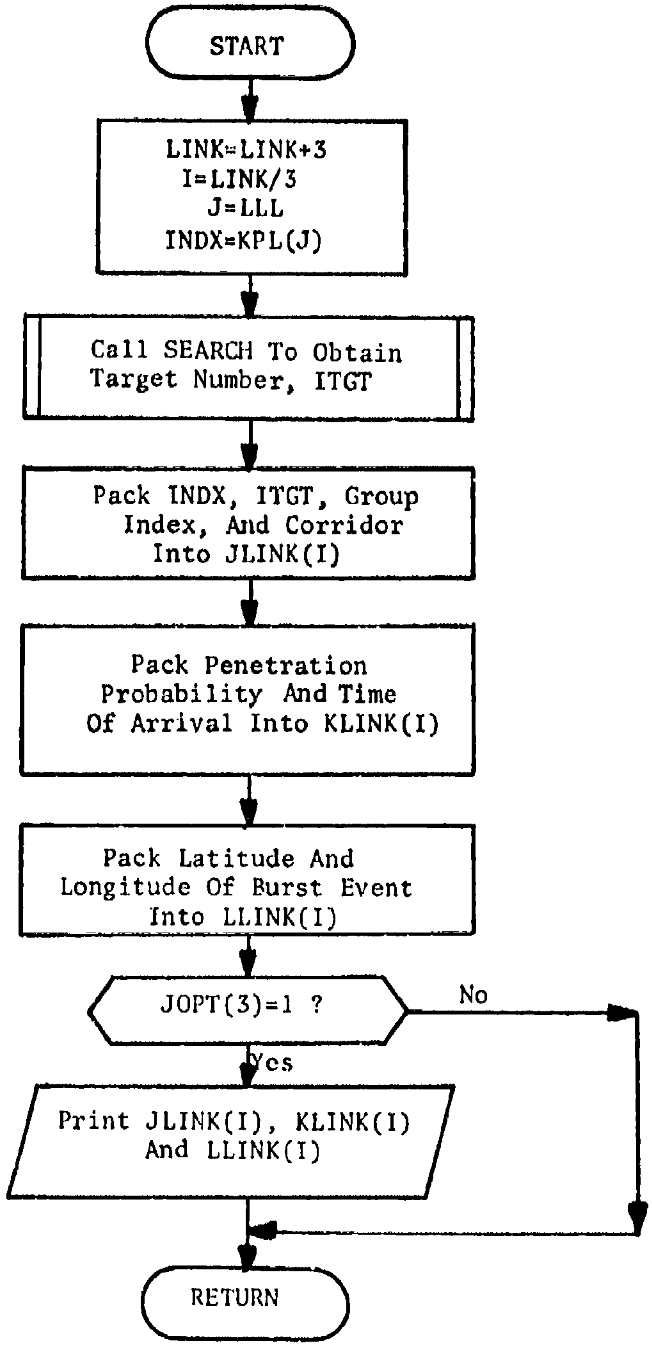


Fig. 208. Subroutine BOMRPAKR

FUNCTION DIST

PURPOSE:	To calculate the distance between the target centroid and a point near the centroid at which a given weapon is aimed.
ENTRY POINTS:	DIST
FORMAL PARAMETERS:	TLT1 - Target latitude TLON1 - Target longitude TLTZ - Latitude of aim point TLONZ - Longitude of aim point
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	EVALPLAN

Method

DIST (See figure 209) computes the distance between the target centroid and weapon aim point using the following formula:

$$DIST = (TLT1 - TLT2)^2 + (TLON1 - TLON2)^2$$

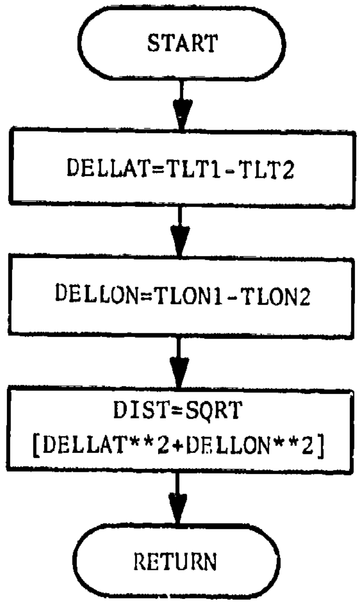


Fig. 209. Function DIST

SUBROUTINE EVALPLAN

PURPOSE: To classify the weapons allocated to each target and compute the corresponding change in target value after the attack

ENTRY POINTS: EVALPLAN

FORMAL PARAMETERS: None

COMMON BLOCKS: LATLON, LITTLE, MIS, OPT, TGTMOD, WPNMOD, 2

SUBROUTINES CALLED: ABORT, DIST, INITPROB, SSKPC, SSPCALC

CALLED BY: EVAL2

Method

EVALPLAN is called once for each target to consider the damage done by all weapons allocated to the target. When called, internal arrays are initialized and the number of target value components is checked. If the target has more than three value components an error message is printed and control is returned to the calling program.

If this is not a first pass (initial) evaluation and EVALALOC is being run after program ALOC, EVALPLAN determines if the target is defended by effective terminal ballistic missile interceptors. If so, EVALPLAN recomputes the penetration probability for each missile allocated to the target before performing target survival calculations.

If this is a first pass, EVALPLAN classifies each weapon into one of seven categories: alert LRA, nonalert LRA, TAC, SLBM, MRBM, IRBM or ICBM. It then updates entries in the following arrays in common block /2/: NALLCLAS, NALLTYPE, ALLCLAS, ALLTYPE, SKDWPCL, SKDPTYP, DELWPCL, and DELWPTYP.

For each weapon, EVALPLAN updates the DELYLD and PLANYLD parameters in common block /LITTLE/ for all passes of EVALALOC and calculates the value FVALTOA of the target at the weapon time of arrival. Then it uses functions DIST and SSKPC and subroutine SSSPCALC to calculate kill and survival probabilities SSK and SSS. These probabilities are subsequently used to compute values for PRODSS, a hardness component probability factor, CUMDES, the value of each hardness component destroyed, and CUMESC, the value of each hardness component escaping during the attack.

The values of DE and TIMEVAL in common block /TGTMOD/ are then calculated using these variables. Finally, EVALPLAN calculates the target survival probability (SURV), the total target value destroyed (VALDES), and the total target value escaping during the attack (VALESC). These results are later saved in subroutine EVAL2 by index of target type INDTYPE.

Subroutine EVALPLAN is illustrated in figure 210.

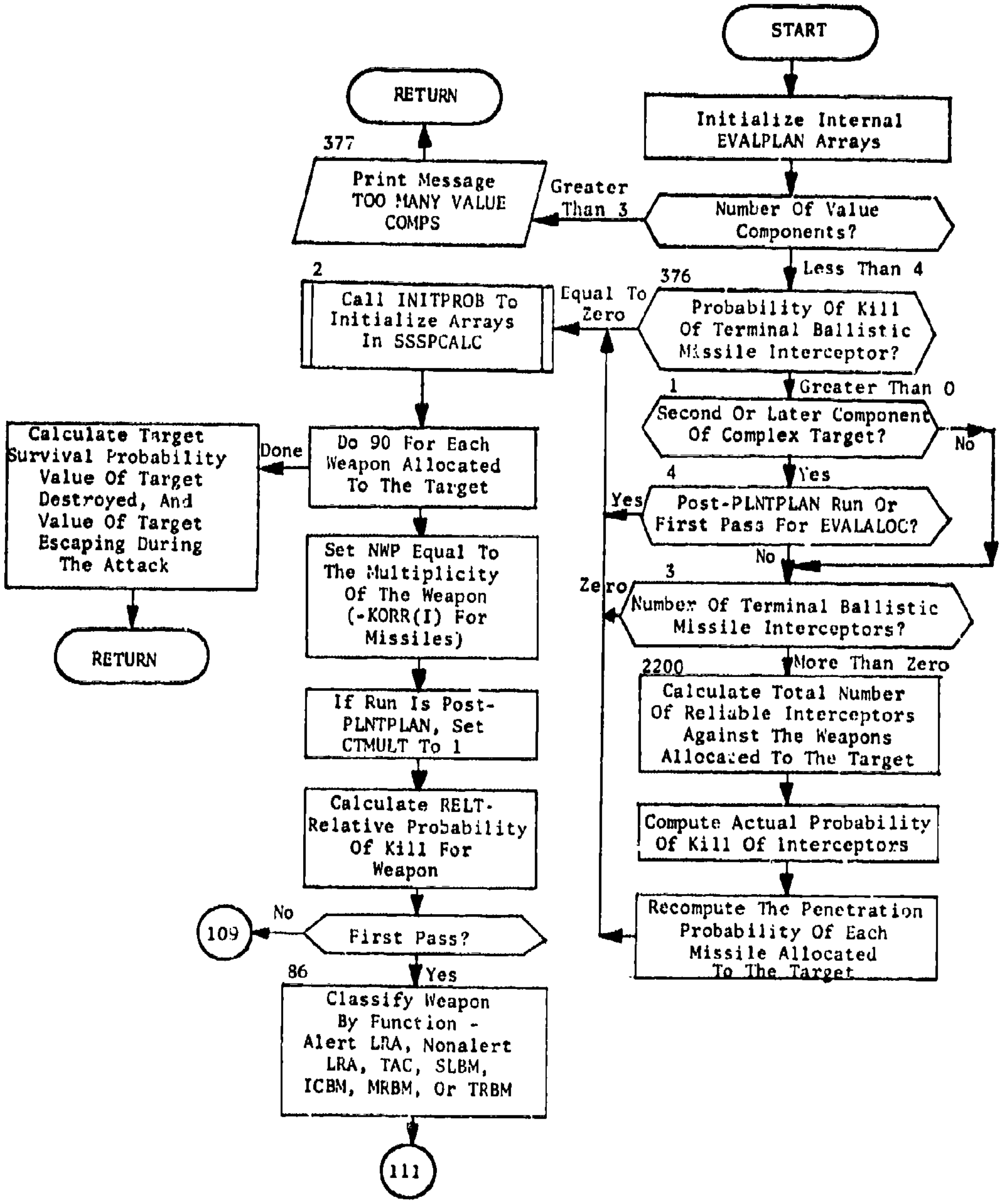


Fig. 210. Subroutine EVALPLAN
(Sheet 1 of 2)

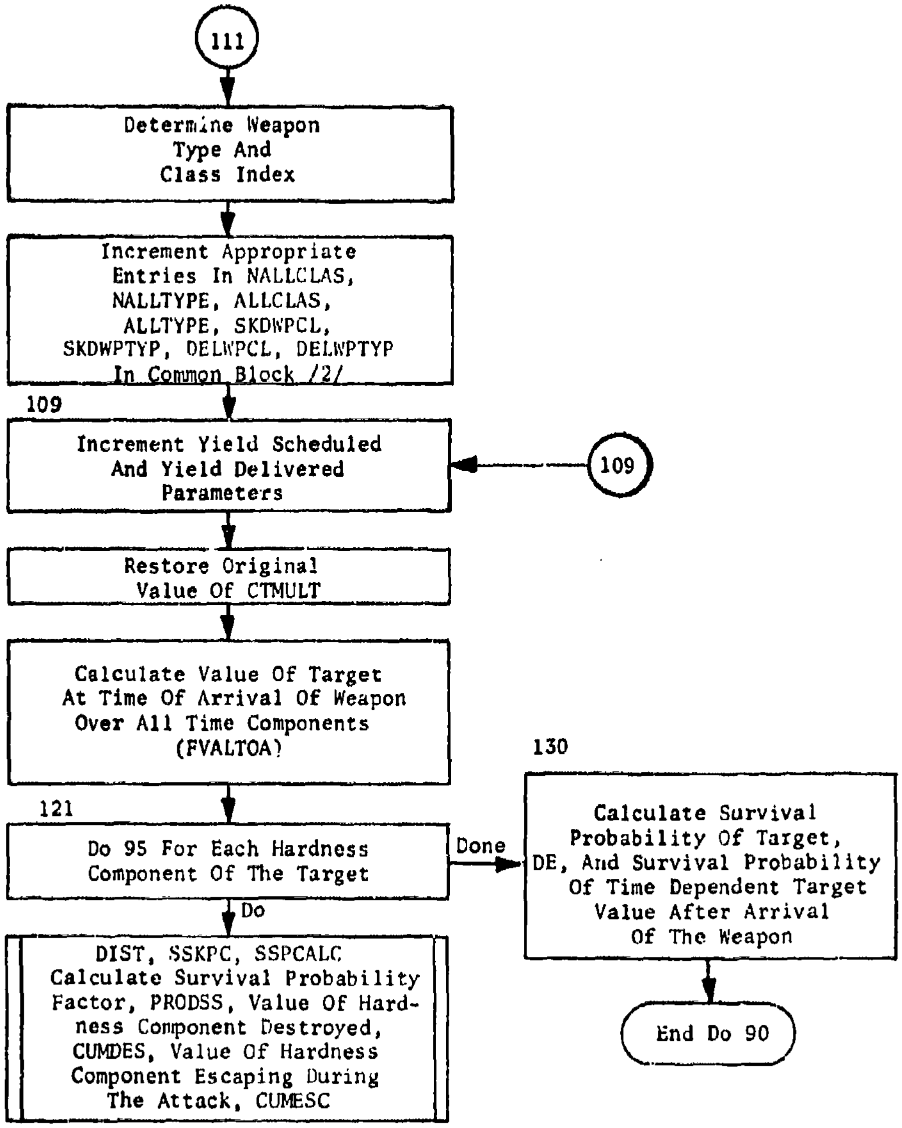


Fig. 210. (cont.)
(Sheet 2 of 2)

SUBROUTINE EVAL2

PURPOSE: To read the weapon allocation data, evaluate and classify it, summarize it, and print it in the summaries produced by EVALALOC.

ENTRY POINTS: EVAL2

FORMAL PARAMETERS: None

COMMON BLOCKS: 2, FILABEL, FILES, ITP, LATLON, LITTLE, MIS, MYIDENT, NOPRINT, OPT, PLOT, TGTMOD, TWORD, WPNMOD

SUBROUTINES CALLED: EVALPLAN, ITLE, ORDER, PAGESKP, RDARRAY, RDWORD, REORDER, SETREAD, SETWRITE, SKIP, TERMTAPE, TGTMODIF, WPNMODIF, WRARRAY, WRWORD

CALLED BY: EVALALOC

Method

EVAL2 first reads weapon parameter arrays from the BASFILE. Then it initializes control numbers and the arrays which will contain data for the summary prints to zero. If this is not the first pass (NPASS $\neq$ 1), EVAL2 calls subroutine WPNMODIF to modify the weapon parameters in the manner specified by the user. In this way the sensitivity of the weapon allocation and the evaluation performed by EVAL2 to changes in these parameters can be tested.

If the run uses the weapon allocation prepared by program ALOC, EVAL2 then reads the processes target and weapon allocation data from the ALOCTAR file, one block at a time. Otherwise, it reads these data from the SCRATCH file prepared by subroutine PACK using the ALOCTAR file and the PLANTAPE.

For each target, EVAL2 orders the weapon arrays by time of arrival; in the case of multiple or complex targets, additional data for each multiple target element or complex target component are read from the POSTDATA section of the BASFILE. Then if this is not the first pass, and the run is after PLNTPLAN (JOPT(1)=1), EVAL2 calls TGTMODIF once for each target, target element, or target component to perform user-specified modifications on the target parameters; again this is done to test sensitivity of the evaluation to these parameters. It also calls EVALPLAN for each target, target component to classify the weapons allocated, the expected damage done by the weapons, and the change in value of the target.

This process is repeated for each target on the ALOCTAR or SCRATCH file. Two differences exist if EVALALOC is run before ALOCOUT instead of after PLNTPLAN. First, EVAL2 calls TGTMODIF and EVALPLAN only once for each multiple target in this case. Second, if this is the first pass, it writes the target index number and the ALOCTAR weapon arrays on a scratch file which is used in subsequent passes by subroutine TGTMODIF to perform its target parameter modifications. At the user's option, during this phase of EVAL2 processing, the target and weapon data are printed for a specified number of targets in a SAMPLE TARGET LIST whose contents are described in the EVALALOC section of Chapter 3, Plan Generation Subsystem, User's Manual, Volume II.

When all target and weapon data from the ALOCTAR or SCRATCH (post-PLNTPLAN run) file have been processed, EVAL2 prepares summary tables which describe the results of the allocation. These cumulative results of the weapon allocation are stored and printed by target class and type. The TARGET DESTRUCTION SUMMARY is produced during every pass through EVAL2. (EVAL2 does a new evaluation of the weapon evaluation for each set of user weapon and target parameter modification cards; each evaluation is a new <u>pass</u> through EVAL2.) The other summaries are produced by EVAL2 only during the first pass. They are the SCHEDULE OF WEAPONS ALLOCATED, the SCHEDULE OF WEAPONS DELIVERED, SCHEDULED MEGATONNAGE, and DELIVERED MEGATONNAGE. When all passes through EVAL2 are completed, control returns to EVALALOC.

Subroutine EVAL2 is illustrated in figure 211.

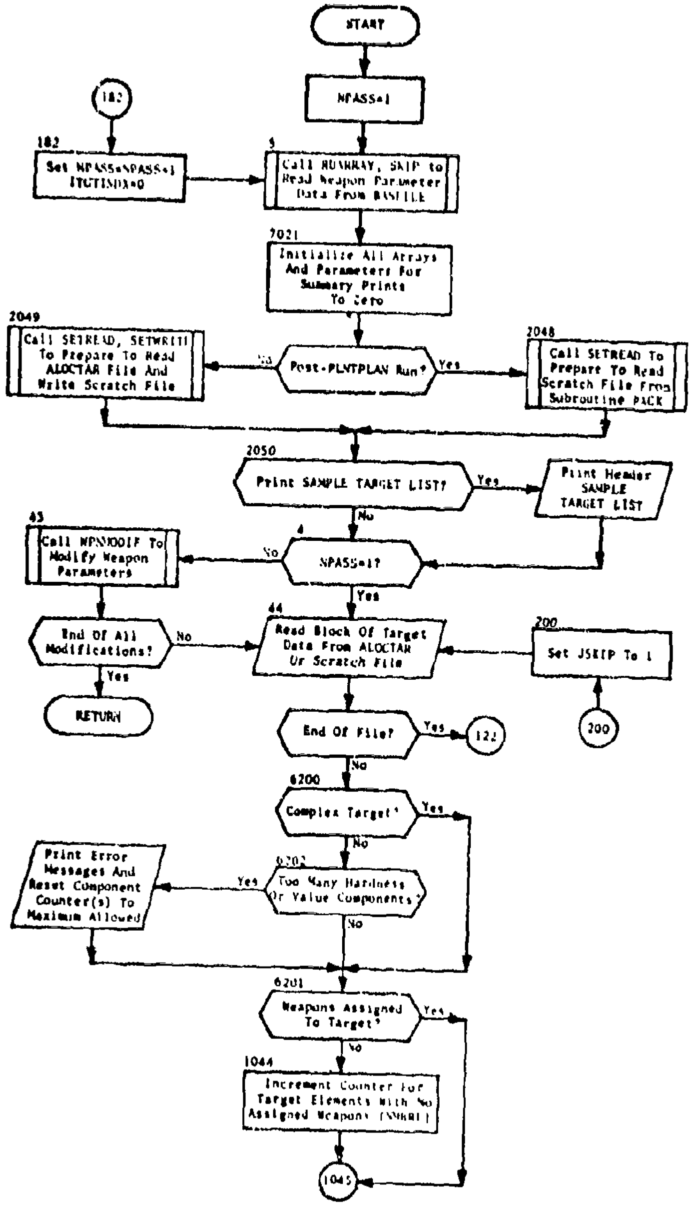


Fig. 211. Subroutine EVAL2
(Sheet 1 of 4)

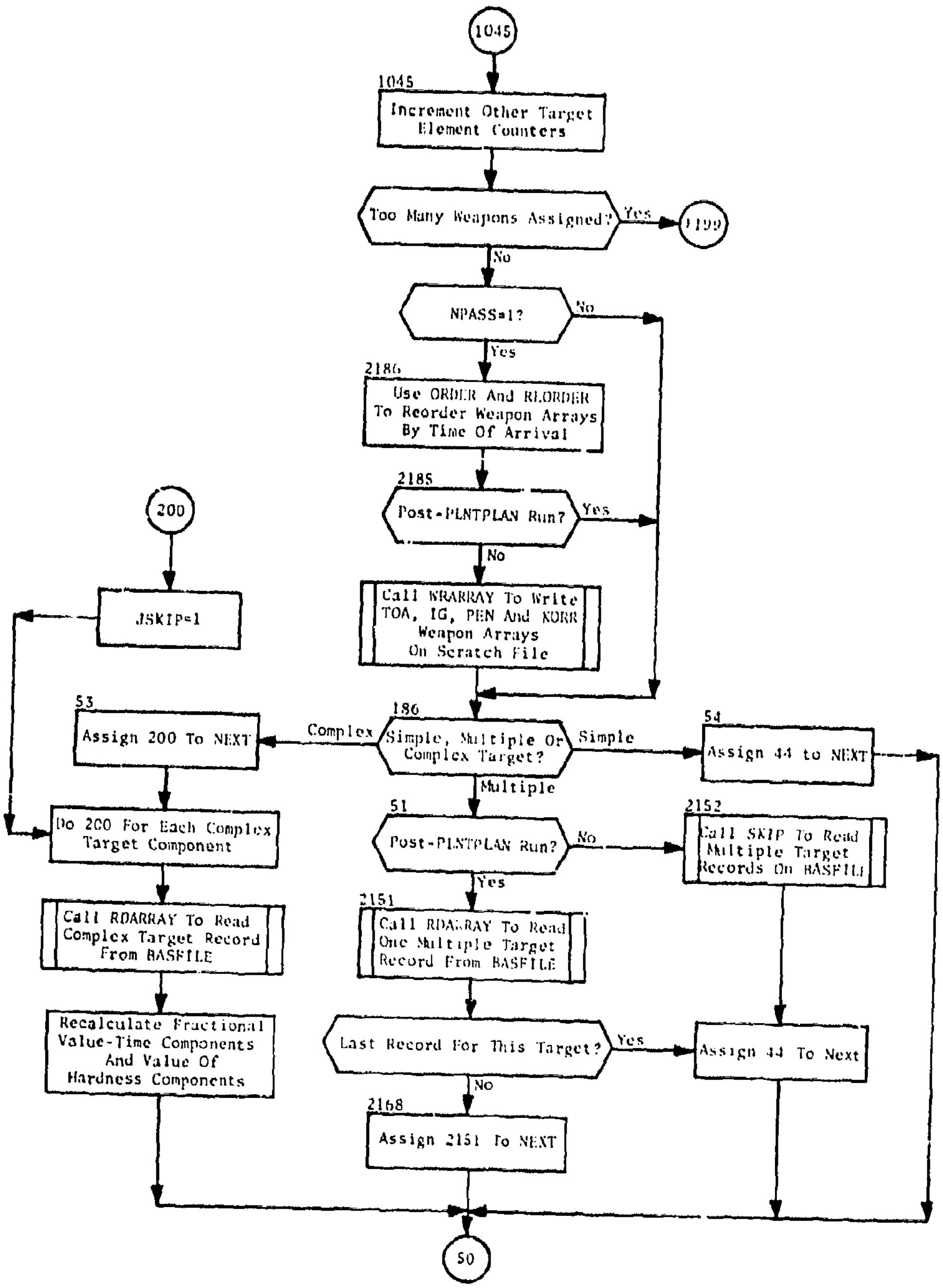


Fig. 211. (cont.)
(Sheet 2 of 4)

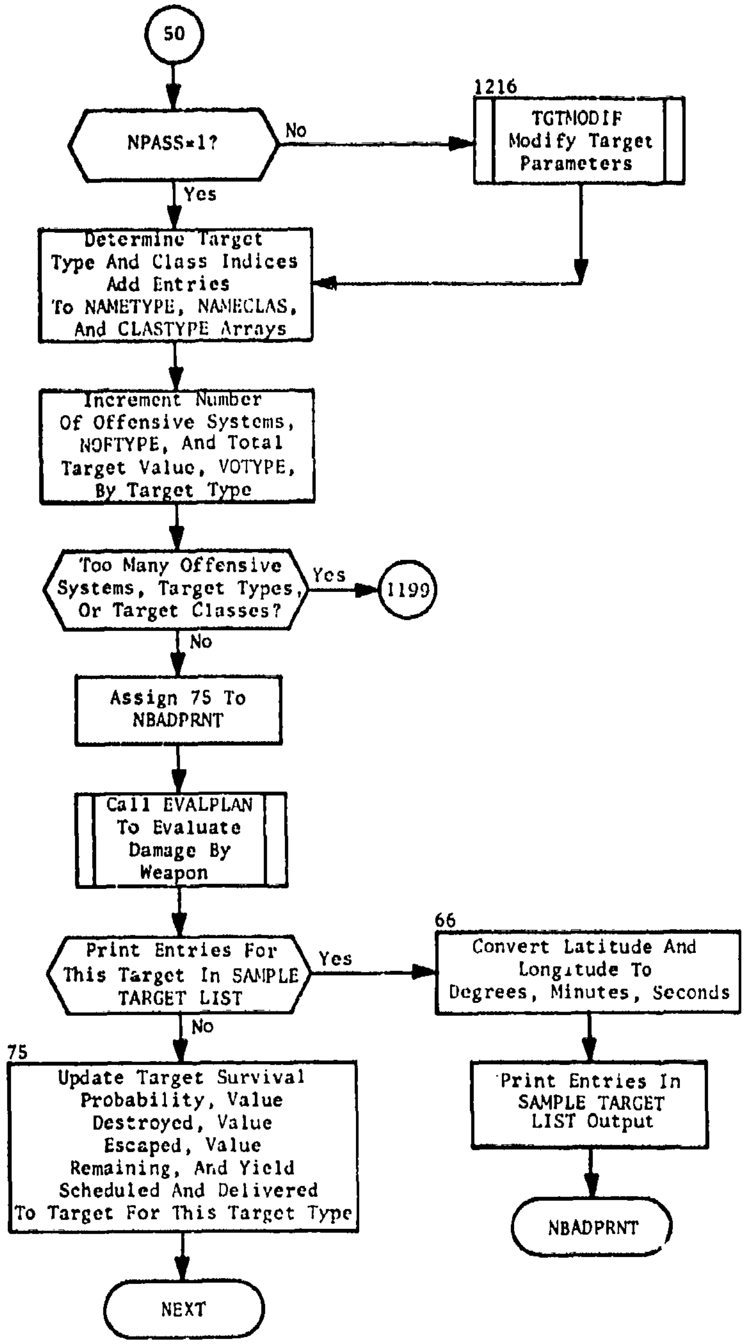


Fig. 211. (cont.)
(Sheet 3 of 4)

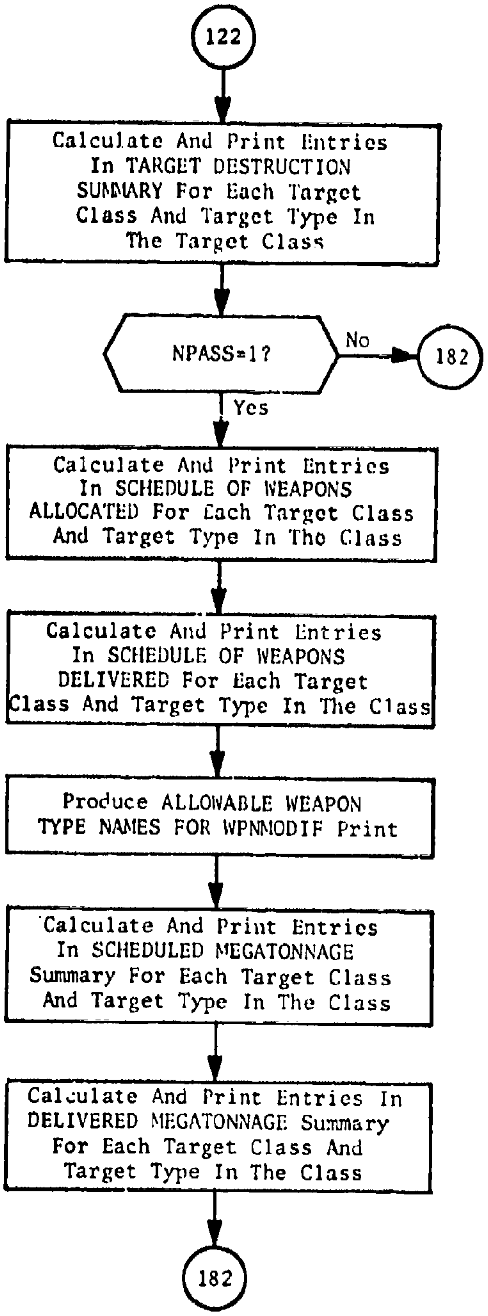


Fig. 211. (cont.)
(Sheet 4 of 4)

SUBROUTINE MISLPAKR

PURPOSE:	To pack the weapon allocation data for each missile target event into three words.
ENTRY POINTS:	MISLPAKR
FORMAL PARAMETERS:	LLL - index of entry in ITGTX array which contains the target index number
COMMON BLOCKS:	1, 2, OPT
SUBROUTINES CALLED:	SEARCH
CALLED BY:	PACK

Method

The method here is exactly the same as that used in BOMRPAKR except that the time of arrival parameter ITIME and the weapon delivery latitude and longitude LAT and LON come from different variables in common block /1/, and the target index number is ITGTX(J).

Subroutine MISLPAKR is illustrated in figure 212.

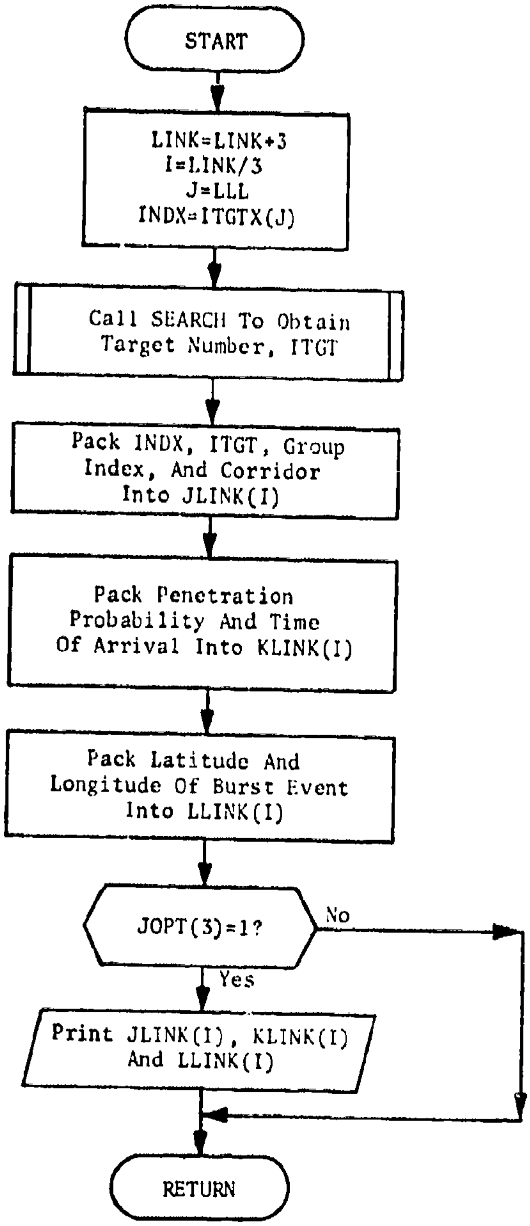


Fig. 212. Subroutine MISLPAKR

SUBROUTINE PACK

PURPOSE:	PACK with its associated subroutines BOMRPAKR, MISLPAKR, SEARCH, and UNPACKER, converts the sortie-order weapon allocation on the PLANTAPE to target-ordered allocation and writes this new allocation onto a SCRATCH file which is treated like the ALOCTAR file in subsequent processing by EVAL2. PACK is used only in post-PLNTPLAN operation of EVALALOC.
ENTRY POINTS:	PACK
FORMAL PARAMETERS:	None
COMMON BLOCKS:	1, 2, BINSCH, FILABEL, FILES, ITP, LATLON, MYLABEL, NOPRINT, OPT, PLANREC, TGTMOD, TWORD
SUBROUTINES CALLED:	BOMRPAKR, MISLPAKER, ORDER, RDWORD, RDARRAY, REORDER, SETREAD, SETWRITE, TERMTAPE, UNPACKER
CALLED BY:	EVALALOC

Method

Before PACK can reorder the PLANTAPE weapon allocation into target order, it must obtain the target order from the ALOCTAR file. It does this by reading the target index number and target number one target at a time from the ALOCTAR file and by packing these into the next unfilled word of the JORDER array until the whole file has been read. Then it uses ORDER and REORDER to reorder the array in target index number, IND, order (since the PLANTAPE does not contain target number data). Now PACK is ready to process the PLANTAPE which it does one record at a time. Tanker records on the PLANTAPE are ignored by PACK since they contain no weapon allocation data.

PACK packs data for each event on the PLANTAPE which is a target event. When processing a bomber record, it calls subroutine BOMRPAKR once for each event involving a target burst to pack the weapon allocation data into the next unused words in the JLINK, KLINK, and LLINK arrays and to determine the target number on the ALOCTAR file which corresponds to this target index number. Subroutine MISLPAKR is called by PACK to perform the analagous function for missile target events. This process is repeated for every missile and bomber record on the PLANTAPE until the JLINK, KLINK, and LLINK arrays are filled. Then PACK reorders these arrays in target number order. If the arrays are filled for the first time, they are simply written onto a SCRATCH file in the new order. If not, they are merged with the packed data which must be read from the previously written SCRATCH file; the merged data, which is in target number order, is written onto a new SCRATCH file. Then PACK continues reading and processing PLANTAPE records as before until the packed arrays are again filled or until the PLANTAPE has been completely read and all the weapon allocation data are contained in target number order on one scratch file. (If the total number of warheads is no greater than 5,000, no SCRATCH file is needed or written.)

The final phase of processing by PACK consists of reading the ALOCTAR file one target block at a time, calling UNPACKER to unpack the associated weapon allocation data for the target from the previously written SCRATCH file, and writing the ALOCTAR target data and PLANTAPE weapon allocation data together on a new SCRATCH file. When all targets on the ALOCTAR file and weapon data from the PLANTAPE have been processed in this way, PACK writes the ALOCTAR end-of-file record on the ALOCTAR-like SCRATCH file and returns control to EVALALOC.

Subroutine PACK is illustrated in figure 213.

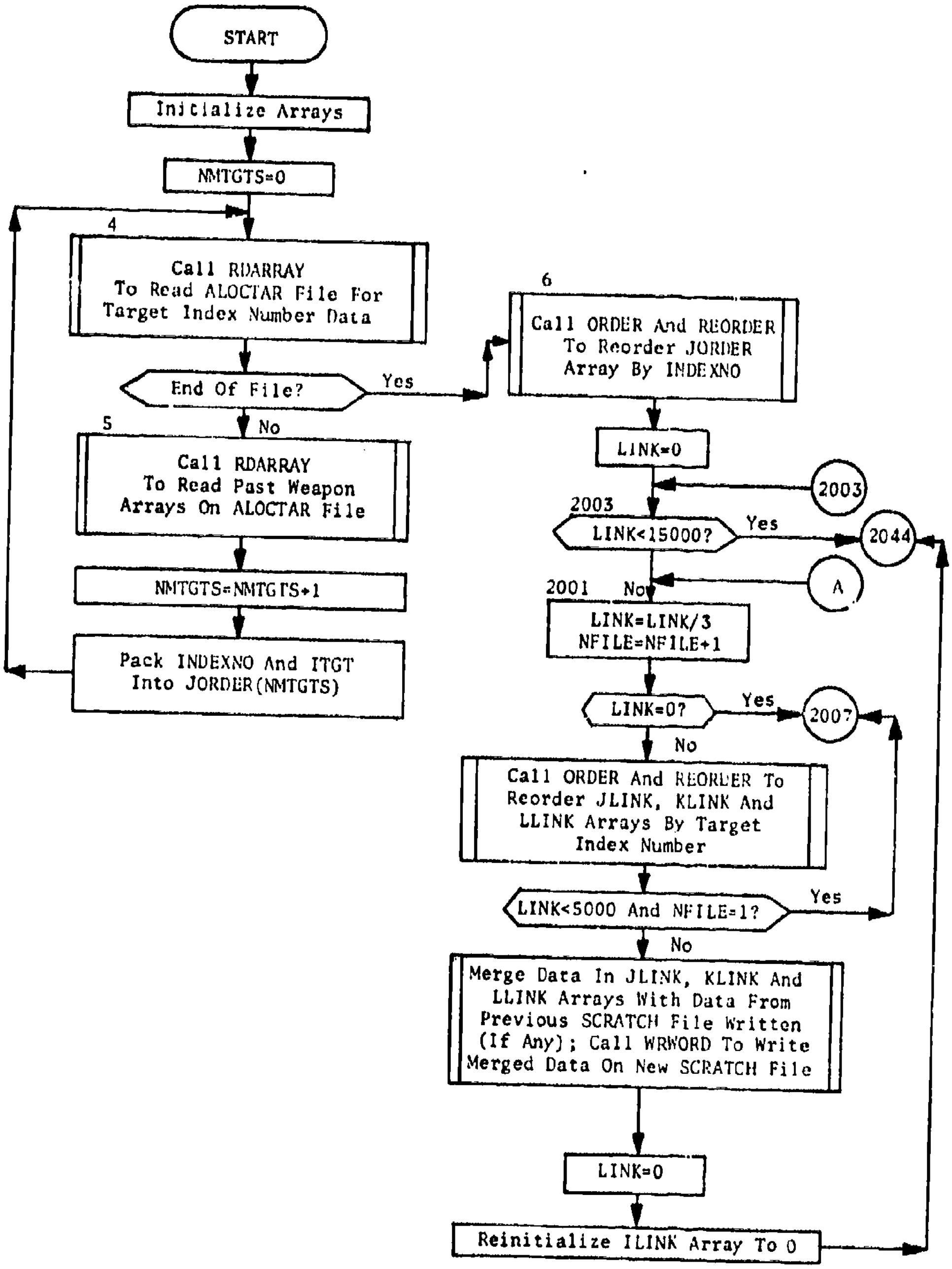


Fig. 213. Subroutine PACK
(Sheet 1 of 3)

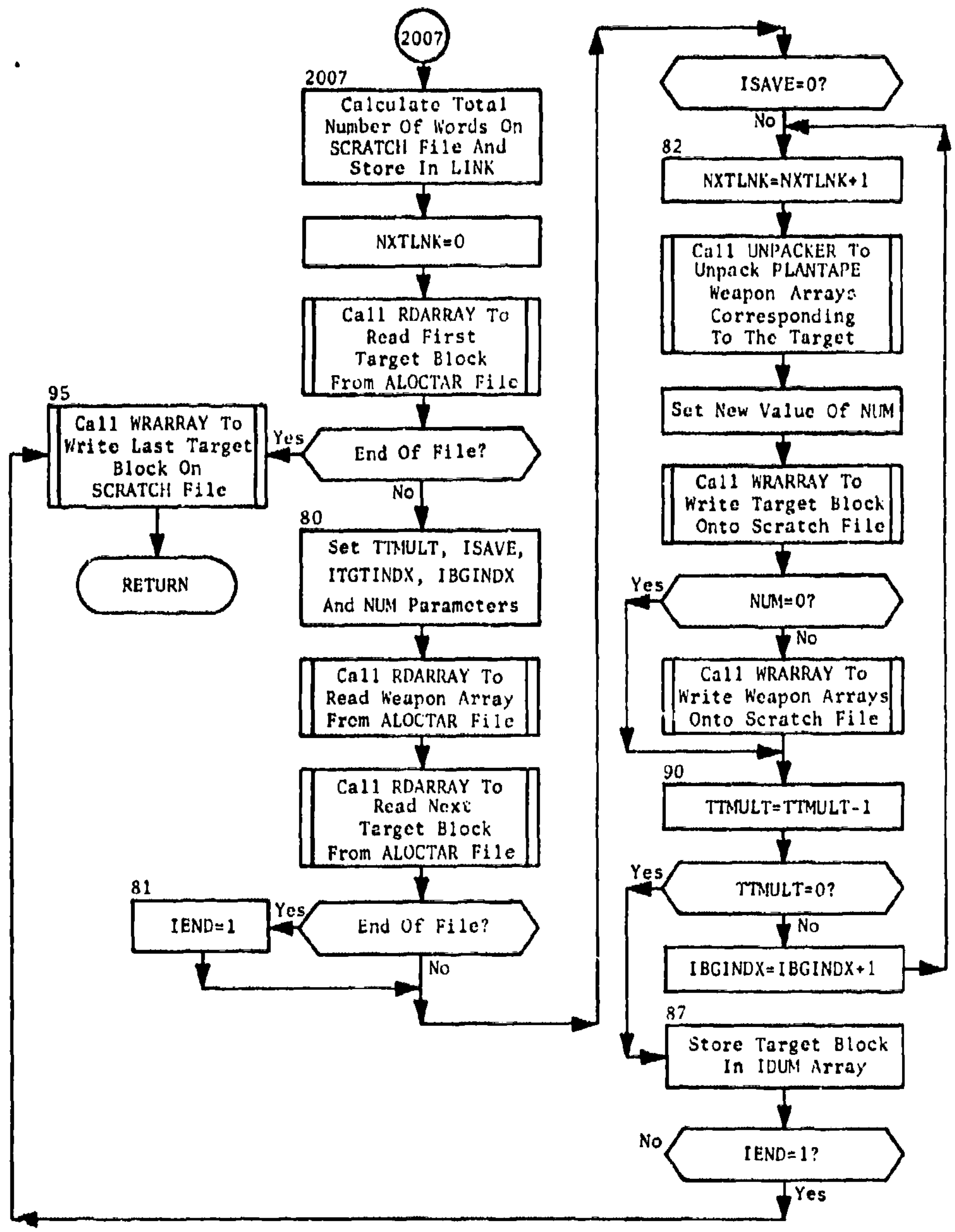


Fig. 213. (cont.)
(Sheet 2 of 3)

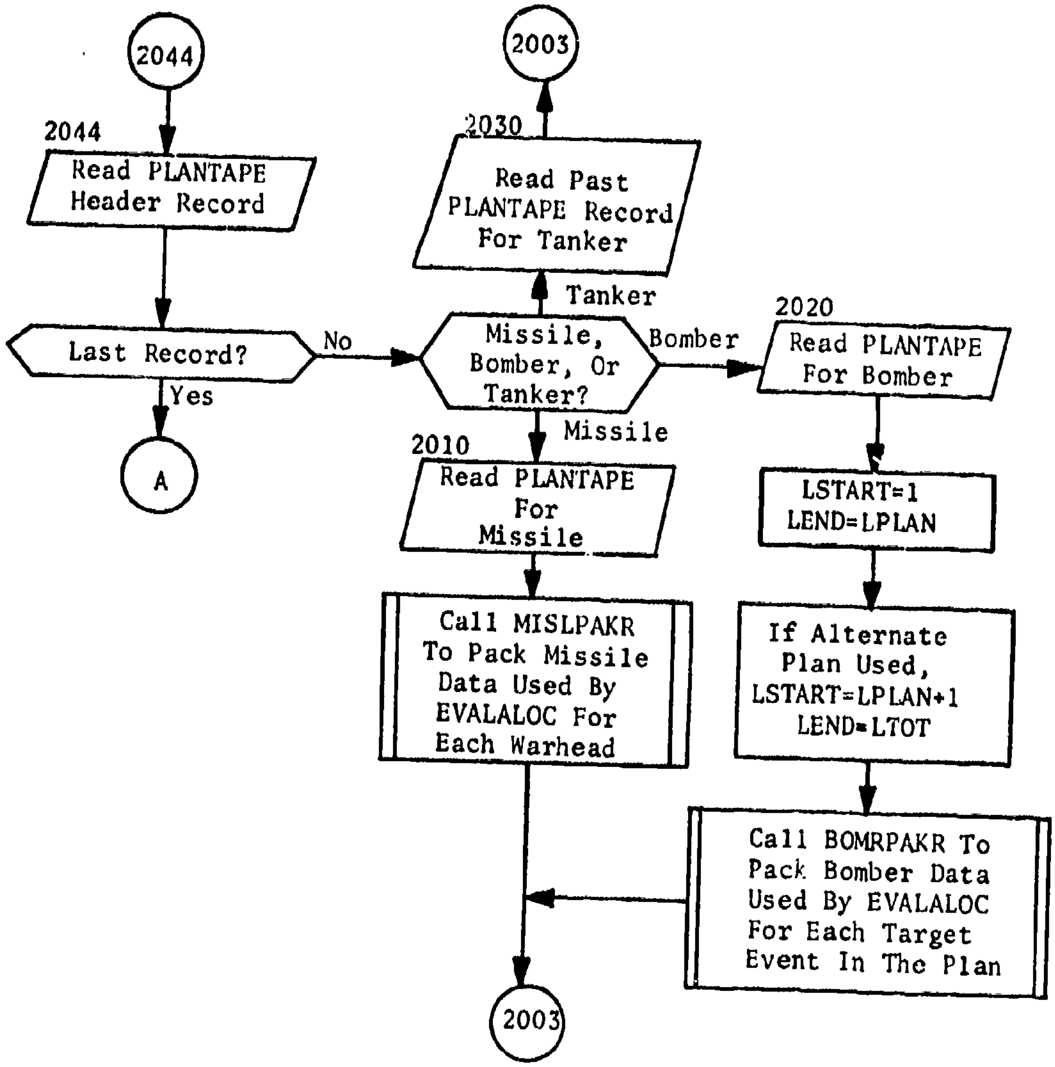


Fig. 213. (cont.)
(Sheet 3 of 3)

SUBROUTINE SEARCH

PURPOSE:	To determine the target number from the ALOCTAR file which corresponds to the index number of a PLANTAPE target.
ENTRY POINTS:	SEARCH
FORMAL PARAMETERS:	INDEXNO - The target index number for which the target number is sought ITGT - The target number corresponding to INDEXNO and determined by SEARCH
COMMON BLOCKS:	BINSCH
SUBROUTINES CALLED:	None
CALLED BY:	MISLPAKR, BOMRPAKR

Method

Subroutine PACK sets up the JORDER array, which is a list of packed words. Each word in the array contains a target index number (INDEX) and a target number (ITGT) corresponding to a target on the ALOCTAR file, and the array is in increasing order by INDEX. SEARCH uses a binary search to find the word in the JORDER array which has target index number INDEXNO or which is the index number closest to INDEXNO but not larger than INDEXNO. This word has index NNOW; SEARCH unpacks JORDER(NNOW) to obtain ITGT.

The first time SEARCH is called it determines the parameters NLOG, INT, and NOW used in the binary search. These parameters are contained in the common block /BINSCH/.

Subroutine SEARCH is illustrated in figure 214.

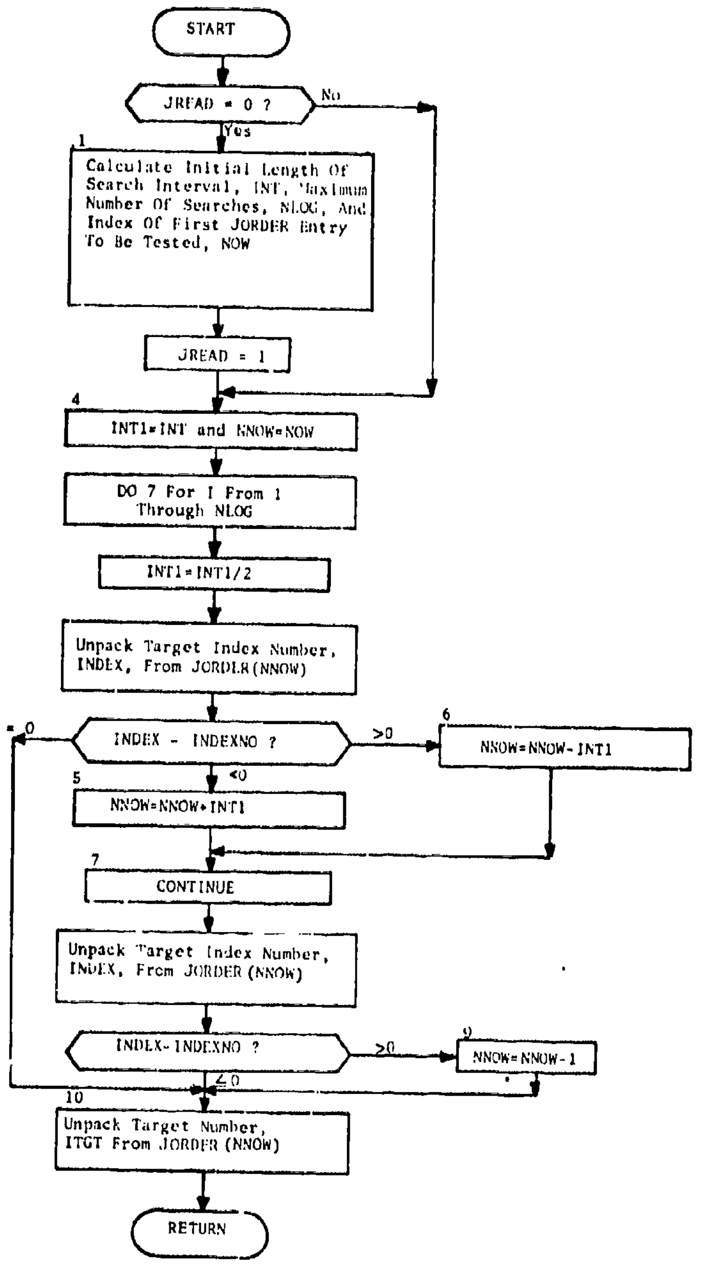


Fig. 214. Subroutine SEARCH

SUBROUTINE SSSPCALC

PURPOSE: To calculate target survival probabilities for multiple weapon attacks. This routine will consider either the exponential or square root damage law.

ENTRY POINTS: INITPROB, SSSPCALC

FORMAL PARAMETERS:
SSS - A single shot survival probability
NWP - A number of weapons
J - Index to hardness component

COMMON BLOCKS: LAW, LITTLE

SUBROUTINES CALLED: None

CALLED BY: EVALPLAN

Method

The INITPROB entry point is used to initialize two local arrays which are used in the calculations. This entry is called once for each target, before processing the weapon damage calculations in EVALPLAN. The formal parameters have no effect on this entry point. The two local arrays are indexed by hardness component. They are defined as follows:

CUMKILL(K)	Current fraction of Kth hardness component surviving. Initialized to 1.0.
SUMSK(K)	Current sum of kill factors for Kth hardness component. Initialized to 0.0.

Entry SSSPCALC computes the multiple weapon survival probability from the single shot survival probability. If the exponential damage option has been selected, then the multiple weapon survival probability is equal to the product of all the single shot survival probabilities for each weapon.

If the square root damage law option has been selected, the routine checks to see if the target radius is greater than zero. If not, the exponential damage function is used. If so, the routine must calculate the square root kill factor corresponding to the input single shot survival probability. The algorithm used for this is the same one that is used in subroutine SETABLE in program PREALOC2 and program ALOC. The algorithm is a recursive, one-dimensional search procedure to find the appropriate kill factor. The new kill factors are added to the old sum of kill factors to determine a new sum. This new sum defines the new fraction of the target that survives. The multiple weapon survival probability is then the ratio of the new fraction surviving to the old fraction surviving.

Subroutine SSSPCALC is illustrated in figure 215.

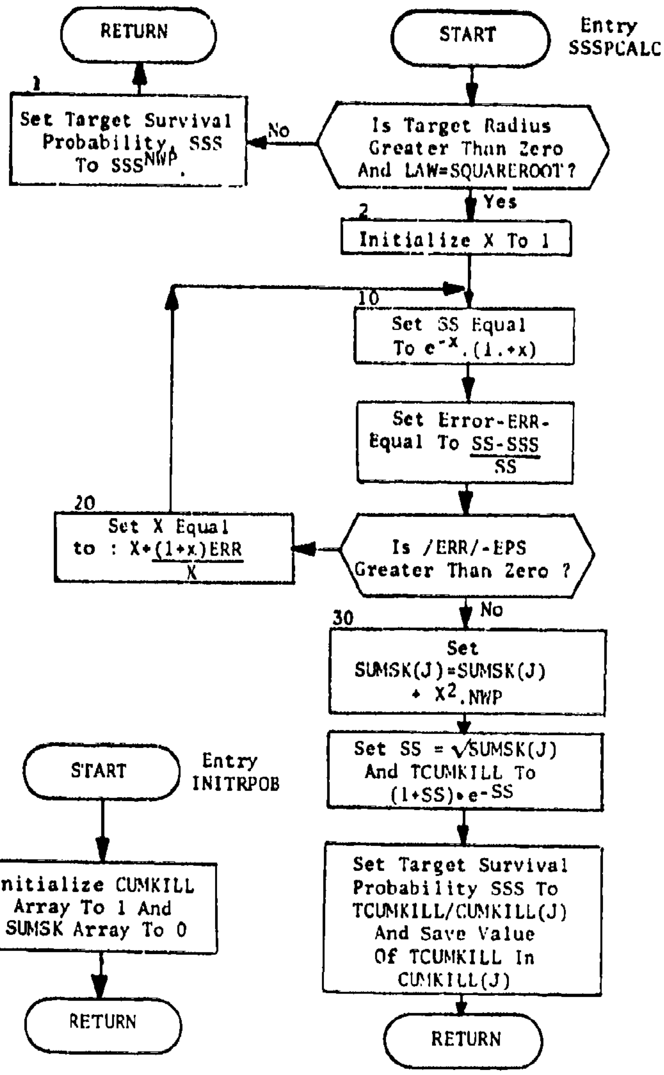


Fig. 215. Subroutine SSSPCALC

SUBROUTINE TGTMODIF

<u>PURPOSE:</u> To enable the user to modify five target parameters:

- H(1) - The hardness of the target component in PSI
- VO(1) - The value of the target at hardness H(1)
- TAU(1) - The time at which the target changes value rapidly
- FVAL(1) - The fraction of the target value associated with TAU(1)
- PEN - Penetration probability of a weapon allocated to the target

<u>ENTRY POINTS:</u> TGTMODIF

<u>COMMON BLOCKS:</u> FILABEL, FILES, ITP, MYIDENT, NOPRINT, OPT, TGTMOD, TWORD, WPNMOD

<u>SUBROUTINES CALLED:</u> ITLE, RDARRAY, RDWORD

<u>CALLED BY:</u> EVAL2

Method

Modifications of target parameters are made in accordance with target parameter modification data cards supplied by the user. The target type may be ALLTGTS or a specific class or type name such as MILITARY, BEAR, etc. If the type names included in the data base are not known, they can be read from the target damage summary printed on the first pass through EVALALOC.

The attribute to be modified must be H(1), VO(1), TAU(1), FVAL(1), or PEN. If any other word is punched, the program prints an error message, (the word punched and a list of allowable words), no changes are made, and execution continues. Modifications made by the subroutine are printed out along with the number of the pass.

The penetration probability PEN is weapon-target dependent. If it is to be modified, the last four words on the data card are the weapon types to which the modification applies. The first of these four words may be ALLWPNS, BOMBERS, or MISSILES if a specific type name is not used. If one of these three general names is used, no other type names will be read from the card.

The target is described by either one or two hardness components. The total value is divided between the hardness components, VO(1) being that part of the target value associated with the first hardness component H(1). If VO(1) is changed, a check is made to ensure that the modification value does not exceed the total target value.

The target value in the QUICK system is always time dependent. The functional form of this dependence is

$$\text{Value}(T)=\text{Value}(o)*\sum_{j=1}^{NK}\frac{FVAL(j)}{[1+T/TAU(j)]^4}$$

where T is time in hours, and NK is between one and three. $\sum_{j=1}^{NK} FVAL(j)$ always equals unity.

Such a function gives a step-like dependence of the target value on time. Targets such as cities usually have only one time component (NK=1), and TAU(1) is very large. Other targets such as alert bomber bases may have a small value of TAU(1). Usually, the major part of the target value is associated with FVAL(1).

In all cases TGTMODIF modifies the specified target parameter for the specified target type(s) by multiplying it by the factor XTGTATT which is also specified by the user. If there is more than one hardness or time value component, then both VO(1) and VO(2) or all three of FVAL(1), FVAL(2), and FVAL(3) will be modified.

Subroutine TGTMODIF is illustrated in figure 216.

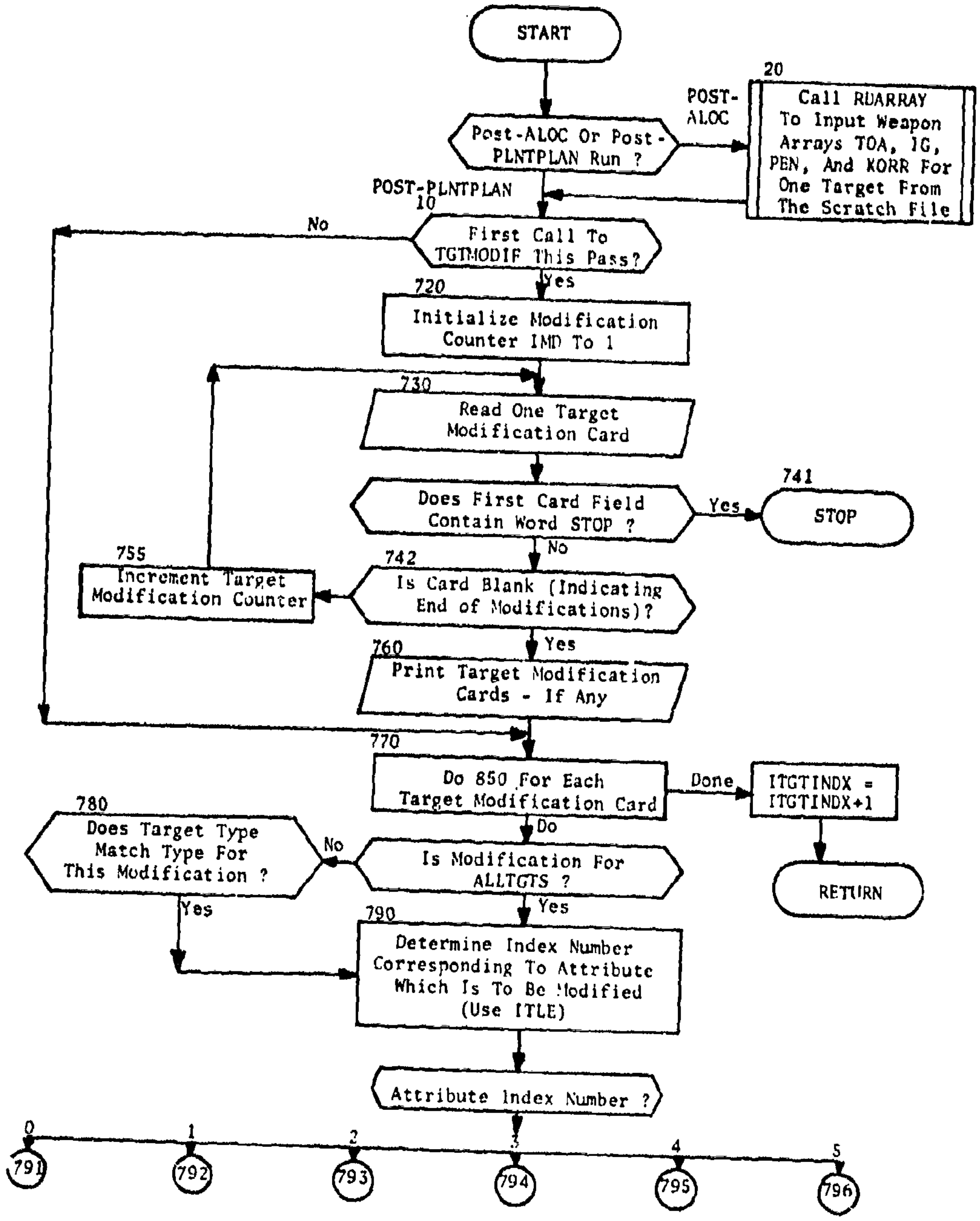


Fig. 216. Subroutine TGTMODIF
(Sheet 1 of 3)

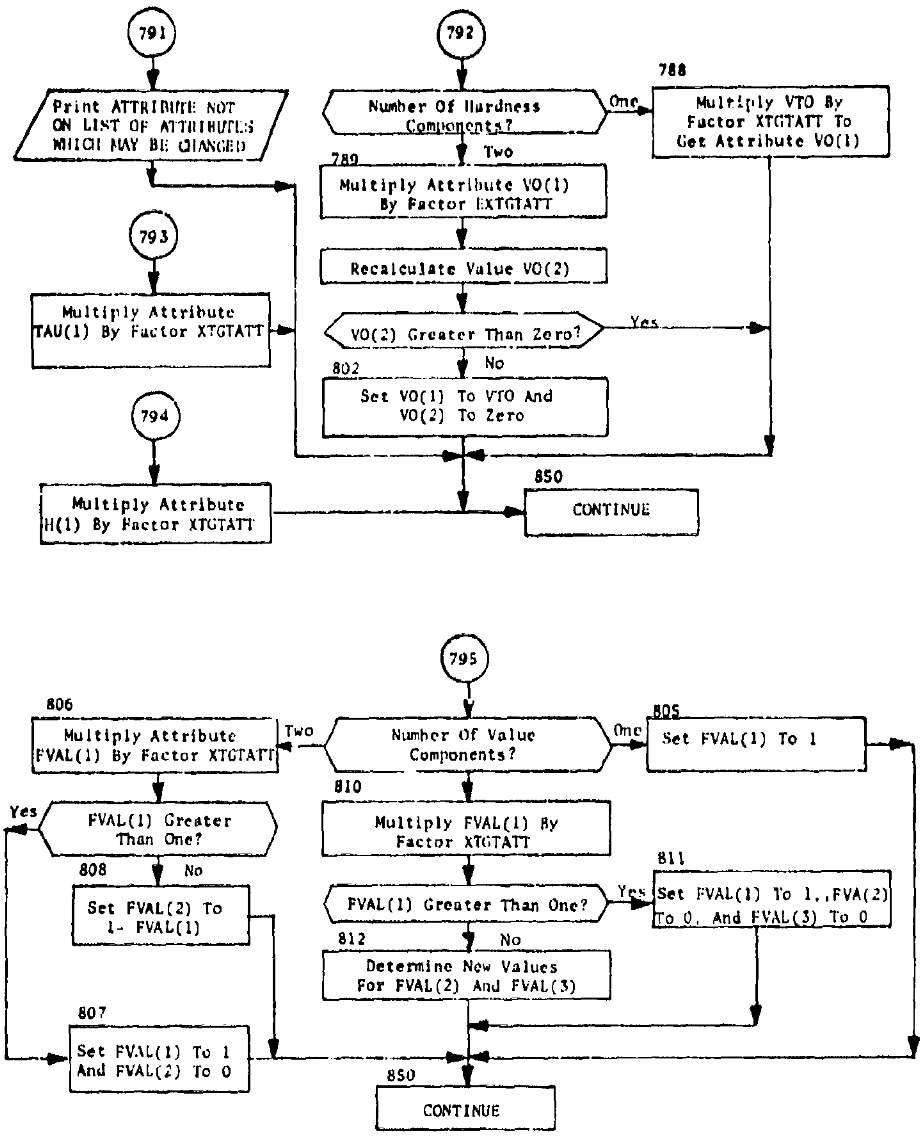


Fig. 216. (cont.)
(Sheet 2 of 3)

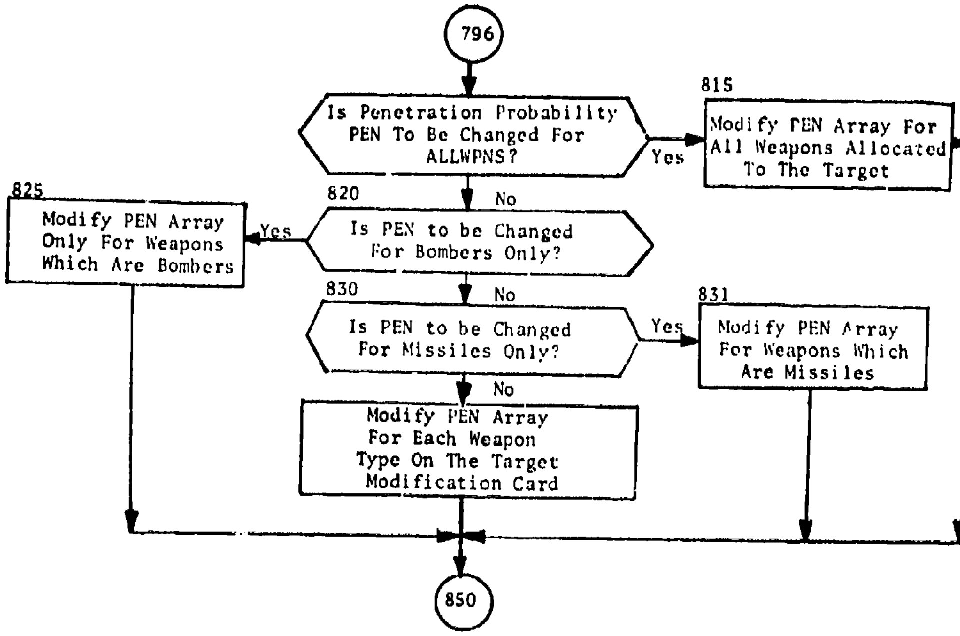


Fig. 216. (cont.)
(Sheet 3 of 3)

SUBROUTINE UNPACKER

PURPOSE:	UNPACKER reads and unpacks the total PLANTAPE weapon allocation for a target from the JLINK, KLINK, and LLINK arrays.
ENTRY POINTS:	UNPACKER
FORMAL PARAMETERS:	LNK - The first index in the JLINK, KLINK and LLINK arrays used by UNPACKER. IUNIT - The number of the scratch file containing the packed arrays
COMMON BLOCKS:	2, FILABEL, ITP, LATLON, MYIDENT, MYLABEL, NOPRINT, OPT, TGTMOD, TWORD, WPNMOD
SUBROUTINES CALLED:	ABORT, RDWORD
CALLED BY:	PACK

Method

Each time UNPACKER must determine whether to unpack weapon data from the JLINK, KLINK, and LLINK arrays, it tests the new value of LNK to see if it exceeds MAXLNK, the current maximum index for weapon array words in these packed arrays. If LNK does exceed MAXLNK, UNPACKER refills JLINK, KLINK, and LLINK from the scratch file IUNIT, keeps track of how many words are left on IUNIT, and resets LNK to unity. If LNK does not exceed MAXLNK, or after the arrays have been filed with packed data, UNPACKER unpacks the target index number INDEXNO from JLINK (LNK) and determines whether it is in the range from ITGTINDX to IENDTGTX. If it is, UNPACKER unpacks the rest of the weapon data from KLINK(LNK) and LLINK(LNK), and increments the weapon counter NWPN for this target. LNK is then increased by one and the above process is repeated.

If the number of weapons allocated to a target exceeds 30, UNPACKER attempts to reduce the number by combining the weapons, where possible. If it cannot reduce the number of weapons, it prints the message UNPACKER UNABLE TO REDUCE NUMBER OF WEAPON GROUPS ON TARGET and aborts the run (via ABORT).

If the scratch file and ALOCTAR file are not in synchronization (i.e., the scratch file has a target number or index number greater than the current one on the ALOCTAR file), a message indicating this fact is printed, and the run is aborted (via ABORT).

The first time INDEXNO is not in the correct range, UNPACKER set NUM (the number of weapons assigned to the target) to the current value of NWPN, and LNK to LNK-1. Control is then returned to PACK where the target and weapon data are output on the ALOCTAR-like scratch file.

Subroutine UNPACKER is illustrated in figure 217.

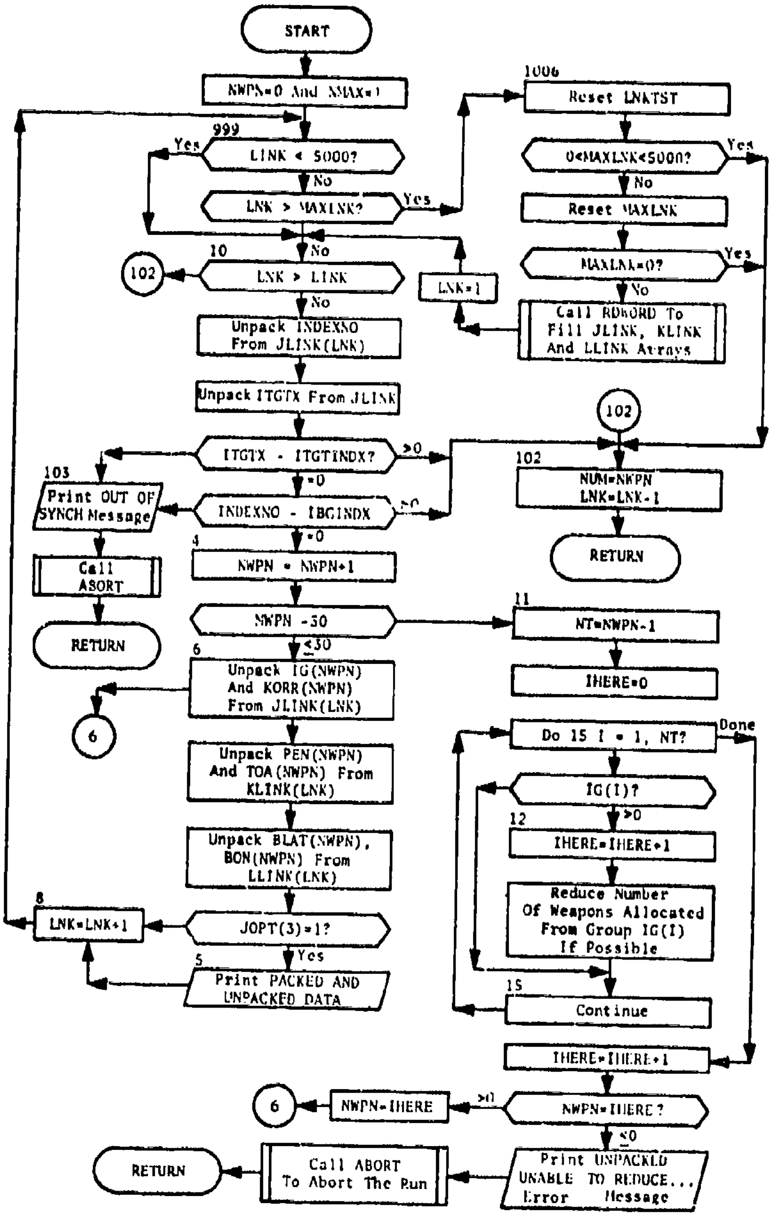


Fig. 217. Subroutine UNPACKER

SUBROUTINE WPNMODIF

PURPOSE:	To allow the user to modify reliability REL and circular error probability CEP by weapon class or type and DBL probability and weapon yield by weapon group
ENTRY POINTS:	WPNMODIF
COMMON BLOCKS:	OPT, WPNMOD
SUBROUTINES CALLED:	None
CALLED BY:	EVAL2

Method

WPNMODIF allows the user to modify specified weapon parameters, namely, reliability (REL), circular error probability (CEP), DBL probability, and weapon YIELD for chosen weapon types. The type name may be ALLWPNS, BOMBERS, MISSILES, a specific type name such as B-58-E, or GROUP. The parameters DBL and YIELD can only be changed by the type name GROUP. WPNMODIF reads the users weapon parameter modifications one card at a time, and multiplies the parameter by the factor on the card for the specified weapon type, class, or group. Internal checks are made by WPNMODIF to ensure that REL never exceeds unity. As each modification is made it is printed out.

Exit from WPNMODIF is accomplished when it finds a blank card or a RETURN card at the end of the modifications.

Subroutine WPNMODIF is illustrated in figure 218.

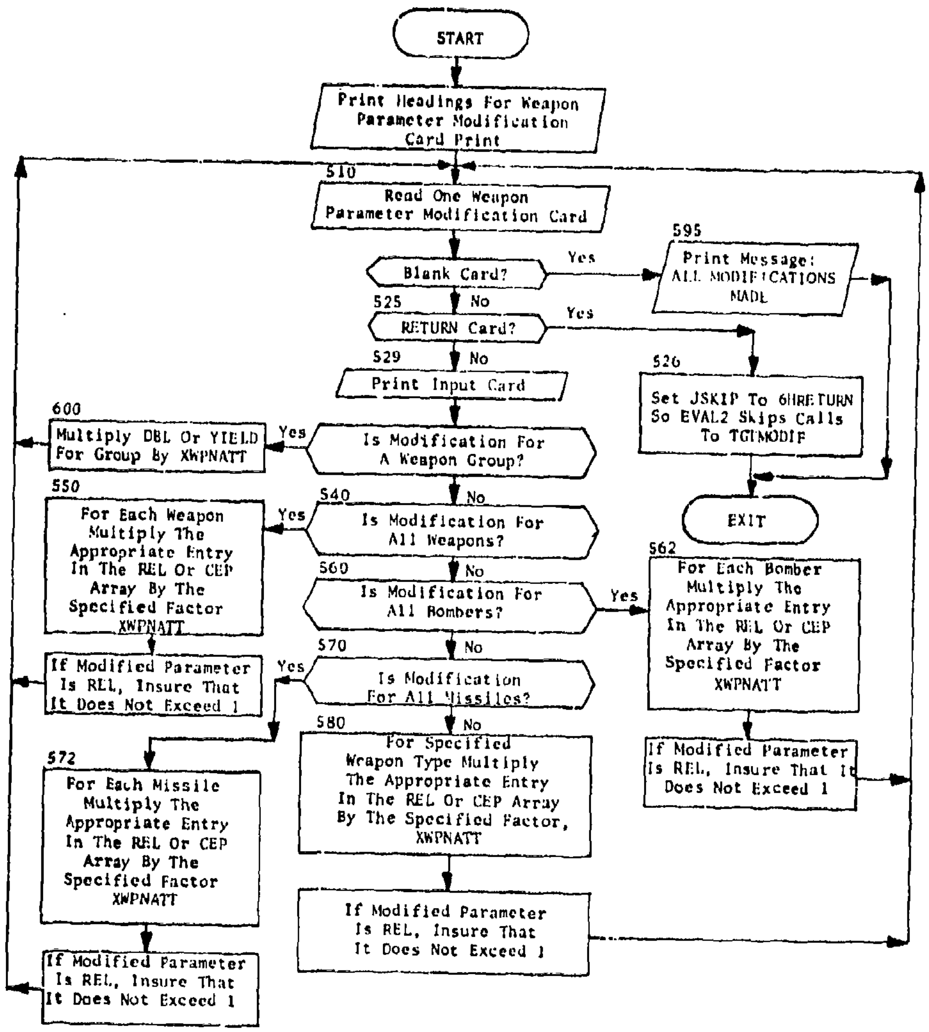


Fig. 218. Subroutine WPNMODIF

CHAPTER 10
PROGRAM INTRFACE

PURPOSE

Program INTRFACE is one of the two special-purpose processors which provide an interface between QUICK and other computerized simulation systems used in strategic war gaming. These programs, INTRFACE and program TABLE, extract and reformat data from QUICK-developed files and output data which are used as input to the Event Sequenced Program (ESP), the Nuclear Exchange Model (NEMO), and to a NMCSSC damage assessment system such as SIDAC (Single Integrated Damage Analysis Capability System). INTRFACE (and TABLE) do not produce output files which are used within QUICK. Specifically, INTRFACE processes data contained on the PLANTAPE and reformats it for output on three tapes, the STRIKE tape (PTAPE), the STRKREST tape (STAPE) and the sortie specifications tape (ABTAPE).

INPUT FILES

The QUICK developed files used by INTRFACE are the BASFILE prepared by program PREPALOC and the PLANTAPE output by program PLNTPLAN. The BASFILE is used to obtain weapon and vehicle type data which are not included on the PLANTAPE. The PLANTAPE provides detailed information about missile, bomber, and tanker plans prepared by the QUICK Plan Generator.

OUTPUT FILES

INTRFACE produces three output tapes, all containing 80-column BCD card images.

1. The STRIKE tape (PTAPE) containing strike card ("S" card) type data for each missile and bomber weapon scheduled for delivery (for bombers, only the weapons associated with the primary plan are considered). The strike card format is shown in table 56.

2. The STRKREST tape (STAPE) containing strike card data which are essentially the same as that on the STRIKE tape except that it is formatted differently. The STRKREST card format is shown in table 57.

3. The sortie specifications tape (ABTAPE). This tape contains a set of BCD card images for each missile, bomber (primary mission), and tanker plan contained on the PLANTAPE. A card set consists of one "A" card which contains general descriptive information and a variable number of "B" cards which define the individual flight legs of the mission. The "A" and "B" card formats are described in tables 58 and 59, respectively.

Table 56. Format of Strike Card on STRIKE Tape (Sheet 1 of 2)

CARD COLUMN	VARIABLE NAME	INFORMATION	CONTENT
1		Strike card indicator	S
2		Zero	0
3	ICMD	Command or function code	1-9
4-8	INDEXT	QUICK target index number	00001-99999
9-10	DAHOMIMO	Day of target detonation	01-31
11-12	↓	Hour of target detonation	00-23
13-14	↓	Minutes of target detonation	00-59
15-16	↓	Month of target detonation	01-12
17-18	YEAR	Year of target detonation	00-99
19-20	LATTGT	Degrees (Latitude of target)	
21-22	↓	Minutes (Latitude of target)	
23-24	↓	Seconds (Latitude of target)	
25	↓	North or South (Latitude of target)	N or S
26-28	LONGTT	Degrees (Longitude of target)	
29-30	↓	Minutes (Longitude of target)	
31-32	↓	Seconds (Longitude of target)	
33	↓	East or West (Longitude of target)	E or W
34-38	JDESIG	Target designator	2 Alpha, 3 Numeric
39-40	IPLS	PLS - Probability of pre-launch survival	00-99
41-42	IPTP	PTP - Penetration probability	00-99
43-44	IWSR	WSR - Weapon system reliability	00-99
45	IREG	Region code	01-99
46		Blank	
47-49	IFRAC	Fission/yield ratio	000-999
50-54	IYIELD	Weapon yield (KT)	00001-99999
55	KHOB	HOB (Height of burst)	A or G (air or ground)
56-57		Blank	

Table 56. (cont.)
(Sheet 2 of 2)

CARD COLUMN	VARIABLE NAME	INFORMATION	CONTENT
58-60	KCEP	CEP (in 100s of feet)	000-999
61		Blank	
62-63	ITASK	Target task code	2 Alpha
64-65	IXNTRY	Code for country of target location	2 Alpha
66		Blank	
67-68	IPABORT	Percent chance of target attrition	00-99
69		Blank	
70-71	IPLNETYP	Plane type code	00-99
72-73	IWPNTYPE	Weapon type code	00-99
74-77	IUNIT	Unit number	0000-9999
78-79	ISORTIE	Sortie number	00-99
80		Blank	

Table 57. Format of Strike Card on STRKREST Tape
(Sheet 1 of 2)

CARD COLUMN	VARIABLE NAME	INFORMATION		CONTENT
1		Strike card indicator		S
2		Zero		0
3	ICMD	Command or function code		1-9
4-8	INDEXT	QUICK target index number		00001-99999
9-10	DAHOMIMO	Day	of target detonation	01-31
11-12	↓	Hour		00-23
13-14	↓	Minutes		00-59
15-16	↓	Month		01-12
17-18	YEAR	Year		00-99
19-20	LATTGT	Degrees	Latitude of target	
21-22	↓	Minutes		
23-24	↓	Seconds		
25	↓	North or South		N or S
26-28	LONGTT	Degrees	Longitude of target	
29-30	↓	Minutes		
31-32	↓	Seconds		
33	↓	East or West		E or W
34-38	JDESIG	Target designator		2 Alpha, 3 Numeric
39-40	IPLS	PLS - Probability of pre-launch survival		00-99
41-42	IPTP	PTP - Penetration probability		00-99
43-44	IWSR	WSR - Weapon reliability		00-99
45	IREG	Region code		1-9
46-48	IFRAC*	Fission/yield ratio		000-999
49		Blank		
50-54	IYIELD	Weapon yield		00001-99999

* When yield (cc 0-54) is less than 100, the field for fission/yield ratio (cc 46-48) is blank.

Table 57. (cont.)
(Sheet 2 of 2)

CARD COLUMN	VARIABLE NAME	INFORMATION	CONTENT
55	KHOB	HOB (Height of burst)	A or G (air or ground)
56-57		Blank	
58-60	KCEP	CEP (in 100s of feet)	000-999
61		Blank	
62-63	ITASK	Target task code	2 Alpha
64-65	ICNTRY	Code for country of target location	2 Alpha
66		Blank	
67-68	IPABORT	Percent chance of target attrition	00-99
69-71	IWPNSYS*	Weapon system	3 Alpha
72-73	IPLNETYP	Plane type code	00-99
74-77	IUNIT	Unit number	00000-99999
78-79	ISORTIE	Sortie number	00-99
80		Blank	

*Weapon system (cc 69-71) is derived from command or function code (cc 3) as follows:

Command or Function Code	Weapon System
1	ICM
2	IRM
3	IRM
4	IRM
5	S/M
6	S/M
7	LRA
8	TAC
9	Blank

Table 58. Format of "A" Card on Sortie Specifications Tape (ABTAPE)

CARD COLUMN	VARIABLE NAME	INFORMATION	CONTENT
1		Card designator	A
2-4	LINEAI	Line number	000-999
5-8	IUNIT	Unit number	0000-9999
9-10	ISORTIE	Sortie number	00-99
11-12		Blank	
13-14	IPLNETYP	Plane type code	00-99
15		Zero	0
16-17		Blank	
18		Zero	0
19-22	IREFTIME	Reference time (launch time in hours and minutes)	0000-2359
23	ITIMEREF	Time reference	1 = launch
24-30		Zero	0000000
31-80		Blank	

Table 59. Format of "B" Card on Sortie Specifications Tape (ABTAPE) (Sheet 1 of 3)

CARD COLUMN	VARIABLE NAME	INFORMATION	CONTENT
1		Card designator	B
2-4	LINEB	Line number	000-999
5-8	IUNIT	Unit number (QUICK index number)	0000-9999
9-10	ISORTIE	Sortie number	00-99
11-12	LEG	Leg number	00-99
13-14	IOP	Operation code for QUICK RISOP-71 plans	1 - Takeoff 2 - Aerial refueling 3 or 4 - Dogleg 5 - Not used 6 - ASM launch 7 - ASM on target 8 - Decoy release 9 - Decoy impact 10 - Missile or bomb on target 11 - MIRV on target 12 - Not used 13 - Recovery splash 14 - Splash

Table 59. (cont.)
(Sheet 2 of 3)

CARD COLUMN	VARIABLE NAME	INFORMATION	CONTENT
15-19	IDES	Location identifier for given operation code. The contents column shows the entry associated with the following codes.	
		IOP= 1	Base index INDEXNO
		= 2	Area number
		= 3	Zeros
		= 4	Zeros
		= 6	00001
		= 7	Target DESIG code
		= 8	00001
		= 9	000-1
		=10	Target DESIG code
		=11	Target DESIG code
		=13	Recovery base INDEXNO
20-25	LAT	Latitude at end of leg	(Degrees, minutes, seconds)
26-33	LON	Longitude at end of leg	(Degrees, minutes, seconds)
34	MODE	Mode of operation	1 - High altitude 4 - Low altitude
35		Zero	0
36-39	ICUMTIME	Time of event	Hours and minutes
40		Zero	0
41		Blank	
42	ISOUTH	Southern latitude indicator	S if southern latitude, blank if not

Table 59. (cont.)
(Sheet 3 of 3)

CARD COLUMN	VARIABLE NAME	INFORMATION	CONTENT
43-44		Blank	
45		Zero	0
46-49		Blank	
50	IECM	ECM	0 - Off 1 - On
51		Zero	0
52-53	IPAR1	Warhead type	00-99
54	IHXX	Height of burst (HOB)	0 - Ground 1 - Air
55-58	IPLNETYP	Plane type code	0000-9999
59-60	ICNTRY	Code for country of target location	2 Alpha
61	IREGB	Region code	1-9
62		Blank	
63-64	ITASK	Target task code	2 Alpha
65-68		Blank	

CONCEPT OF OPERATION

As indicated in figure 219, INTRFACE first reads and prints the user-input cards and stores the information in the appropriate common blocks -- namely, /BURST/, /INHOB/, /GAMETIME/, /IPRT/, and /IFUNC/. (These parameters are described in User's Manual, Volume II.) Then it initializes the filehandler and proceeds to input data for common blocks /MASTER/, /CUMNO/, /PAYLOAD/, /FRACYLD/, /VEHIC/, and /PROB1/ from the BASFILE. Depending on the user print option IPRT, INTRFACE then prints weapon and vehicle tables from common blocks /FRACYLD/ and /VEHIC/.

After this initial input process, INTRFACE processes the PLANTAPE records for missiles, bombers, and tankers -- one at a time. First it reads in a PLANTAPE header block and uses word 8 (missile or weapon type) to obtain IPLNETYPE, ICMD, and CEP1 for the STRIKE and STRKREST tapes and ABTAPE. If the record is for a missile, INTRFACE then reads in the target information blocks from the PLANTAPE. If the record is for a bomber, INTRFACE first reads in the plan information blocks to determine the number (KK) of scheduled weapons (i.e., the number of drop bomb and ASM target events), and then reads in KK target information blocks. If the record is for a tanker, INTRFACE reads in the plan information blocks from the PLANTAPE.

If the user has specified that he does not want a STRIKE or STRKREST tape, INTRFACE skips the process described below.

Since INTRFACE produces strike cards only for events which are target hits by ASMs, bombs, or missiles, tanker records are ignored during the processing for the STRIKE tape. INTRFACE processing for the STRIKE tape falls into six main categories. The first is conversion of many numbers into BCD code and replacement of leading blanks in these numbers by zeros. Functions KNOBLANK and NOBLANK are used to accomplish this process. The second involves determining target impact time, TGT1TIME, in hours from the beginning of the game, and using common block /GAMETIME/ and subroutine FINDTIME to convert TGT1TIME into BCD coded data. The third is determination of target latitude and longitude in degrees and conversion of these numbers to a BCD coded word which contains the latitude or longitude in degrees, minutes, and seconds--north or south, or east or west. Function LATS (and entry LONS in LATS) are used to accomplish this process. The fourth is conversion of probabilities into percent by using function IPROB. The fifth is using IGETHOB to determine height of burst for the weapon. And the last is using subroutine YLDFRAC to calculate equivalent yield and associated fission fraction if the record is for an MRV missile. When these six categories of processing have been completed for missile and bomber records, INTRFACE writes a strike card record on the STRIKE tape and, depending on the user's option, also prints the card. It also writes a strike card record in a different format on the STRKREST tape.

When all the strike cards corresponding to the PLANTAPE record have been produced, INTRFACE tests whether the user has specified that the ABTAPE is to be made. If not, INTRFACE skips the processing of "A" and "B" cards and returns to the branch where it reads in a PLANTAPE record. Otherwise, corresponding to each aircraft in the case of bomber or tanker record and to a booster in the case of a missile, INTRFACE processes information for the "A" card. Only two categories of processing are involved. The first is conversion of integers to BCD code as mentioned before. The second is calculation of bomber or tanker takeoff time or of missile launch time in hours and minutes from the beginning of the game; NTIME is used to convert the time from floating point hours to BCD hours and minutes. When this processing for the "A" card is complete, INTRFACE writes an "A" card record on the ABTAPE. If the user has specified that the ABTAPE is to be printed and that it should be printed together with the STRIKE tape, INTRFACE prints the "A" card. It then begins processing the "B" cards. It produces a "B" card corresponding to each missile launch, missile re-entry vehicle on target, and to each bomber or tanker event which is a launch, refuel, dogleg, ASM launch, ASM on target, decoy release, decoy impact, bomb on target, recovery, or splash. Processing for this card falls into four main categories. The first is again conversion of integers to BCD code using KNOBLANK and NOBLANK. The second is conversion of floating point numbers for latitude and longitude at the end of a leg into BCD codes; LATBT and entry LONBT in LATBT are used to perform this function. The third is calculation of number of hours since beginning of game to the beginning of the route leg and conversion of this number into BCD code by NTIME. The last is determination of the target region associated with the missile re-entry vehicle, ASM or bomb. At the completion of "B" card processing, INTRFACE writes the "B" card on the ABTAPE. Again, if the user specified printing of the ABTAPE together with the STRIKE tape, INTRFACE also prints the "B" card.

When all of the PLANTAPE records have been processed to produce the specified strike cards and/or "A" and "B" cards, INTRFACE writes end-of-file marks on the STRIKE and STRKREST tapes and/or ABTAPE and tests to determine if the user specified a print of the ABTAPE separate from the STRIKE tape. If a separate print was specified, INTRFACE rewinds ABTAPE, reads it one card at a time, decodes it, and prints it; at the end of the print INTRFACE calls PRNTOFFS to print the offensive systems table for the PLANTAPE, and then prints INTRFACE timing information before stopping.

COMMON BLOCK DEFINITION

The common blocks used by program INTRFACE are described in table 60.

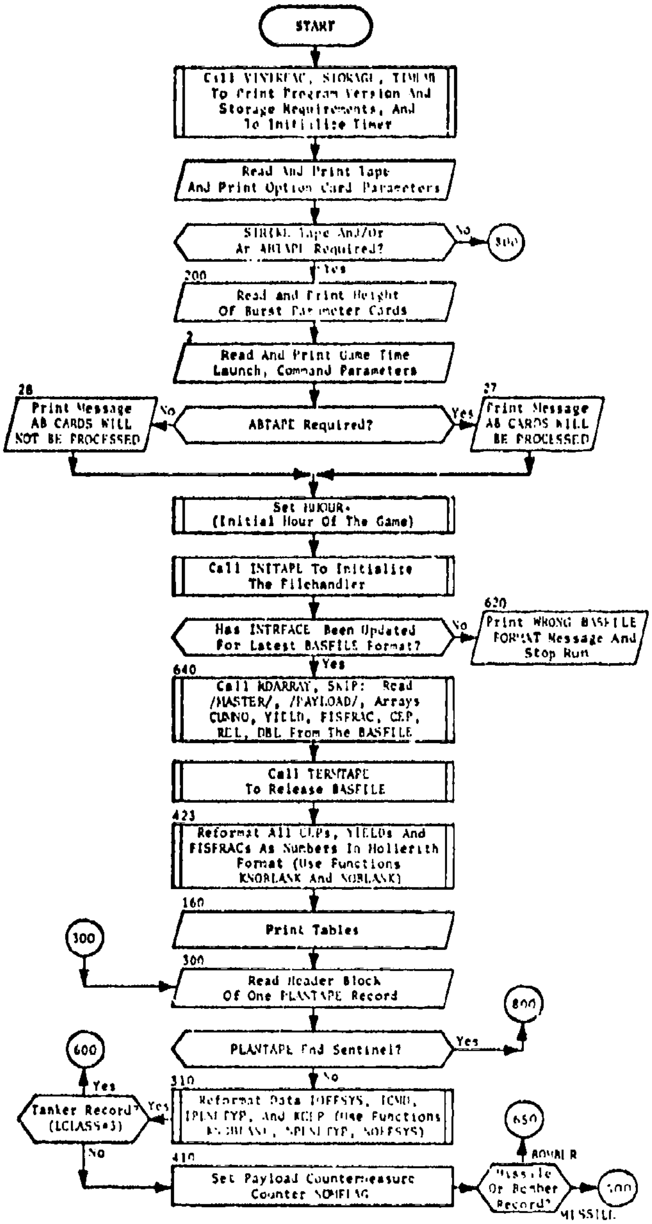


Fig. 219. Program INTRFACE
(Sheet 1 of 4)

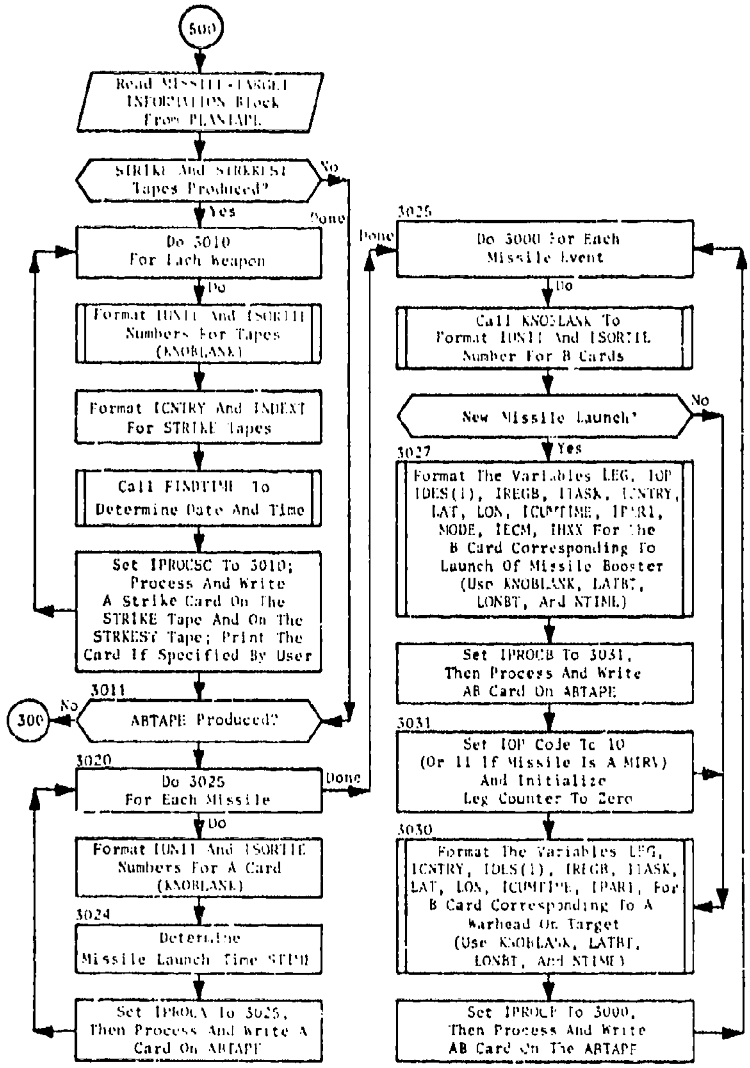


Fig. 219. (cont.)
(Sheet 2 of 4)

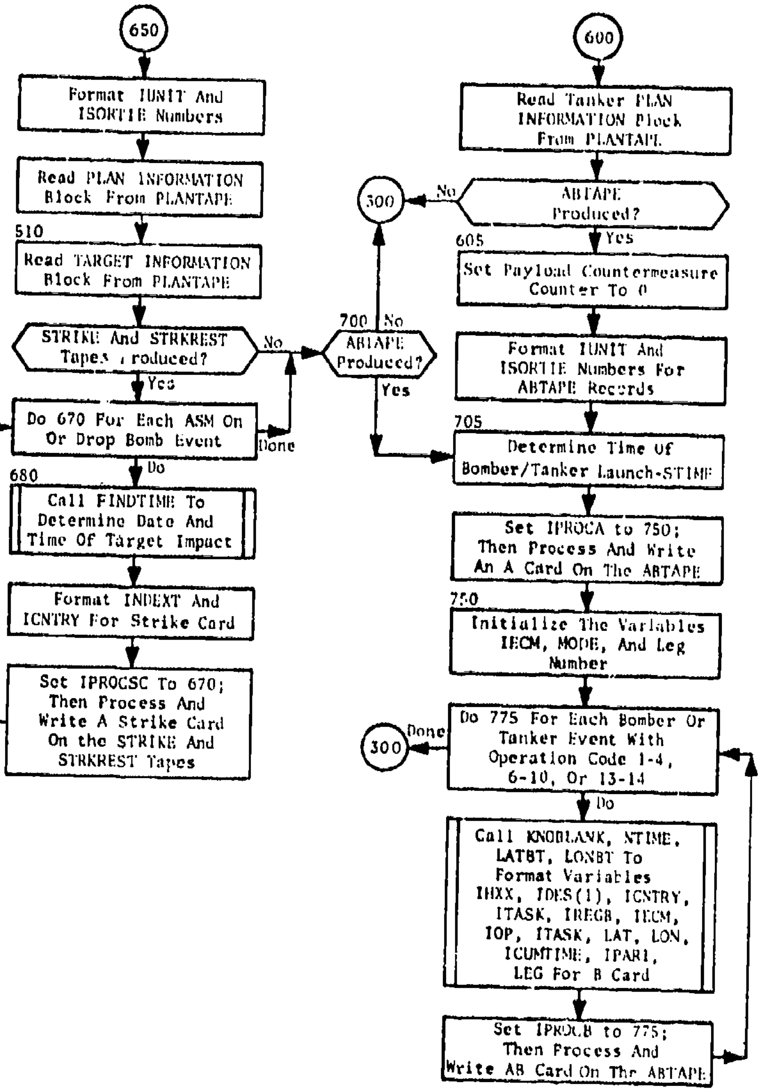


Fig. 219. (cont.)
(Sheet 3 of 4)

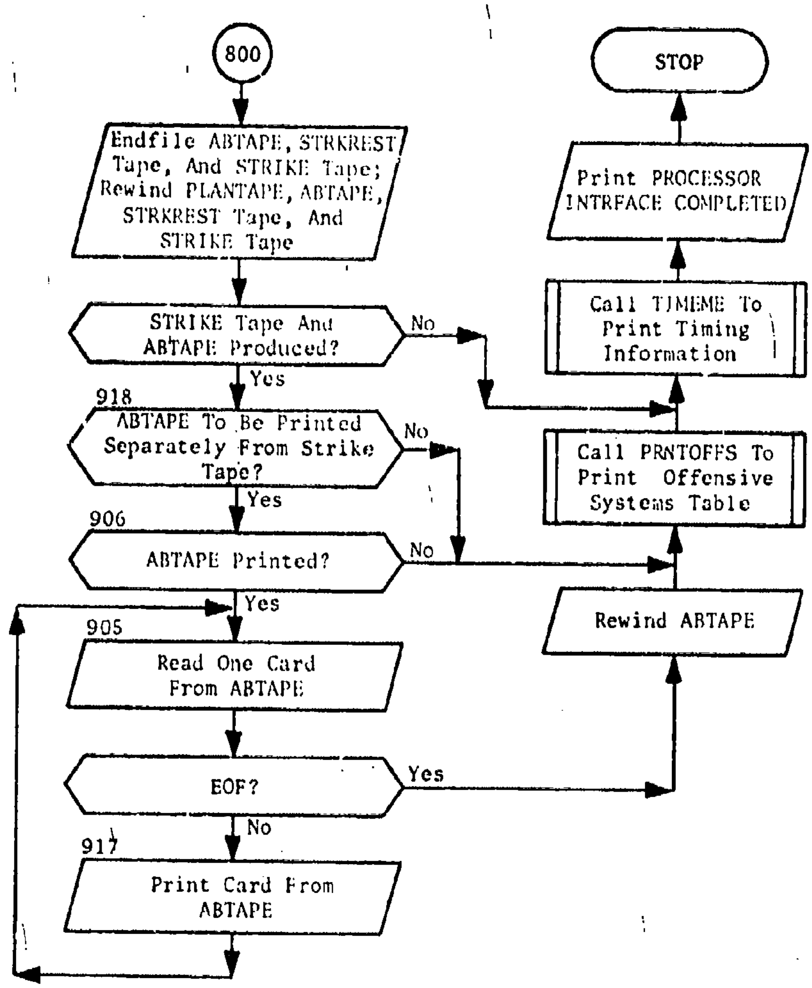


Fig. 219. (cont.)
(Sheet 4 of 4)

Table 60. Program INTRFACE Common Blocks (Sheet 1 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
BURST	LDESIG	Beginning numerical part of target designator code for target requiring specific height of burst (HOB)
	INDESIG	Ending number for specific HOB
	IWPNG	Weapon type which requires ground burst
	ITGTG	Alphabetic portion of target designator code for target requiring a ground burst
	IHOBDFLT	Default made for HOB; A = air burst, and G = ground burst
	NWPNG	The number of weapons requiring ground burst
	NTGTG	The number of targets requiring ground burst
CUMNO	ICUMNO	Cumulative number of types in each class
DATE	TGT1TIME	Time in hours, since beginning of game, of target hit
	DAHOMIMO	BCD; coded day, hour, minute, and month of target hit
	YEAR	BCD; year of target hit
DINDATA		Second part of PLANTAPE record for missiles, bombers, or tankers (see description of PLANTAPE records)
FILABEL		FILABEL is a filehandler common block
FRACYLD	YIELD	Weapon yield
	FISFRAC	Fission fraction for weapon
	NWPN	Number of weapons
GAMETIME	KDAY	Day of game

Table 60. (cont)
(Sheet 2 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
GAMETIME (cont.)	KMON	Month of game
	KYEAR	Year of game
IFUNC	IFUNC	IFUNC(I) is function or command to which QUICK plane type I belongs
	JFUNC	JFUNC(I) is function or command code if I is odd; JFUNC(I+1) is the Hollerith name for JFUNC(I)
	INDFUNC	INDFUNC(I) is the same as JFUNC(I) when I is even
INHOB	INHOB	INHOB(I) is 1 if air burst is required over region I; it is 0 if ground burst is required
IPRT	IPRT	Print option for weapon and vehicle tables; IPRT = 1 for a print; IPRT = 2 if no print of tables is desired
ISOUTH	ISOUTH	ISOUTH = 1HS if latitude is southern; otherwise ISOUTH = 1H
ITP		ITP is a filehandler common block
MASTER		MASTER is the first common block on the BASFILE. (See BASFILE description)
MISSILE	IPLTYP(40)	IPLTYP(I) is the plane type of a missile for which a launch interval is specified by the user
	DLMIS(40)	DLMIS(I) is the missile launch interval in hours corresponding to plane type IPLTYP(I)
	NDMSS	The number of type of missiles for which the user has specified a nonzero missile launch interval

Table 60. (cont.)
(Sheet 3 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
MODE	MODE	MODE is 1 for high altitude and 4 for low altitude
MYIDENT		MYIDENT is a filehandler common block
NOPRINT		NOPRINT is a filehandler common block
OFFSYS	MASKO	For offensive system type corresponding to index I, MASK(I) contains the values of NOBOMB1(I), IWHD(I), NOBOMB2(I), IWHD2(I), NASM(I), and IASM(I) from common /PAYLOAD/ and the plane type code
	IHTYP	Type name
	IPGTYP	Plan Generator type number
	ICOUNT	ICOUNT(J) is the number of offensive systems corresponding to MASKO(J)
	NT	Number of different offensive systems
PAYLOAD		PAYLOAD is the sixth common block on BASFILE; (see the description of BASFILE)
PAYLOAD2	MASK	MASK(I) is a coded sum of NOBOMB1(I), IWHD1(I), NOBOMB2(I), IWHD2(I), NASM(I), and IASM(I)
PROB1	REL	Weapon reliability
	SBLX	Pre-launch survival probability
STUB		STUB contains some of the variables found on the first part of each PLANTAPE record; (see the description of the PLANTAPE)
TAB	ITAB	The table of possible operation codes
	NOPSHOT	NOPSHOT is the dogleg number for doglegs 3 or 4; otherwise, it is 0

Table 60. (cont.)
(Sheet 4 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
TAPES	PLANTAPE	Output tape from QUICK system Plan Generator which contains records for missile, bomber, and tanker events
	BASFILE	File put out by PREPALOC in the Plan Generator; it contains common block information needed by many programs in the QUICK system
	ABTAPE	The sortie specifications tape put out by INTRFACE
	PTAPE	The STRIKE tape put out by INTRFACE
	STAPE	The STRKREST tape put out by INTRFACE
VEHIC	CEP1	Weapon CEP
	NVHC	The number of weapon vehicles

SUBROUTINE FINDTIME

PURPOSE:	To convert TGT1TIME from a floating point number of hours to a date and time in integer format where the date and time are computed from the base game time.
ENTRY POINTS:	FINDTIME
FORMAL PARAMETERS:	None
COMMON BLOCKS:	DATE, GAMETIME
SUBROUTINES CALLED:	KNOBLANK
CALLED BY:	INTRFACE

Method

TGT1TIME is the time in hours since the beginning of the game of a target hit. FINDTIME first converts TGT1TIME into integers IHOUR and MIN for minutes where MIN is a result of rounding off to the nearest integer. If MIN equals 60, FINDTIME resets it to 0 and adds 1 to IHOUR. Then if IHOUR is at least 24, FINDTIME alternately decreases IHOUR by 24 and increases IDAY by 1 until IHOUR is less than 24. (The initial values of IDAY, IMON, and IYEAR are the input values for the day, month, and year of the game.) Similarly, FINDTIME tests IDAY and if it is greater than 31 it alternately adds 1 to IMON and decreases IDAY by 31 until IDAY is less than 32. FINDTIME's final test in on IMON. If IMON is larger than 12, FINDTIME alternately decreases it by 12 and increases IYEAR by 1 until IMON is less than 13. If for some unanticipated reason IYEAR is larger than 99, FINDTIME resets it to 99 and prints an error message for ILLEGAL DATE.

Finally, FINDTIME codes IMON, IDAY, IHOUR and MIN into integer by performing the sum of IMON, 100·MIN, 10000·IHOUR, and 1000000·IDAY to obtain IDATE. KNOBLANK is used to obtain DAHOMIMO and YEAR in BCD format.

Subroutine FINDTIME is illustrated in figure 220.

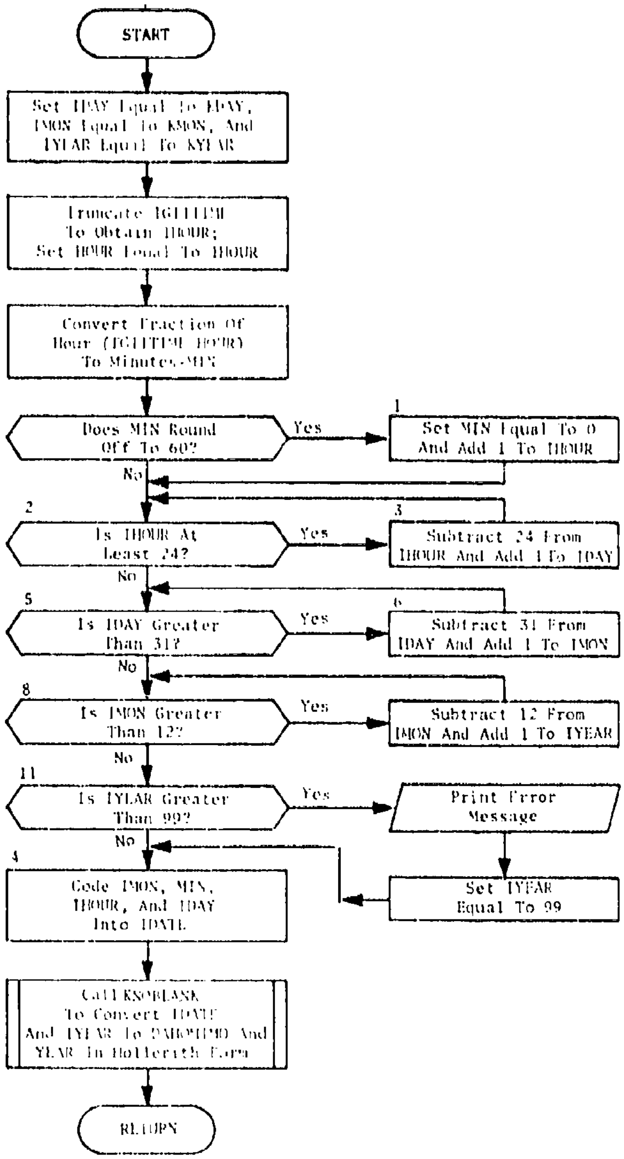


Fig. 220. Subroutine FINDTIME

FUNCTION IGETHOB

PURPOSE:	To obtain height of burst for weapon type KWPN with designator code KDSG.
ENTRY POINTS:	IGETHOB
FORMAL PARAMETERS:	KDSG - Target designator code; KDSG comes from DES array on the PLANTAPE
	KWPN - Weapon type; KWPN comes from the IWH array on the PLANTAPE
COMMON BLOCKS:	BURST, INHOB
SUBROUTINES CALLED:	None
CALLED BY:	INTRFACE

Method

First IGETHOB decodes KDSG into ITPART and determines whether the target is in one of the three target regions defined by LDESIG and INDESIG. If the target is in an undefined region, IGETHOB is set equal to the default value for height of burst -- namely IHORDFLT.

Otherwise, the target is in region I where I = 1, 2, or 3, and IGETHOB checks to see whether region I requires a definite ground or an air burst and sets IGETHOB equal to the letter G or A accordingly (G for ground, A for air). If the user has not specified the type of burst for the region, IGETHOB decodes KDSG again, this time into IAPART, and proceeds to determine whether the target is one which was input as a ground burst target in the ITGTG array. If it is one of these targets, IGETHOB is set equal to 1HG. Otherwise, the function IGETHOB checks to see whether weapon type KWPN is one which was input as a ground burst weapon in the IWPNG array. If it is, IGETHOB is set equal to 1HG. Otherwise, the default value IHOBDFLT is returned as the value of IGETHOB.

Function IGETHOB is illustrated in figure 221.

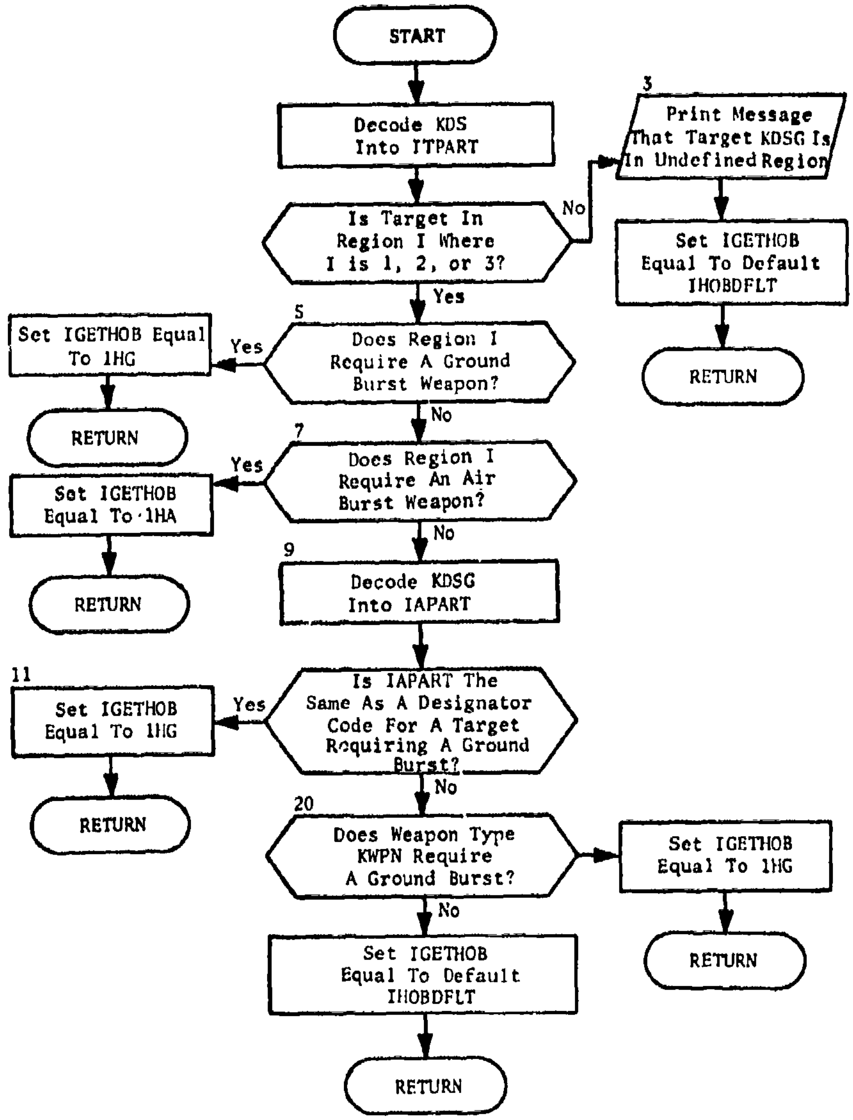


Fig. 221. Function IGETHOB

462-546 O - 72 - 40

FUNCTION IPROB

PURPOSE:	To convert probability A to a BCD integer preceded by zeros
ENTRY POINTS:	IPROB
FORMAL PARAMETERS:	A - A floating point number between 0 and 1 representing a probability
COMMON BLOCKS:	None
SUBROUTINES CALLED:	KNOBLANK
CALLED BY:	INTRFACE

Method

IPROB adds .005 to A, multiplies by 100, and truncates to convert A to integer IA. Then if IA is greater than or equal to 100, IPROB resets IA to 99. Finally IPROB is set equal to KNOBLANK(IA).

Function IPROB is illustrated in figure 222.

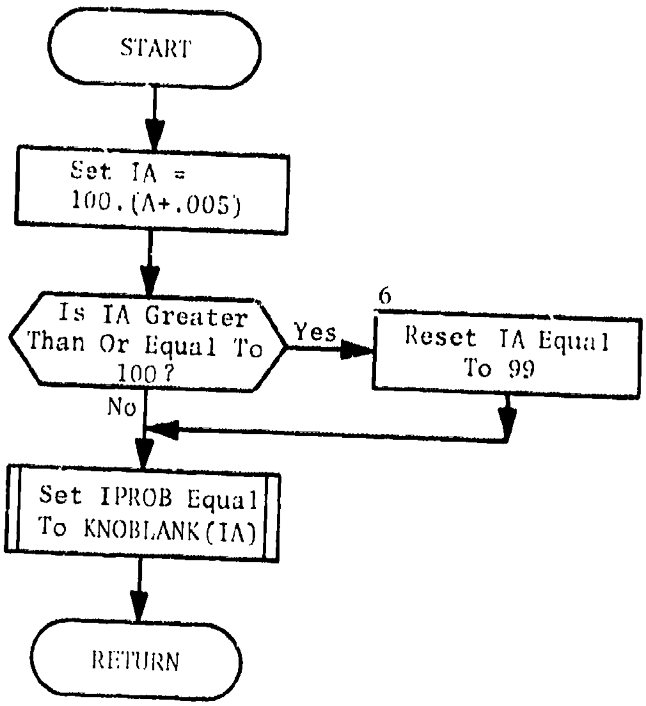


Fig. 222. Function IPROB

FUNCTION KNOBLANK

PURPOSE:	To convert an integer into BCD format with leading zeros
ENTRY POINTS:	KNOBLANK
FORMAL PARAMETERS:	IND - The integer to be converted to BCD format
COMMON BLOCKS:	None
SUBROUTINES CALLED:	NOBLANK
CALLED BY:	FINDTIME, INTRFACE, IPROB, NOFFSYS, NOP, NTIME

Method

KNOBLANK sets IT equal to IND, encodes IT into IN in I8 format, and uses NOBLANK to remove leading blanks from IN.

Function KNOBLANK is illustrated in figure 223.

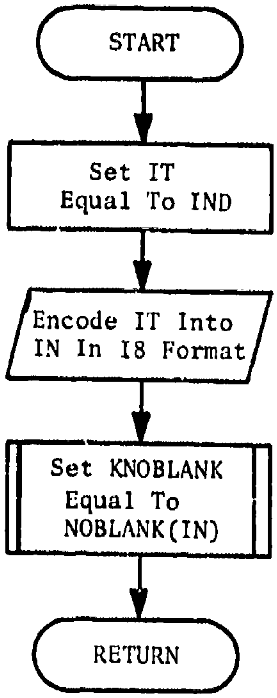


Fig. 223. Function KNOBLANK

FUNCTION LATBT

PURPOSE: To convert a floating point latitude or longitude to the form DDDMMSSH or ODDDMMSS for printout in A-format where DDD is degrees, MM is minutes, SS is seconds and H is E, W, S, or N for the appropriate hemisphere.

ENTRY POINTS: LATBT, LONBT

FORMAL PARAMETERS: A - Floating point longitude or latitude in degrees

COMMON BLOCKS: ISOUTH

SUBROUTINES CALLED: None

CALLED BY: INTRFACE

Method

First LATBT converts latitude or longitude from decimal degrees into integer degrees, minutes and seconds (LATFIX, MINFIX, and SECFIX, respectively). If entry LONBT is used, east or west longitude is set in LHEM (=1RE or 1RW) depending on whether A is greater than 180 degrees or not. Ultimately, in this case the BCD character E or W occupies the rightmost position in the word LATBT which is returned by this function.

On the other hand, if entry LATBT is used, the variable ISOUTH is set to 1H or 1HS depending on whether the latitude is north or south. In this case the north or south latitude information is never put into the word LATBT which is returned by the function.

For both entries, the parameters LATFIX, MINFIX, and SECFIX are all put into one word - LERESFIX by performing the sum of: SECFIX, 100·MINFIX, 10000·LATFIX, and 10000000. Then LERESFIX is encoded into LATBT in I8 format and if LATBT is to be returned as a longitude determination, it is shifted one BCD character to the left and LHEM is placed in the rightmost character of LATBT.

Function LATBT is illustrated in figure 224.

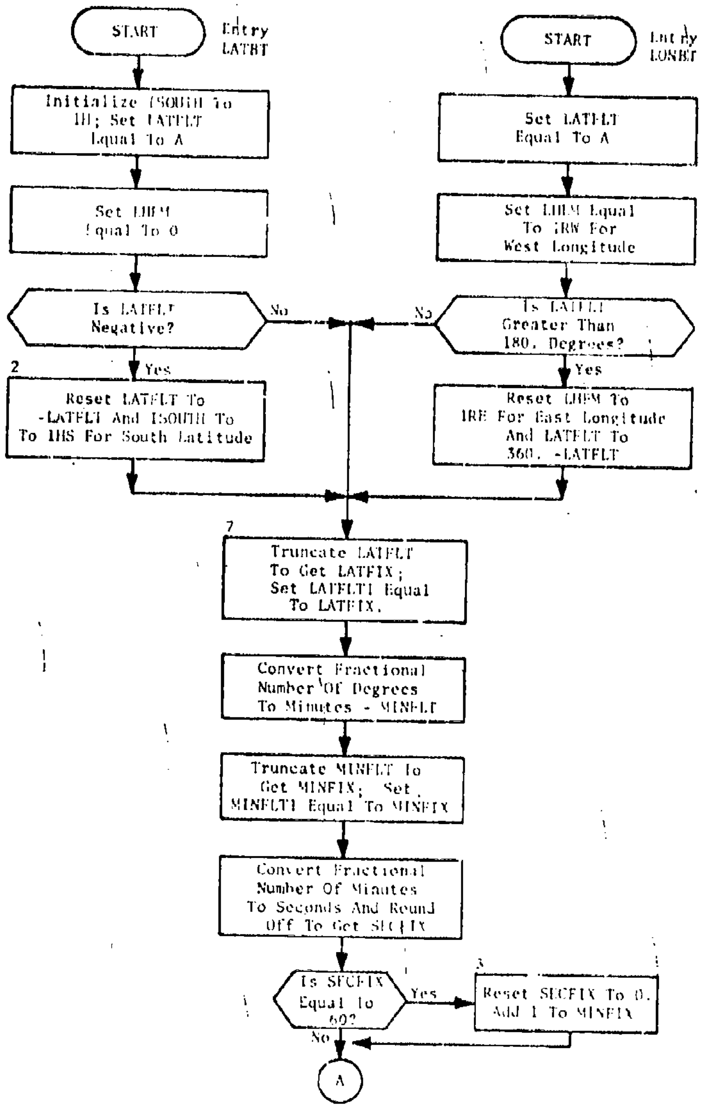


Fig. 224. Subroutine LATBT
(Sheet 1 of 2)

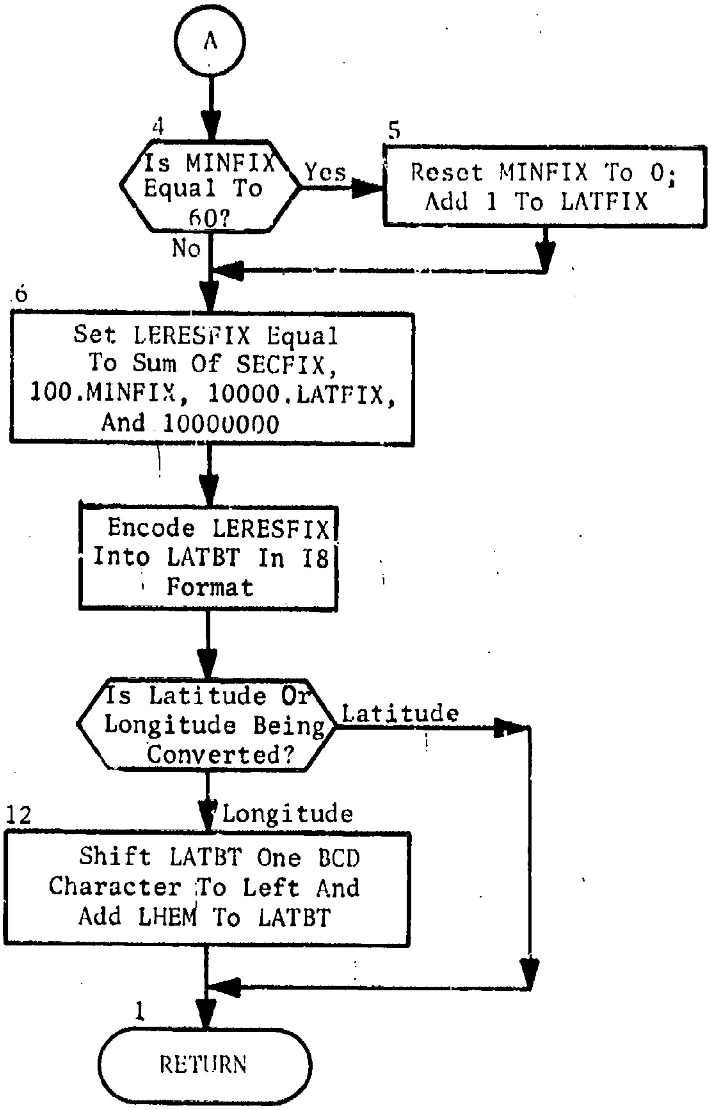


Fig. 224. (cont.)
(Sheet 2 of 2)

FUNCTION LATS

PURPOSE: To convert decimal latitude or longitude into degrees, minutes, and seconds, north, or south, or east or west, respectively.

ENTRY POINTS: LATS, LONS

FORMAL PARAMETERS: A - Latitude or longitude as a decimal number of degrees. A is negative if latitude is south and A is greater than 180 if longitude is east.

COMMON BLOCKS: None

SUBROUTINES CALLED: NOBLANK

CALLED BY: INTRFACE

Method

If entry LATS is called and A is positive, ALAT is set equal to A and latitude is north so LHEM is equal to 1RN. Otherwise, ALAT is set to -A and latitude is south so LHEM is set to 1RS.

Similarly, if entry LONS is called, and A is less than or equal to 180, ALAT is set equal to A and longitude is west so LHEM is set to 1RW. Otherwise, ALAT is set to 360 - A and longitude is east so LHEM is set equal to 1RE.

In all four cases, LATS then converts ALAT, which is in decimal degrees between 0 and 180, to integer degrees, minutes and seconds (LDEG, MIN, and LSEC). Then it codes these values into one number -- LAT -- by setting LAT equal to the sum of LSEC, 100·MIN and 10000·LDEG. Finally, LAT and LHEM are encoded into one BCD number -- LATS -- and NOBLANK is used to replace leading blanks in LATS and zeros.

Function LATS is illustrated in figure 225.

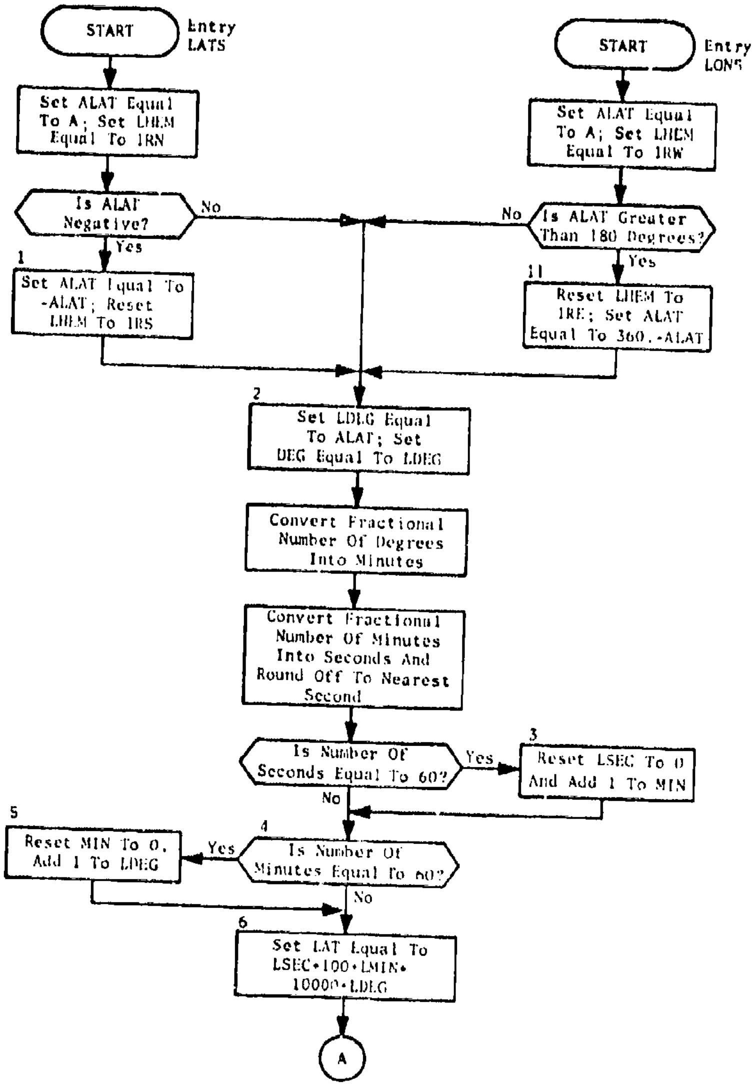


Fig. 225. Function LATS
(Sheet 1 of 2)

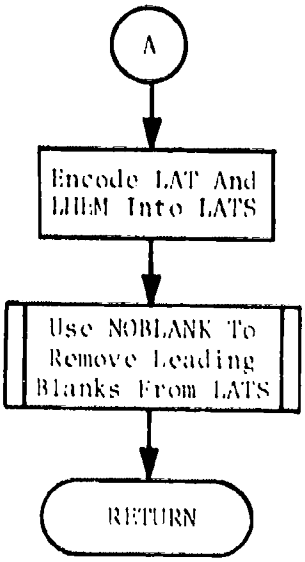


Fig. 225. (cont.)
(Sheet 2 of 2)

FUNCTION NOBLANK

PURPOSE: To convert an integer which may be preceded by blanks into right-justified BCD code preceded by zeros

ENTRY POINTS: NOBLANK

FORMAL PARAMETERS: M - M is the integer to be converted to BCD code by NOBLANK

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: INTRFACE, KNOBLANK, LATS, YLDFRAC

Method

The first time NOBLANK is called, IFIRST is reset from 1 to 2, IBLANK is set to BCD blanks, and the array IB is set by performing the logical .AND. operation between IBLANK and each item in the MASK array.

If this is not the first time NOBLANK has been called, the step above is skipped. Then the octal representation of the integer M is checked two digits at a time beginning with the leftmost pair to determine whether M is preceded by any blanks. If the first pair of digits is a 60 (and hence a blank), NOBLANK replaces the 60 by 00 and proceeds to check the two digits immediately to the right of the first pair for a blank. Again, if this pair of digits is a 60, it is replaced by 00 and the above procedure is repeated again. As soon as NOBLANK tests a pair of digits in M and determines that it is not a blank, NOBLANK is set equal to M (in which all previous blanks are now zeros) and control is returned to the calling subprogram.

Function NOBLANK is illustrated in figure 226.

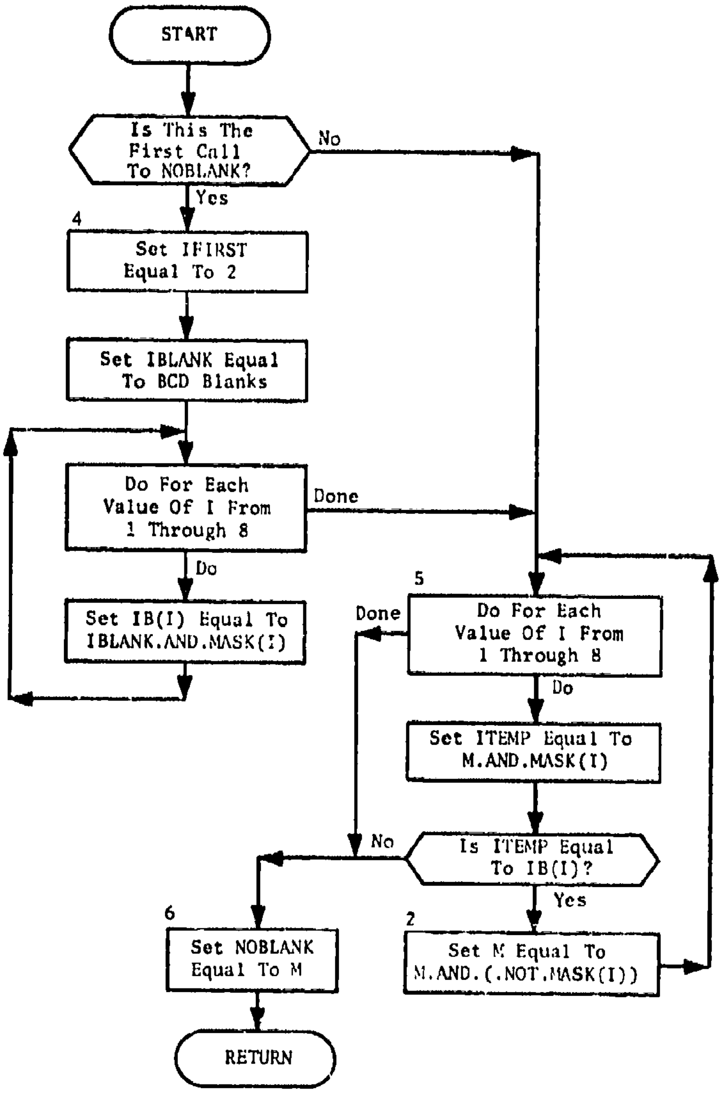


Fig. 226. Function NOBLANK

FUNCTION NOFFSYS

PURPOSE:	To prepare information for the offensive systems table output at the end of INTRFACE.
ENTRY POINTS:	NOFFSYS
FORMAL PARAMETERS:	IT - Plane type
COMMON BLOCKS:	MASTER, OFFSYS, PAYLOAD, PAYLOAD2, STUB
SUBROUTINES CALLED:	KNOBLANK
CALLED BY:	INTRFACE

Method

The first time NOFFSYS is called, it initializes the MASK array for values of I between 1 and NPAYLOAD and initializes NT to zero.

Then, for all calls to NOFFSYS, a final mask -- MSK -- is formed from MASK(LPAYLOAD) and from IT -- the plane type. This mask is compared with previously set values of MASKO and if it matches MASKO for some index-J, NOFFSYS is set equal to KNOBLANK(J) and ICOUNT(J) is incremented by one.

Otherwise MSK corresponds to a new offensive system and the offensive system counter NT is incremented. Also the arrays MASKO, JHTYP, IPGTYP and ICOUNT are updated to reflect the new system. Finally NOFFSYS is set equal to KNOBLANK(NT).

In the unlikely event that NT exceeds 100 -- the present size of the offensive systems table--the message "NO OF OFFENSIVE SYSTEMS EXCEEDS TABLE SIZE" will be printed and NOFFSYS will be filled with BCD blanks.

Function NOFFSYS is illustrated in figure 227.

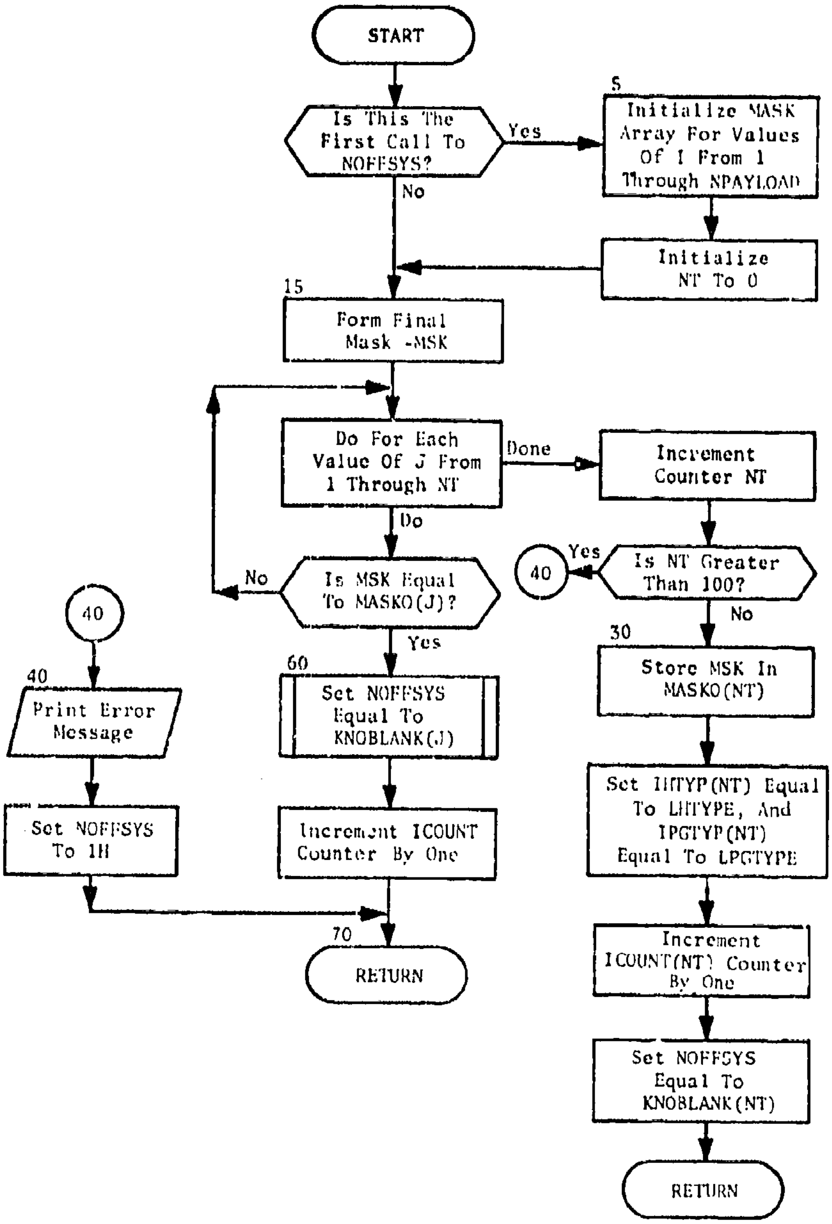


Fig. 227. Function NOFFSYS

FUNCTION NOP

PURPOSE: To find the operation code associated with event type I.

ENTRY POINTS: NOP

FORMAL PARAMETERS: I - Event type

COMMON BLOCKS: MODE, TAB

SUBROUTINES CALLED: KNOBLANK

CALLED BY: INTRFACE

Method

First, if NOPSHOT equals 3 or 4, the values of ITAB(J) for J=18, 19, 20, and 21 are reset to NOPSHOT and NOPSHOT is reset to zero.

In any case NO is set equal to ITAB(I+1) and KNOBLANK is called to convert NO to NOP (which is NO in BCD code with initial blanks replaced by zeros).

Finally, if I is 19, MODE is set equal to 4 or if I is 18, MODE is set equal to 1.

Function NOP is illustrated in figure 228.

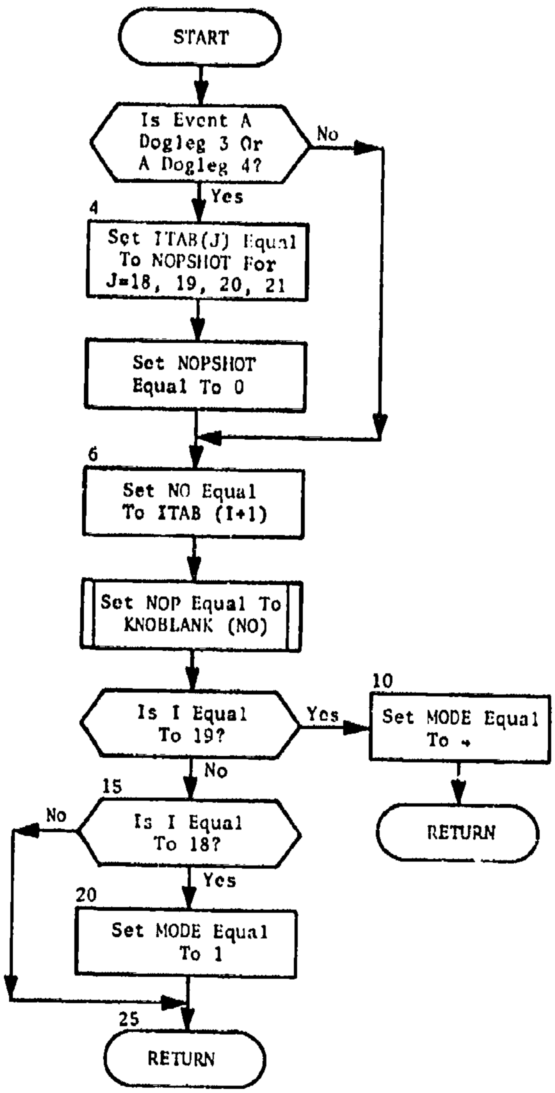


Fig. 228. Function NOP

482-548 O - 72 - 41

FUNCTION NPLNETYP

PURPOSE: To convert the missile type or weapon type index from word 8 in the header block of a PLANTAPE record into a plane type code.

ENTRY POINTS: NPLNETYP

FORMAL PARAMETERS: I - The missile type or weapon type index on the current PLANTAPE record

COMMON BLOCKS: CUMNO, STUB

SUBROUTINES CALLED: None

CALLED BY: INTRFACE

Method

If LCLASS is 1, then I is the missile type and NPLNETYP is set equal to I.

Otherwise I is a weapon type index for a bomber or tanker and NPLNETYP is set equal to the sum of I and ICUMNO(LCLASS-1) the cumulative number of types in class LCLASS.

Function NPLNETYP is illustrated in figure 229.

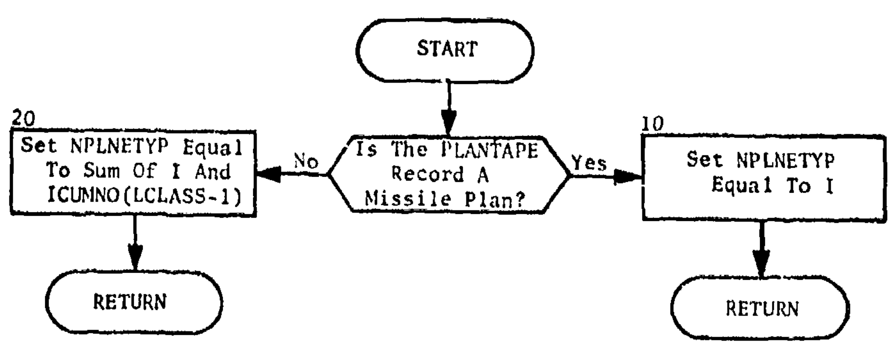


Fig. 229. Function NPLNETYP

FUNCTION NTIME

PURPOSE:	To convert decimal time in hours into integer representing hours and minutes
ENTRY POINTS:	NTIME
FORMAL PARAMETERS:	T - Floating point number of hours
COMMON BLOCKS:	None
SUBROUTINES CALLED:	KNOBLANK
CALLED BY:	INTRFACE

Method:

Function NTIME converts time T to hours and minutes (IHR and MINI). It then codes IHR and MINI into the integer NT by performing the sum of MINI and 100·IHR. Finally, NTIME is obtained by using KNOBLANK to convert NT into BCD code.

Function NTIME is illustrated in figure 230.

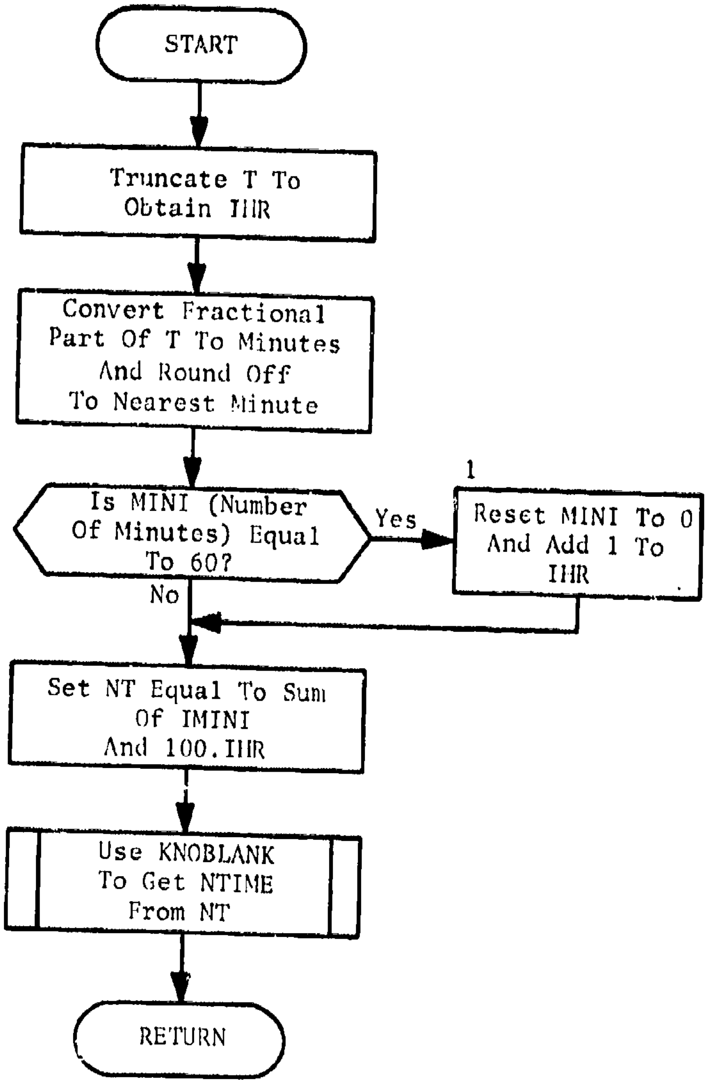


Fig. 230. Function NTIME

SUBROUTINE PRNTOFFS

PURPOSE: To print offensive system number table.

ENTRY POINTS: PRNTOFFS

FORMAL PARAMETERS: None

COMMON BLOCKS: OFFSYS

SUBROUTINES CALLED: None

CALLED BY: INTRFACE

Method:

First PRNTOFFS prints the table title and headings on a new sheet of printer paper. Then for each offensive system it processes and outputs a line in the "Offensive System Table." Note that the values NOBOMB1, IWHD1, NOBOMB2, IWHDZ, NASM, and IASM were coded into MASKO(I) for offensive system I in function NOFFSYS.

Subroutine PRNTOFFS is illustrated in figure 231.

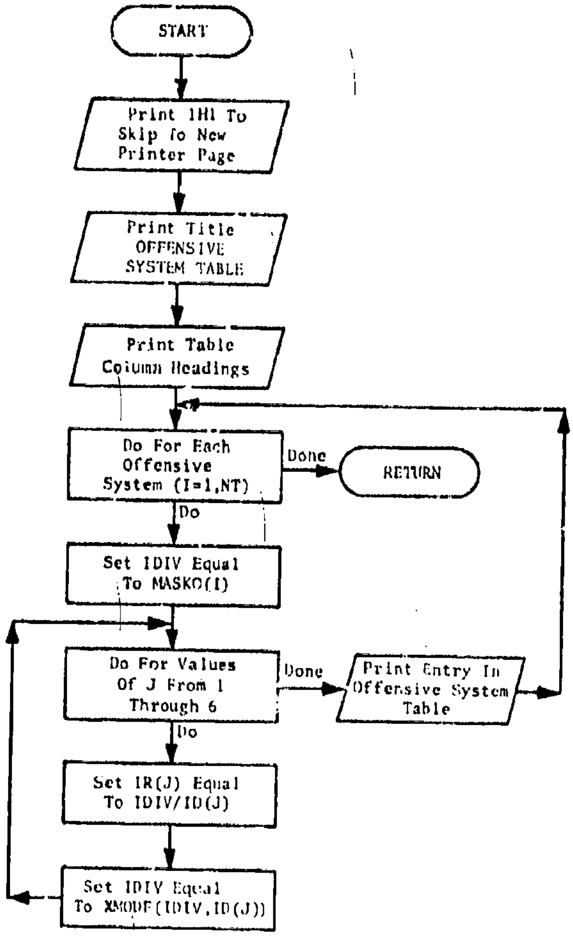


Fig. 231. Subroutine PRNTOFFS

SUBROUTINE YLDFRAC

PURPOSE: If weapon is MRV missile, YLDFRAC calculates its equivalent yield.

ENTRY POINTS: YLDFRAC

FORMAL PARAMETERS:
IWH - Weapon type index
IYLD - Weapon yield
IFRAC - Fission Fraction

COMMON BLOCKS: FRACYLD, PAYLOAD, STUB

SUBROUTINES CALLED: NOBLANK

CALLED BY: INTRFACE

Method:

If the weapon is not an MRV missile, YLDFRAC sets IYLD equal to YIELD(I) and IFRAC equal to FISFRAC(I) and returns processing control to INTRFACE.

Otherwise it sets FNB equal to the number of warheads, decodes YIELD (IWH) into an integer KYLD, and calculates equivalent yield for the MRV by multiplying KYLD by FNB raised to the 1.5 power. Then KYLD is encoded into JYLD and leading blanks are removed from JYLD by NOBLANK to get IYLD. Similarly, FISFRAC(I) is decoded into ITEMP, and ITEMP is scaled down by dividing it by the square root of FNB and rounding off to the nearest integer. Finally, ITEMP is encoded into IFRAC and NOBLANK is used to remove leading blanks from IFRAC.

Subroutine YLDFRAC is illustrated in figure 232.

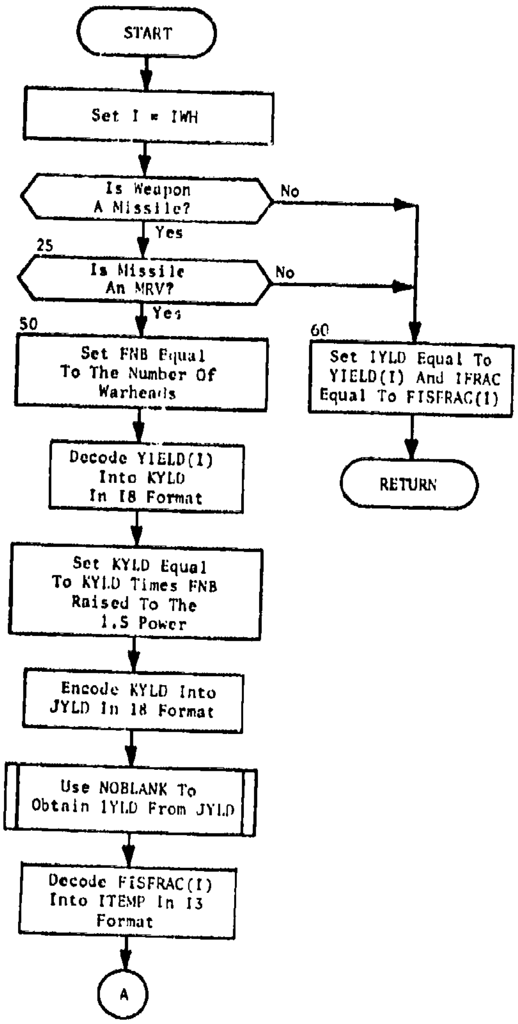


Fig. 232. Subroutine YLDFRAC
(Sheet 1 of 2)

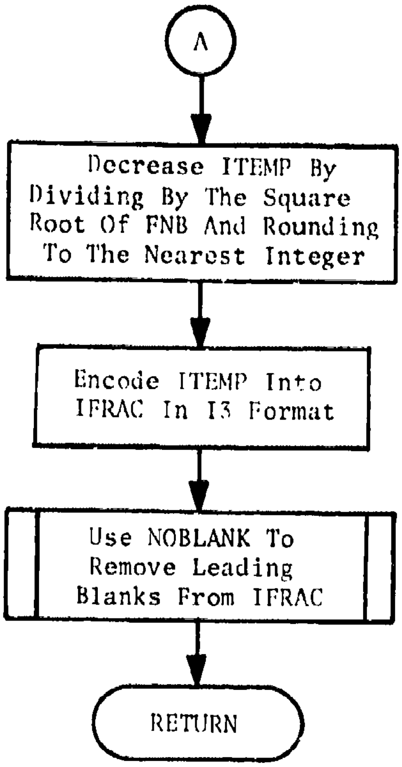


Fig. 232. (cont.)
(Sheet 2 of 2)

CHAPTER 11
PROGRAM TABLE

PURPOSE

Program TABLE is one of the two programs (TABLE and INTERFACE) which provide an interface between QUICK and two external simulators used in RISOP/SIOP gaming; i.e., the Event Sequenced Program (ESP) and the Nuclear Exchange Model (NEMO).

Specifically, program TABLE reads either the INDEXDB tape produced by program INDEXER or the INMODDB tape produced by program BASEMOD and records, in abbreviated form, selected data concerning weapon systems and targets described therein. The extracted data are written on an output tape, TABLTAPE. This program performs no other functions and is not required to operate the QUICK system. However, because it summarizes part of the indexed data base, it enables the user to review the data base before embarking on plan generation if he so chooses.

INPUT FILE

The sole input file is the INDEXDB tape produced by program INDEXER or the INMODDB tape produced by program BASEMOD. The format of both these files is identical and is described as the output of INDEXER in Chapter 7 of the Data Input Subsystem portion of this manual. For the remainder of this chapter, the input file is assumed to be the INDEXDB tape. The INMODDB tape replaces this file only if the user has used program BASEMOD after program INDEXER to modify the indexed game data base.

The data extracted from the input file include descriptions of targets, missile and bomber launch bases, delivery vehicles (missiles and bombers), and weapon characteristics.

The sole user-input parameter to program TABLE specifies the attacking side for the current plan. Thus, TABLE is run once for each side.

OUTPUT FILE

The sole output file is the TABLTAPE. This tape is written as five data lists: Target list (F1TARGET), Vehicle Characteristics list (F1VEHIC), Weapon Characteristics list (F1WEAPON), Missile Base list (F1MIBASE), and Bomber Base list (F1BASE). The lists are placed on the TABLTAPE in the above order. Each entry in each list is written as one 80-character BCD logical record. (The TABLTAPE consists of only one file; the lists are not separated into separate files.)

The information on the TABLTAPE is also printed on the standard output file to provide hard copy output of these lists. Figures 233 through 237 display the format of the 80-character records for each list.

CONCEPT OF OPERATION

Program TABLE is a very simple processor. It merely reads through the input data base one item at a time. All indexed items on the defending side are added to the target list (F1TARGET). The values of appropriate attributes are encoded into the 80-character entry for this list and written on the TABLTAPE directly.

The second and third lists on the TABLTAPE, Vehicle Characteristics list (FIVEHIC) and Weapon Characteristics list (F1WEAPON), are maintained in core during the operation of program TABLE. These lists are stored in common block /111/ in arrays TABVEH and TABWEP, respectively.

The fourth list, the Missile Base list (F1MIBASE), is stored temporarily on a scratch file, OUTAP2. This file, produced by the QUICK filehandler on the CDC 814 disk, uses filehandler buffer number 2. The 80-character list entries are output as a 10-word data block. In the closing phases of TABLE processing, this scratch file is read and the data are transferred to the TABLTAPE.

The fifth and final list on the TABLTAPE, the Bomber Base list (F1BASE), is stored temporarily on a scratch file OUTAP3. This file uses filehandler buffer 4 and is used in the same manner as OUTAP2.

TABLE performs some simple error checking and data conversion. The number of entries in the vehicle and weapon tables is checked to prevent table overflow. In the vehicle table, the circular error prorability (CEP) is converted from nautical miles (on the INDEXDB) to hundreds of feet

(TABLTAPE). In the weapon table, the yield is converted from megatons to kilotons and the fission fraction is converted from a fraction (between 0.0 and 1.0) to hundredths (between 0 and 100).

Subroutine HELP is used to convert latitudes and longitudes. On the INDEXDB tape, these attributes are stored in QUICK system format. In this format, latitudes are expressed in degrees and fractions of degrees with north latitude positive and south latitude negative. Longitudes are expressed in degrees and fractions of degrees ascending in a westward direction from the Greenwich Meridian in a range from 0.0 to 360.0 degrees.. Subroutine HELP converts data from this format to the standard degrees/minutes/seconds/direction format.

COMMON BLOCK DEFINITION

Common blocks /DIRECTRY/ and /PROCESS/ of the data base handling package are used in processing the indexed data base (INDEXDB). These blocks are described in Appendix A, Programming Specifications Manual, Volume I. In addition, /111/, described below, is used for internal processing.

BLOCK	ARRAY	DESCRIPTION
111	TABTAR(10)	Temporary storage for single entry in target list (F1TARGET)
	TABMIS(10)	Temporary storage for single entry in missile base list (F1MIBASE)
	TABVEH(800)	Vehicle characteristics list (F1VEHIC)
	TABBAS(10)	Temporary storage for single entry in bomber base list (F1BASE)
	TABWEP(700)	Weapon characteristics list (F1WEAPON)

COLUMNS	DESCRIPTION	REMARKS
1-8	Format and table name	F1TARGET
9	Side	1 = BLUE 2 = RED
10-14	Line number	1 to 9999
15	Blank	
16-23	Target designator code (Desig/CNTRYLOC/Flag)	2 Alpha 3 numeric 2 Alpha 1 numeric
24	Blank	
25-31	Latitude	Degrees, minutes, seconds, S if south, blank if north
32-39	Longitude	Degrees, minutes, seconds, E if east, W if west
40-46	Blank	
47-54	Target name	8 characters only
55-59	Category code	5 numeric
60-61	Country code	2 Alpha
62-67	Major reference number	6 numeric
68	Blank	
69-70	Task	2 Alpha
71-72	Blank	
72-76	Index number (INDEXNO)	1-12,000 assigned by INDEXER
77	Blank	
78-80	Complex number	1-999 assigned by INDEXER

Fig. 233. Target List (Program TABLE)

COLUMNS	DESCRIPTION	REMARKS
1-8	Format and table name	F1VEHIC
9	Side	1 = BLUE 2 = RED
10-14	Line number	1-9999
15	Card number	1
16-20	Plane type	Type number = 1 to 99
21-55	Blank	
56-58	CEP Mode 1 or CEP Ms1*	Hundreds of feet-0 to 999
59-61	Blank	
62-64	CEP Mode 4*	Hundreds of feet-0 to 999
65-67	Blank	
68-75	Vehicle type name	8 characters
76-80	Blank	

*Mode 1 (high altitude) and Mode 4 (low altitude) refer to bomber flight profiles. QUICK permits only one value of CEP for each type bomber. This assigned CEP is entered for both modes (cc 56-58 and 62-64)

Fig. 234. Vehicle Characteristics List (Program TABLE)

COLUMNS	DESCRIPTION	REMARKS
1-8	Format and table name	F1WEAPON
9	Side	1 = BLUE 2 = RED
10-14	Line number	1 to 9999
15	Blank	
16-19	Weapon number	Weapon number = 1 to 50 (WHDTYPE)
20	Weapon type	0 = Bomb 1 = ASM 2 = Decoy
21-37	Blank	
38-43	Weapon yield	Kilotons
44-46	FFRATIO	000-100
47-80	Blank	

Fig. 235. Weapon Characteristics List (Program TABLE)

COLUMNS	DESCRIPTION	REMARKS
1-8	Format and table name	F1MIBASE
9	Side	1-BLUE 2-RED
10-14	Line number	1 to 9999
15	Blank	
16-20	Base identification number	QUICK index number INDEXNO (1-12000)
21	Blank	
22-28	Latitude	Degrees, minutes, seconds, S if south, blank if north
29-36	Longitude	Degrees, minutes, seconds, E if east, W if west
37-40	Blank	
41	Blank	
42-43	Missile type (ICODTYPE)	Two-digit code.
44-45	Blank	
46	Beginning sortie number	Always 1
47	Fixed slash (/)	
48-49	Ending sortie number	1 to 99
50	Blank	
51	Hard or soft site (vulnerability)	H or S
52	Blank	

Fig. 236. Missile Base List (Program TABLE)
(Sheet 1 of 2)

COLUMNS	DESCRIPTION	REMARKS
53	First or second salvo	All sites listed once only; hard sites are first salvo only; A(1) indicates all missiles at the base are scheduled for the first salvo; A(2) indicates the missiles are scheduled 50% first salvo and 50% second salvo
54-59	Blank	
60-67	Target name	8 characters only
68-69	Blank	
70-71	Country location code	2 Alpha Characters
72-80	Blank	

Fig. 236. (cont.)
(Sheet 2 of 2)

462-546 O - 72 - 42

COLUMNS	DESCRIPTION	REMARKS
1-6	Format and table name	F1BASE
7-8	Blank	
9	Side	1 = BLUE 2 = RED
10	Blank	
11-14	Base number (NUMBAS)	
15	Blank	
16-20	Base identification number	QUICK index number INDEXNO (1-12000)
21	Blank	
22-28	Latitude	Degrees, minutes, seconds, S if south, blank if north
29-36	Longitude	Degrees, minutes, seconds, E if east, W if west
37	Blank	
38	Red launch command	1 = SLBM 2 = LRA 3 = TAC
39	Blank	
40	Base functions (either home base or dispersal base)	X = yes: Blank or zero = no; Note: differentiation between a "home base" and a "dispersal base" is not made
41-43	Blank	
44	Tanker	1 Alpha characters
45-59	Blank	
60-67	Target name	8 Alpha characters
68-69	Blank	
70-71	Country Location	2 Alpha characters

Fig. 237. Bomber Base List (Program TABLE)

PROGRAM TABLE

PURPOSE: This routine retrieves, reformats, and writes the information required for the TABLTAPE.

ENTRY POINTS: TABLE

FORMAL PARAMETERS: None

COMMON BLOCKS: 111, DIRECTRY, EDITERM, ITP, MYIDENT, NOPRINT, PROCESS

SUBROUTINES CALLED: HELP, INITAPE, INITEDIT, INPITEM, ITLE, NEXTITEM, RDARRAY, SETREAD, SETWRITE, TERMTAPE, WRARRAY

CALLED BY: Operating System; this is a main program

Method

This routine is the main processor. The processing is quite straightforward. The input data base is investigated item by item. A series of checks determines if the item is a target, launch base, or weapon. If not, the item is ignored. If the item is one of these, control transfers to a part of this routine which reformats the appropriate attribute values into the form required on the TABLTAPE.

Three local arrays are used in this process:

VEH(200) - A logical array; set true if vehicle type has already been processed to vehicle table

NYLD(50) - Yield in kilotons for each warhead

NFFRAC(50) - Fission fraction in hundredths for each warhead.

The variables in numbered common /111/, described earlier in this chapter, are placed in numbered common to obtain more storage for program TABLE by overlaying the loader. Figure 238 is a flowchart of this routine.

In the series of statements preceding statement 29 several calls on utility subroutine ITLE are made. These calls look up the index of various attributes in the data base directory (array ATTNAME in common /DIRECTRY/). These indices are used to retrieve the attribute values from the VALUE array in common /PROCESS/. This mode of operation obviates the need for processing the TABLE source code with utility program DECLARES.

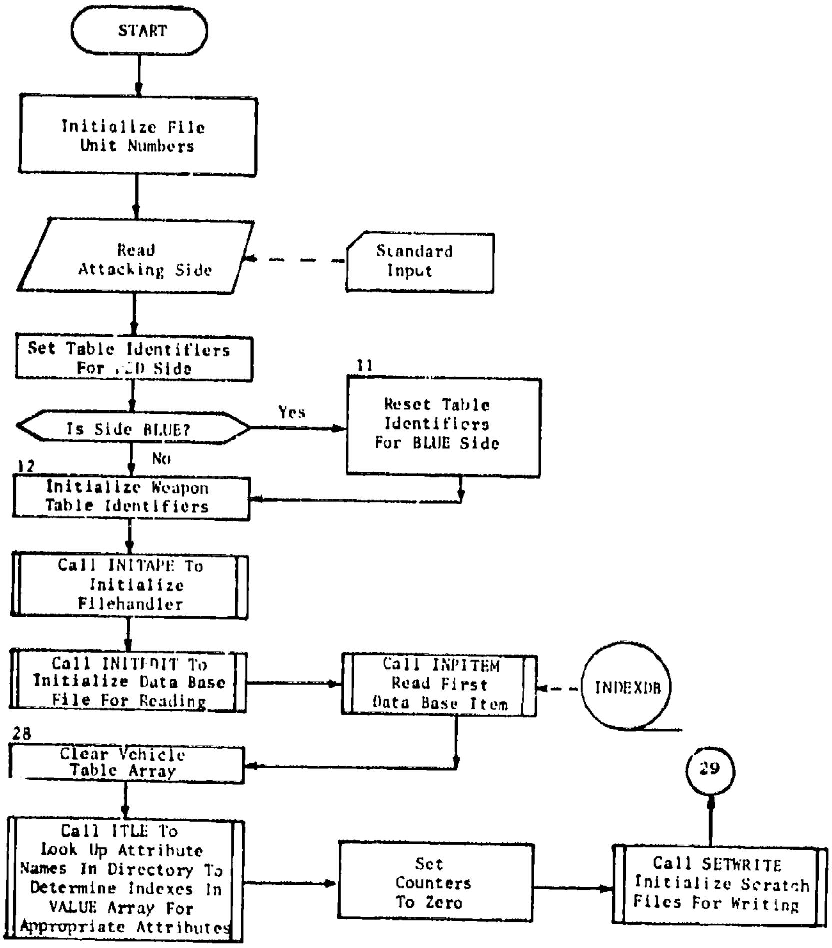


Fig. 238. Program TABLE
Part I: Initialization

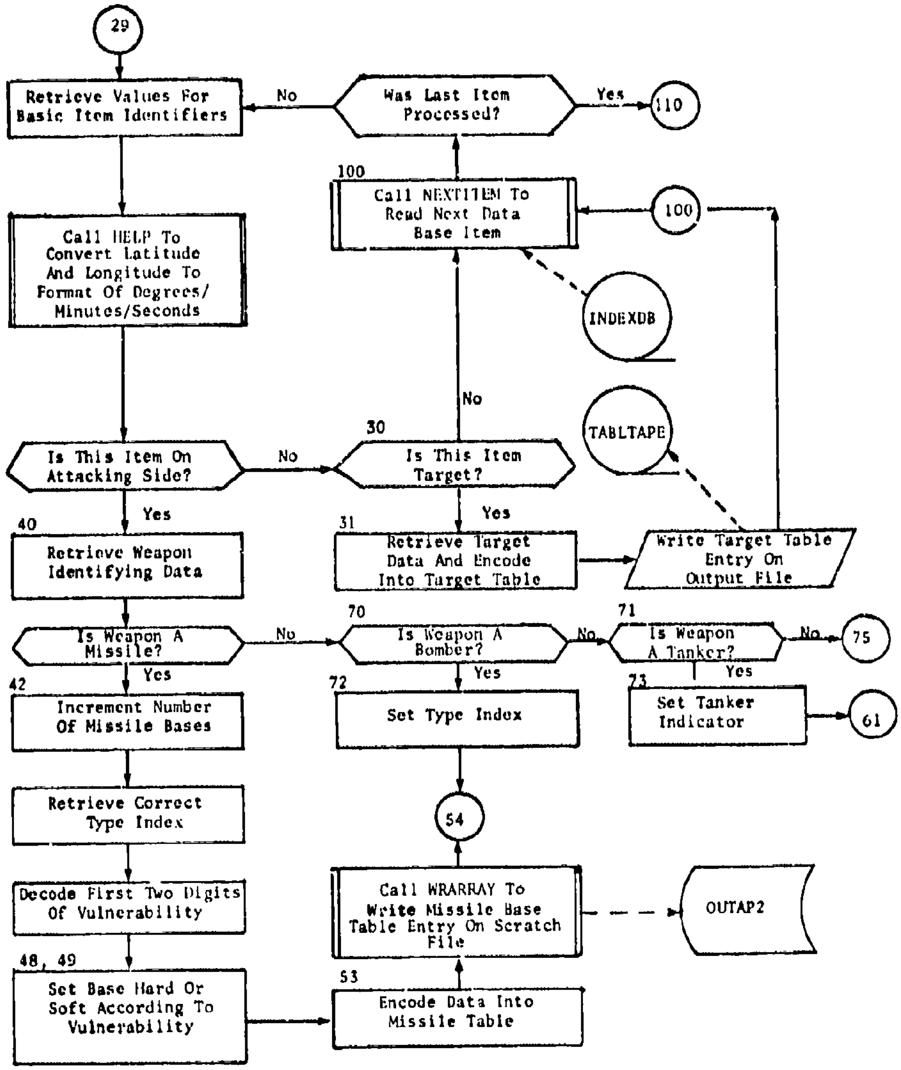


Fig. 238. (cont.)
Part II: Basic Item Processing

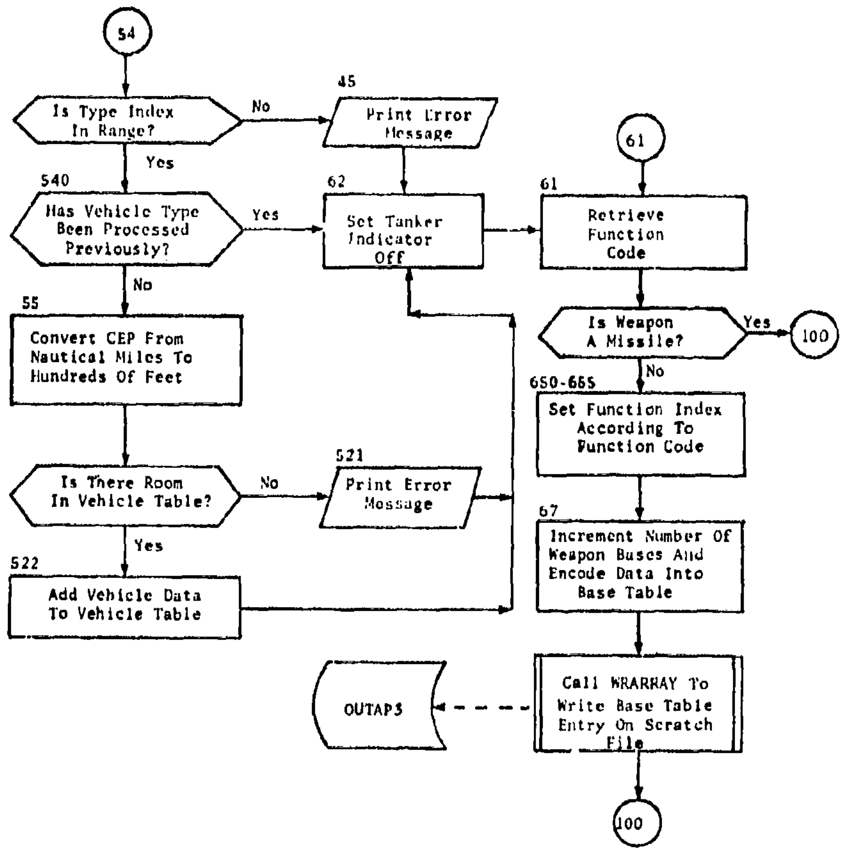


Fig. 238. (cont.)
Part III: Weapon Vehicle Processing

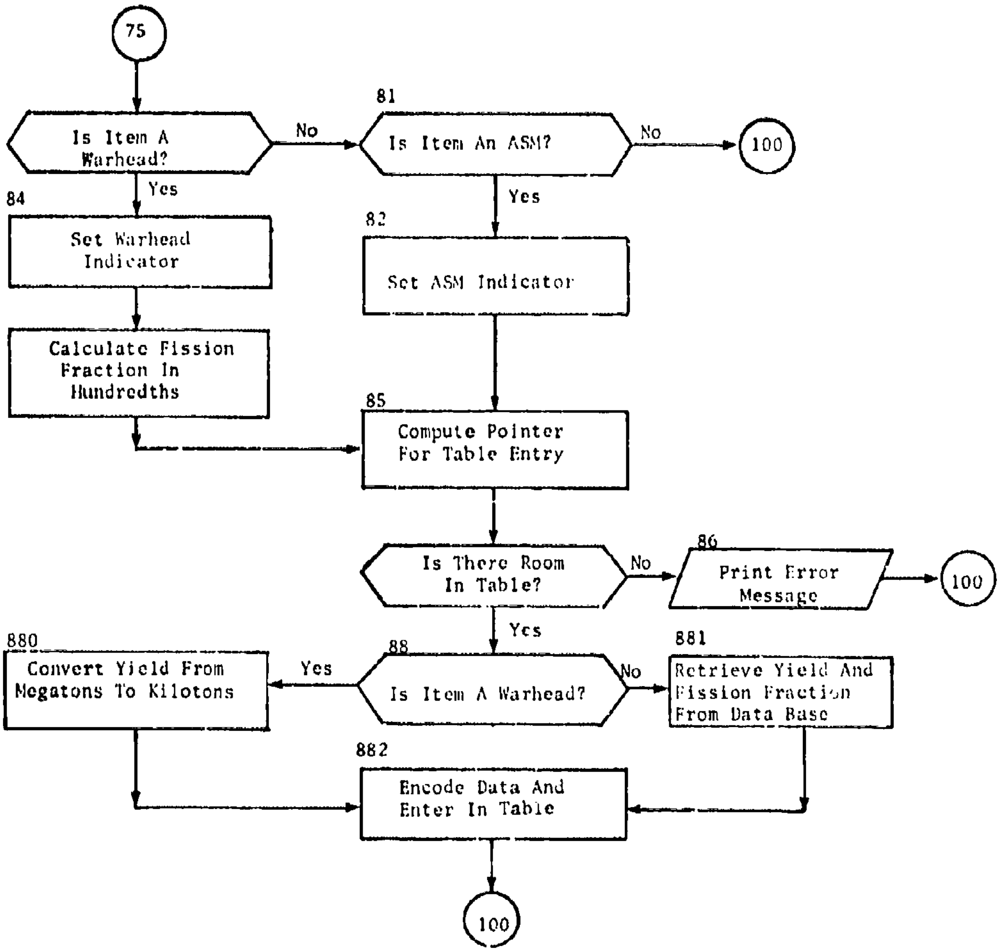


Fig. 238. (cont.)
Part IV: Warhead and ASM Processing

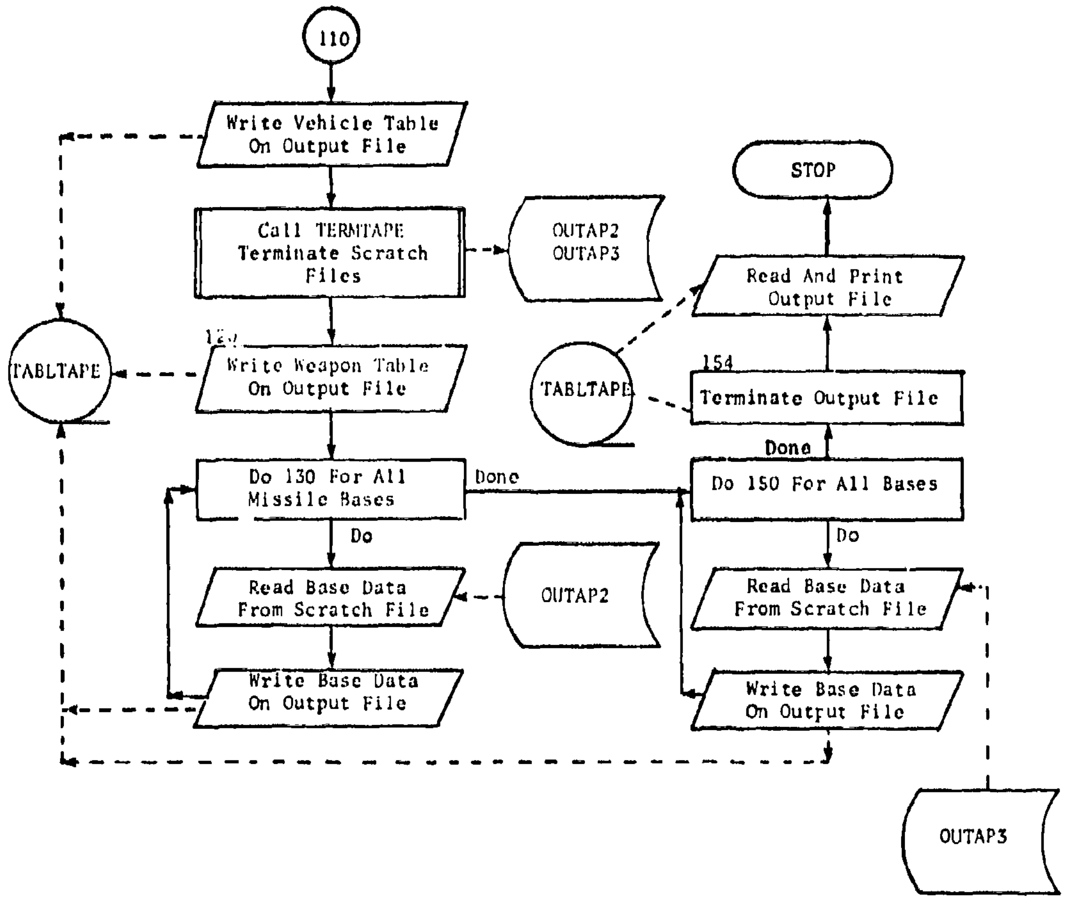


Fig. 238 (cont.)
Part V: Termination Processing

SUBROUTINE HELP

PURPOSE:	To translate latitude or longitude from floating point format to integer format.
ENTRY POINTS:	HELP
FORMAL PARAMETERS:	DEG - Input degrees in floating point format KDONE - Output translation of DEG to integer format LTEST - Input flag set to: 0 - for longitude conversion 1 - for latitude conversion
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	TABLE

Method

Each execution of subroutine HELP translates a value for latitude or longitude from floating point format to integer format. The subroutine's input and output are transmitted via the three formal parameters: DEG, KDONE and LTEST.

The flowchart for HELP is shown in figure 239.

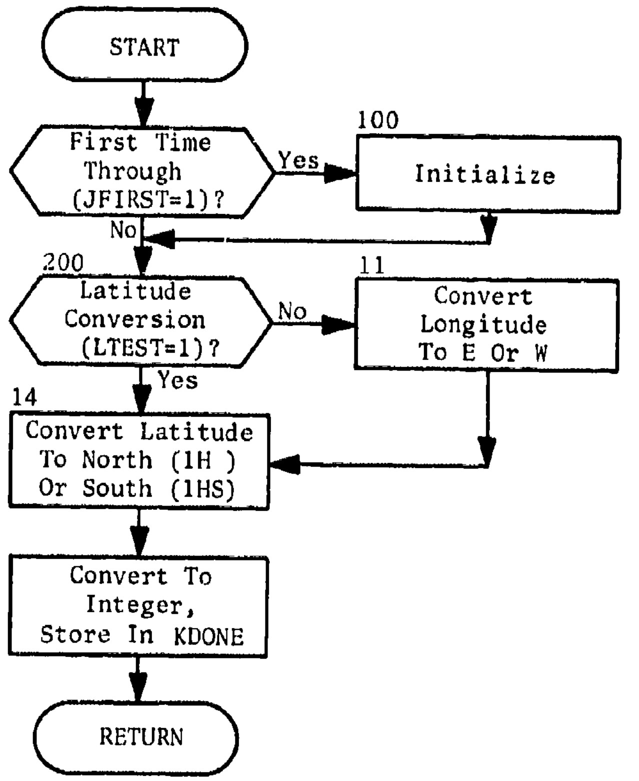


Fig. 239. Subroutine HELP

APPENDIX A
QUICK ATTRIBUTE NAMES AND DESCRIPTIONS

ATTRIBUTE NAME	DESCRIPTION
ABRATE	Probability of aircraft in-flight abort per hour of flying time
ADBLI	ALERTDBL probability for initiative attack
ADBLR	ALERTDBL probability for a retaliatory attack
ADEFCMP	Area ballistic missile defense (BMD) component index (radar or missile launch site)
ADEFZON	Area ballistic missile defense (BMD) zone number
AGX	Offset X-coordinate of AGZ (fiftieths of nautical miles)
AGY	Offset Y-coordinate of AGZ (fiftieths of nautical miles)
AHOB	Actual height of burst of weapon (air or ground)
ALERTDBL	Probability of destruction before launch (DBL) of alert delivery vehicle (missile or bomber)
ALERTDLY	Delay of alert vehicle before commencing launch (hours)
AREA	Area of a bomber defense ZONE (millions of nautical miles^2)
ASMTYPE	Air-to-surface missile type
ATTRCORR	Attrition parameter for a bomber corridor (probability of attrition per nautical mile)
ATTRLEG	Attrition parameter for each route leg in bomber sortie (probability of attrition per nautical mile)
ATTRSUPF	Amount of original attrition that remains after defense suppression

ATTRIBUTE NAME	DESCRIPTION
AZON1	First area defense zone covered by a BMD long-range radar
AZON2	Second area defense zone covered by a BMD long-range radar
AZON3	Third area defense zone covered by a BMD long-range radar
BCODE	Code indicating the outcome of a simulated bomber event
BENO	Bombing encyclopedia number
BLEGNO	Index to boundary line segment
CATCODE	Category Code as reflected in Joint Resource Assessment Data Base (JAD)
CCREL	Regional reliability of offensive command and control (probability)
CEP	Circular error probable (CEP), delivery error applicable to bomber and missile weapons (nautical miles)
CLASS	Class name assigned identify sets of TYPES in data base
CLASST	Target CLASS
CNTRYLOC	Country code for country where item is located
CNTRYOWN	Country code for country which owns the item
CNTYLOCT	Target country code for country where the target is located
CNTYOWNT	Target country code for country which owns the target
CODE	Outcome code for a general event used in simulation
CPACTY	Capacity of a bomber recovery base (number of vehicles)

ATTRIBUTE NAME	DESCRIPTION
DATEIN	Earliest date in inventory (year)
DATEOUT	Latest date in inventory (year)
DEFRANGE	Typical range of interceptors at defense bases near a corridor (nautical miles)
DELAY	Delay time (e.g., launch delay time) (hours)
DELTA	Time interval between successive vehicle launches from the same base (missile or bomber) (hours)
DESIG	Target designator code; e.g., AB100, which uniquely identifies each target element included in the data base
DGX	Offset X-coordinate of desired ground zero (DGZ) (fiftieths of nautical miles)
DGY	Offset Y-coordinate of DGZ (fiftieths of nautical miles)
DHOB	Height of burst of weapon (0-ground, 1-air)
EFECNES1 } EFECNES2 }	Attributes assigned to fighter interceptor units (ICLASS = 5 in the data base): the value EFECNES1 or EFECNES2 is assigned to the attribute EFECTNES depending on value of BASEMOD input parameter POSTURE (if POSTURE=1, EFECNES1 is used; otherwise EFECNES2 value is assigned)
EFECTNES	Air defense capability (arbitrary scale) established by user to indicate relative effectiveness of air defense command and control installations and fighter interceptor bases
EVENT	Index to event type
EVENTN	Index to type of event which did not occur
FFRAC	Fission fraction (fission yield/total yield)
FLAG	Numeric code (1 through 9 permitted) used to impose restrictions on the allocation of weapons within QUICK

ATTRIBUTE NAME	DESCRIPTION
FLTNO	Flight number for a sortie
FUNCTION	Operational application code for a weapon system (e.g., ICBM)
FVALH1	Fraction of value of target in first hardness component
FVALT1	Fraction of target value that disappears by T1 (percent)
FVALT2	Fraction of target value that disappears by T2 (percent)
H1	First hardness component of a target (VULN)
H2	Second hardness component of a target (VULN)
HILOATTR	The ratio of the low-altitude attrition rate to the high-altitude rate (decimal fraction)
IALERT	Alert status; 1 = alert, 2 = nonalert
IALT	Altitude index (1 = high, 0 = low)
IATTACK	Selection index for preferential area BMD; 1 forces target selection for defense.
ICLASS	Class index assigned for game
ICLASST	Target class index
ICOMPLEX	Complex index
ICORR	Bomber corridor index number assigned in program PLANSET: 1 - Tactical (FUNCTION=TAC) aircraft corridor (TYPE name DUMMY in the data base) 2 - Naval attack corridor (TYPE name NAVALAIR in the data base) used by bomber units with PKNAV greater than zero >2 - Other corridors used by long range bombers (FUNCTION=LRA)

ATTRIBUTE NAME	DESCRIPTION
IDBL	Index to data tables for time-dependent destruction before launch probability
IDUD	Dud warhead indicator; assigned to weapons which arrive at the target but fail to detonate; 1=dud warhead
IGIW	Indices of General Industrial Worth (IGIW) (dollars)
IGROUP	Group index assigned for weapon grouping during game
IMIRV	Identifying index for system with multiple independently targetable re-entry vehicles
INDEXNO	Index of a data base item (potential target) used during processing to identify the item
INDV	Vehicle index within base
INTAR	Target index (corresponds to INDEXNO)
IPENMODE	Penetration mode; 1 = aircraft uses penetration corridor, 0 = penetration corridor not used
IPOINT	Index to a geographic point
IRECMODE	Recovery mode; 1 = aircraft should plan recovery, 0 = aircraft recovery not planned
IREFUEL	Bomber refueling code
IREG	Index to identify a geographic region
IREP	Reprogramming index (capability of missile squadron)
ISITE	Site number
ITGT	Target index number assigned by Plan Generation subsystem
ITIME	Index to time periods in time dependent DBL data tables

ATTRIBUTE NAME	DESCRIPTION
ITYPE	Type index assigned for game
ITYPET	Target type index
IVULN	Index to vulnerability number table
IWTYP2	Second warhead type
JTYPE	Type index within class
JTYPET	Target type index within class
KORSTYLE	Parameter to adjust mode of corridor penetration
LAT	Latitude (degrees)*
LEGNO	Index to line segment
LINK	The index of a leg linked to the current point
LONG	Longitude (degrees)*
MAJOR	Major reference number as reflected in the Joint Resource Assessment Data Base (JAD)
MAXFRACV	Maximum value of weapon resources to be used relative to target value (in processing MAXCOST=MAXFRACV)
MAXKILL	Desired maximum damage expected for a target
MINKILL	The required minimum damage established for a target

* Latitude and longitude are carried internally in the QUICK system in the following format:

North latitude	0. (equator) to +90. (North Pole)
South latitude	0. (equator) to -90. (South Pole)
East longitude	180. to 360. (Greenwich Meridian)
West longitude	0. (Greenwich Meridian) to 180.

These attributes may be input in either the above format or in standard degree, minute, second, direction format.

ATTRIBUTE NAME	DESCRIPTION
MINOR	Minor reference number as reflected in JAD to identify an item
MISDEF	Number of terminal ballistic missile interceptors for a target
MVA	Manufacturing value added (MVA); indicates the amount of value added by manufacture within a specific area (expressed in U.S. dollars)
MWHDS	Number of missile warheads penetrating area defenses to terminal defense
NADBLI	NALRTDBL for initiative attack
NADBLR	NALRTDBL for retaliatory attack
NAINT	Number of area ballistic missile interceptors at an interceptor launch base
NALRTDBL	Probability of destruction before launch (DBL) of non-alert vehicle
NALRTDLY	Delay of non-alert vehicle before commencing launch (hours)
NAME	Arbitrary alphameric descriptor for any item included in the data base
NAREADEC	Number of decoys per independent re-entry vehicle for area BMD
NASMS	Number of ASMs carried by a bomber
NCM	Number of countermeasures carried by vehicle
NDECOYS	Number of decoys on a bomber or number of decoys per independent re-entry vehicle for terminal BMD
NDET	Number of warheads detonating in current event
NEXTZONE	The adjacent zone to a side of a defense zone
NMPSITE	Number of missiles per site

ATTRIBUTE NAME	DESCRIPTION
NOALERT	Number of vehicles on alert at a base
NOBOMB1	Number of first bomb type carried by vehicle
NOBOMB2	Number of second bomb type carried by vehicle
NOINCOM	Number of delivery vehicles in commission
NOPERSQN	Number of weapon vehicles per squadron
NOPERSQ1 NOPERSQ2 NOPERSQ3	Attributes used in program BASEMOD to compute the value of the attribute NOPERSQN for bomber units; numbers 1, 2, and 3 specify surprise, initiative, and retaliatory attack plans respectively
NPEN	Number of warheads penetrating in current event
NTARG	Number of targets in missile launch event
NTINT	Number of terminal BMD interceptors at target
NWHDS	Number of warheads per independent re-entry vehicle (missiles)
NWPNS	Number of weapons in a group
NWTYPE	Warhead type
PARRIVE	Probability of bomber arrival in current event
PAYLOAD	Index which identifies entire weapon and penetration aid complement on a vehicle
PDES	Probability that launch failure destroys missile
PDUD	Probability a warhead will fail to detonate
PEN	Penetration probability for a weapon
PFPF	Probability of failure during powered flight (missiles)
PINC	Probability that a missile is in commission

ATTRIBUTE NAME	DESCRIPTION
PKMIS	Probability a missile fails to penetrate terminal defense
PKNAV	Single shot kill probability of a weapon against a naval target (a value greater than zero restricts weapon use to naval targets)
PLABT	Probability of vehicle launch abort
PLACE	Index to geographic location of an event
PLACEN	Index to geographic location of an event which did not occur
POP	Population (cities) (thousands)
POSTURE	Force readiness condition
PRABT	Probability of refueling abort
PRIMETAR	Prime target flag; 1 signifies priority target in a complex
PSASW	Destruction before launch probability assigned a weapon for a specified time period
RADIUS	Size descriptor for area targets (nautical miles)
RANGE	Vehicle range (nautical miles)
RANGEDEC	Range decrement for low-altitude aircraft flight (high range/low range)
RANGEREF	Range (nautical miles) of bomber with refueling
REL	Reliability - probability that weapon system will arrive at target given successful launch
RESERVE	Technique used to remove certain targets from weapon allocation when RESERVE = 0
SIDE	Item side name, currently either "RED " or "BLUE"

ATTRIBUTE NAME	DESCRIPTION
SITENO	Site number (currently for individual missile sites)
SPDLO	Speed at low altitude (knots)
SPEED	Speed (knots)
SQNNO	Squadron number
T1	Time of departure of first value component of a target
T2	Time of departure of second value component of a target
T3	Time of departure of third value component of a target
TAIM	Number of aim points perceived by terminal defense in current event
TARDEFHI	Level of local bomber defense at high altitude*
TARDEFLO	Level of local bomber defense at low altitude*
TASK	Target task code indicating targeting priority
TGTSTAT	Indicates target status as dynamic or nondynamic; in simulation status (alive/dead) is maintained for dynamic targets
TIME	Game time at which event occurred (hours)
TIMEN	Time planned for event which did not occur (hours)
TMDEL	Mean delay time to relaunch after a nondestructive aircraft abort (hours)

---

* Arbitrary units scaled by user-input parameter in Plan Generation subsystem. Minimum value 0 for no defense. Highest allowed defense level is + 7.

ATTRIBUTE NAME	DESCRIPTION
TPASW	Time at which a time period ends for DBL data tables; there may be up to 10 time periods for each table
TRETARG	Time required to retarget for known in-flight missile aborts (hours)
TTOS	Total time on station (for a tanker) (hours)
TVUL	Time a missile remains within vulnerable range of launch site (hours)
TYPE	Arbitrary alphameric designator (type name) to identify smallest sets in data base
TYPET	Target TYPE
TYPE1 } TYPE2 }	Attributes assigned fighter interceptor units (ICLASS=5 in the data base): attribute TYPE is assigned the TYPE1 or TYPE2 value based on BASEMOD input parameter POSTURE (POSTURE=1 TYPE1 is used; otherwise TYPE2 value used)
VAL	Relative value of an item within its CLASS as established in the data base by the user
VALU	Game value of an item (assigned in plan generation based on user-input parameters)
VAL1 } VAL2 }	Attributes assigned fighter interceptor units (ICLASS=5 in the data base): attribute VAL is assigned the VAL1 or VAL2 value based on BASEMOD input parameter POSTURE (POSTURE=1, VAL1 is used; otherwise VAL2 value is assigned)
VULN	Vulnerability number
WACNO	World aeronautical chart number
WHDTYPE	Warhead type index assigned in the data base
WHDTYPEN	Warhead type index (used with EVENTN)
YIELD	Yield (MT)
ZONE	An area bomber defense zone enclosed by a set of linked boundary points

APPENDIX B
ENTRY POINTS FOR QUICK UTILITY ROUTINES

This appendix contains an alphabetic listing of the entry points associated with all utility programs and subroutines. Subroutines associated with each of these entry points are indicated below.

ENTRY POINT	TO SUBROUTINE
ABORT	ABORT
ALOCDIR	FILEHNR
ANOTHER	ANOTHER
ATN2PI	ATN2PI
CHANGE	CHANGE
CLOSPIL	CLOSPIL
CLRCMON	CLRCMON
DEACTIV	FILEHNR
DECLARES	DECLARES
DELLONG	DELLONG
DIFFLNG	DIFFLONG
DIFFLONG	DIFFLONG
DISTF	DISTF
DSTF	DSTF
ENDDATA	ENDDATA
ENDTAPE	ENDTAPE
ERAZE	ERAZE
EQUIV	EQUIV
FILEDUMP	FILEDUMP
FILEHNR	FILEHNR
GETCLK	GETCLOCK

ENTRY POINT	TO SUBROUTINE
GETCLOCK	GETCLOCK
GETDATE	GETDATE
GETDF	GETDF
GETLIMIT	GETLIMIT
GETLOC	GETLOC
GETVALU	GETVALU
IGET	IGET
INBUFDK	INBUFDK
INERRDK	INERRDK
INERRTP	INERRTP
INITAP	FILEHNR
INITAPE	FILEHNR
INITEDIT	INITEDIT
INITEDT	INITEDIT
INLABEL	INLABEL
INPITEM	INPITEM
INTERP	INTERP
INTERPGC	INTERPGC
INTRPGC	INTERPGC
IPUT	IPUT
ITLE	ITLE
IWANT	IWANT
KEYMAKE	KEYMAKE
LOCF	LOCF
LOCREAD	LOCREAD
LOCWRIT	LOCREAD
LOCWRITE	LOCREAD
NEWUNIT	NEWUNIT
NEXTAPE	NEXTAPE
NEXTFILE	NEXTFILE

ENTRY POINT	TO SUBROUTINE
NEXTITEM	INITITEM
NEXTITM	INITITEM
NODIRC	NODIRC
NUMGET	NUMGET
OPENSPL	OPENSPL
ORDER	ORDER
OUTBFDK	OUTBFDK
OUTBFTP	OUTBFTP
OUTDF	OUTDF
OUTERDK	OUTERDK
OUTERTP	OUTERTP
OUTFILE	OUTFILE
OUTITEM	OUTITEM
OUTWORDS	OUTWORDS
OUTWRDS	OUTWORDS
PAGESKIP	PAGESKP
PAGESKP	PAGESKP
PRITEM	PRITEM
PRNTBAS	PRNTBASE
PRNTBASE	PRNTBASE
PRNTBSE	PRNTBASE
PRNTDATA	PRNTDTA
PRNTDTA	PRNTDTA
PRNTDIRC	PRNTDIRC
PRNTDRC	PRNTDIRC
PRNTLAB	FILEHNR
PRNTPAGE	PRNTPGE
PRNTPGE	PRNTPGE
RDARRAY	RDARRAY
READDIR	READDIR
RELOADF	RELOADF

ENTRY POINT	TO SUBROUTINE
REORDER	REORDER
SETHEAD	SETHEAD
SETREAD	SETREAD
SETWRIT	FILEHNR
SETWRITE	FILEHNR
SKIP	SKIP
SSKPC	SSKPC
STORAGE	STORAGE
TERMTAP	TERMTAP
TERMTAPE	TERMTAP
TERMTPE	TERMTAP
TIMEDAY	TIMEDAY
TIMEME	TIMEME
WARNING	ABORT
WRARRAY	RDARRAY
WRITEDIR	WRITEDIR
WRITEDR	WRITEDIR
WRWORD	FILEHNR