SECURITY CLASSIFICATION OF THIS PAGE

# AD-A257 660

## REPORT DOCUME

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

2a. SECURITY CLASSIFICATION AUTHORITY

3. DISTRIBUTION/AVAILABILITY OF REPORT

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

ELECTE
OCT 23 1992
S D

4. PERFORMING ORGANIZATION REPORT NUMBER(S)
Kaman Sciences Corp. K-89-58U(R)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

| 6a. NAME OF PERFORMING ORGANIZATION | 6b. OFFICE SYMBOL (If applicable) | 7a. NAME OF MONITORING ORGANIZATION |
|---|---|---|
| Kaman Sciences Corporation | | Air Force Studies & Analyses Agency (AFSAA/SAS) |

| 6c. ADDRESS (City, State, and ZIP Code) | 7b. ADDRESS (City, State, and ZIP Code) |
|---|---|
| 1500 Garden of the Gods Rd.<br>Colorado Springs, CO 80907 | Room 1D 431 Pentagon<br>Washington DC 20330-5420 |

| 8a. NAME OF FUNDING/SPONSORING ORGANIZATION | 8b. OFFICE SYMBOL (If applicable) | 9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER |
|---|---|---|
| | | F49642-89-C-0009 |

8c. ADDRESS (City, State, and ZIP Code)

10. SOURCE OF FUNDING NUMBERS

| PROGRAM ELEMENT NO. | PROJECT NO. | TASK NO. | WORK UNIT ACCESSION NO. |
|---|---|---|---|
| | | | |

11. TITLE (Include Security Classification)
Microcomputer Missile Performance Software System: ASSIGNMENT Computer Software Component - ALM - Maintenance Manual (MM-04) (U)

12. PERSONAL AUTHOR(S)

| 13a. TYPE OF REPORT | 13b. TIME COVERED | 14. DATE OF REPORT (Year, |
|---|---|---|
| | FROM _____ TO _____ | 16 January 1990 |

16. SUPPLEMENTARY NOTATION

389119

92-27783

33 pg

| 17. COSATI CODES | | | 18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number) |
|---|---|---|---|
| FIELD | GROUP | SUB-GROUP | (U) Microcomputer Missile Performance Software (MPS); (U) Computer Software Component (CSC); (U) ASSIGNMENT: (U) ALM. |
| | | | |
| | | | |

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

(U) The Microcomputer Missile Performance Software System (MPS) consists of ten integrated Computer Software Components (CSCs) that are used to assess ballistic missile performance, and to plan, generate, and evaluate timed laydowns of ballistic missile forces on target sets, subject to operational and nuclear vulnerability constraints. Within the MPS, ALM is the CSC responsible for assigning RVs to DGZs and for scheduling missile launches so as to minimize the span of either missile launch times or RV arrival times. The MPS is designed to execute efficiently on AT class microcomputers, running under an MS-DOS operating system. Over 95% of the MPS software is written in FORTRAN, with the remainder being assembly code. This manual provides a comprehensive specification of the ALM CSC and is intended to support the ALM maintenance programmer during the MPS's operational lifetime.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

| 22a. NAME OF RESPONSIBLE INDIVIDUAL | 22b. TELEPHONE (Include Area Code) | 22c. OFFICE SYMBOL |
|---|---|---|
| Major Nancy Snyder | 703/695-9018 | AFSAA/SAS |

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE

MICROCOMPUTER MISSILE PERFORMANCE SOFTWARE SYSTEM

CONTRACT NUMBER F49642-89-C-0009

# ASSIGNMENT

## COMPUTER SOFTWARE COMPONENT

## - ALM -

# MAINTENANCE MANUAL

## (MM-04)

CDRL ITEM NUMBER A006

K-89-58U (R)

16 JANUARY 1990

Kaman Sciences Corporation
1500 Garden of the Gods Rd.
Colorado Springs, CO 80907

KAMAN

MPS

MICROCOMPUTER MISSILE PERFORMANCE SOFTWARE SYSTEM

CONTRACT NUMBER F49642-89-C-0009

# ASSIGNMENT

## COMPUTER SOFTWARE COMPONENT

## - ALM -

# MAINTENANCE MANUAL

## (MM-04)

CDRL ITEM NUMBER A006

K-89-58U (R)

16 JANUARY 1990


PREPARED BY: Tom Schwab
TOM SCHWAB
ALM Lead Engineer

APPROVED BY: Thomas Lundgren
TOM LUNDGREN
MPS Program Manager

# FOREWORD

This is the fourth volume in an eleven-volume set of maintenance manuals for the Microcomputer Missile Performance Software System (MPS). The MPS was developed by Kaman Sciences Corporation, under the direction of the U. S. Air Force Center for Studies and Analyses (AFCSA/SASM), on Contract Number F49642-89-C-0009. This particular volume describes the ALM Computer Software Component (CSC).

## ACKNOWLEDGMENT

The authors wish to express their deep appreciation to the two U. S. Air Force Center for Studies and Analyses (AFCSA/SASM) Contracting Officer's Representatives for this effort, Lt. Col. Arthur T. Hopkins and Major David W. Knieriem, for their direction, support, and substantial contributions to the development of the MPS.

## ABSTRACT

The Microcomputer Missile Performance Software System (MPS) consists of ten integrated Computer Software Components (CSCs) that are used to assess ballistic missile performance, and to plan, generate, and evaluate timed laydowns of ballistic missile forces on target sets, subject to operational and nuclear vulnerability constraints. Within the MPS, ALM is the CSC responsible for assigning RVs to DGZs and for scheduling missile launches so as to minimize the span of either missile launch times or RV arrival times.

The MPS is designed to execute efficiently on AT class microcomputers, running under an MS-DOS operating system. Over 95% of the MPS software is written in FORTRAN, with the remainder being assembly code.

This manual provides a comprehensive specification of the ALM CSC and is intended to support the ALM maintenance programmer during the MPS's operational lifetime.

# TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

## TABLE OF CONTENTS (Concluded)

## LIST OF FIGURES

## LIST OF FIGURES (Continued)

## LIST OF FIGURES (Concluded)

# SECTION 1
# INTRODUCTION

## 1.1 SCOPE

This manual provides a comprehensive specification of the ALM Computer Software Component (CSC) of the Microcomputer Missile Performance Software System (MPS). This specification describes and explains in detail ALM's function within the MPS, the as-built design of the ALM CSC, and the implementation of that design in software. Specifically, it presents and discusses:

- What ALM does, and how it does it (in terms of the physical and mathematical models and algorithms it employs).
- How ALM is structured and what components (modules, common blocks) comprise it.
- How ALM interfaces with other CSCs and what those interfaces are.

This manual is written with two audiences in mind. First, and foremost, this document is intended to support the ALM maintenance programmer during the MPS's operational lifetime in making revisions to ALM to correct errors, to improve efficiency, to incorporate new capabilities, and to accommodate changes in the host hardware and software environments. A secondary audience for this manual are those users who desire more than a casual familiarity with ALM, and more than the ability to simply set-up and execute it. These users typically want a fuller appreciation and understanding of ALM's capabilities and limitations, and its underlying concepts and models.

## 1.2 DOCUMENT OVERVIEW

This manual consists of ten sections and ten appendices organized by specific topics that are of interest, and importance, to the ALM maintenance programmer:

- Section 1 provides an introduction to, and roadmap of, the ALM Maintenance Manual.
- Section 2 presents an overview of the MPS's purpose, structure,and programming languages. It also describes the operating system and hardware configuration for which the MPS is designed.
- Section 3 provides an overview of ALM's functionality, top-level design, and limitations.
- Section 4 identifies and defines the ALM input files and discusses how those files can be created.
- Section 5 outlines ALM's processing sequence, procedures and algorithms,

and provides a tutorial on the underlying concepts and models that ALM employs.

- Section 6 identifies and defines the outputs produced by ALM.
- Section 7 describes the error processing performed by the MPS and describes all ALM error and warning messages.
- Section 8 provides a sample problem that demonstrates ALM's operation and processing options.
- Section 9 describes how the ALM source code can be compiled and linked to produce executable code.
- Section 10 lists all references alluded to in this manual.
- Appendix A identifies and briefly states the purpose of each ALM module.
- Appendix B provides a module cross reference. For each module, this cross reference identifies the modules that it calls and the modules that call it.
- Appendix C provides a complete specification of each ALM module.
- Appendix D lists all common blocks used by ALM and provides a short characterization of the type of data contained in each.
- Appendix E supplies a common block cross reference that shows the common blocks used by each module, and the modules that use each common block.
- Appendix F identifies and defines all variables within each ALM common block.
- Appendix G identifies and defines all parameters unique to the ALM CSC.
- Appendices H, I and J present and define the symbology used in ALM data flow diagrams, function charts, and flowcharts, respectively.

# SECTION 2
# MPS SUMMARY

## 2.1 MPS OVERVIEW

The MPS is an integrated software system for simulating the performance of ballistic missiles, and for assessing the utility, effectiveness and survivability of ballistic missile forces in conducting coordinated attacks in both benign and hostile environments. Specifically, the MPS can be used to determine ballistic missile range capabilities, post-boost vehicle (PBV) footprint dimensions and fuel usage, and reentry vehicle (RV) flight times. It can also be used to report trajectory state vectors, to calculate optimized weapon allocations, to predict RV fratricide in a timed laydown, and to estimate missile flyout attrition in a launch schedule, given a pindown attack scenario. In addition, the MPS can be used to plan, generate and evaluate optimized laydowns of ballistic missile forces on target sets subject to operational and nuclear vulnerability constraints. Finally, the MPS is equipped with a graphics processor that enables users to produce maps and graphs of missile performance, effectiveness and laydown results.

During MPS development, a number of design features were accorded particular attention. As a result of that precedence, the MPS:

- Is user-friendly and graphics oriented.
- Provides users direct access to the full range of MPS capabilities.
- Maximizes user visibility and control of the analysis process.
- Emphasizes fidelity and realism.
- Incorporates a design that facilitates future growth and expansion.

## 2.2 MPS STRUCTURE

The MPS is an integrated system consisting of ten distinct, but complementary Computer Software Components (CSCs). Two of these (MMI and EXEC) are support CSCs that are invoked every time a user employs the MPS. The remaining eight mission CSCs (ALM, CAIN, DAMAGE, GRAPHS, PATRIT, PFRAT, SOADA and TARGET) are independently operable, and are capable of being exercised with user-provided inputs, output files generated by a prior execution of another CSC, or user-modified versions of such output files.

All communications between CSCs are effected via a set of control, status and data files. Figure 2-1 identifies all

MPS files and it shows the connectivity of CSCs through these files. Each mission CSC interfaces directly to the EXEC/MMI through control and status files. The MPS user communicates with the system via a keyboard and a VDT that are tied directly to the MMI. Those files connected to EXEC/MMI with heavy, double-arrowhead lines can be created/modified by the user through the MMI. Graphical depictions of MPS results are provided by various output devices connected directly to the GRAPHS CSC.

## 2.3 CSC DESCRIPTIONS

Each MPS CSC has a well-defined, and easily-differentiated purpose. A brief description of each CSC's purpose, in terms of the functions it performs, is presented in Figure 2-2.

| CSC | DESCRIPTION |
|---|---|
| ALM | Generates timed laydowns of ballistic missile forces that optimize attack timing objectives subject to operational constraints. |
| CAIN | Quantifies nuclear constraints on attack structure due to RV fratricide, and missile flyout attrition resulting from nuclear pindown detonations. |
| DAMAGE | Calculates the expected installation value damaged in an attack, accounting for weapon system reliabilities, RV fratricide and missile attrition. |
| EXEC | Coordinates the activities of all CSCs and superintends CSC execution. |
| GRAPHS | Produces graphical depictions of MPS outputs. |
| MMI | Provides a user-friendly, interactive panel system that allows the user to define a case, and supply all required input data. |
| PATRIT | Evaluates the probability of missile flyout attrition in a launch schedule, given a pindown scenario. |
| PFRAT | Evaluates the probability of RV fratricide in a timed laydown and optionally adjusts launch times to reduce fratricide. |
| SOADA | Generates DGZs and allocates RVs to those DGZs so as to optimize user-specified damage objectives. |
| TARGET | Estimates missile performance capabilities, evaluates missile sortie feasibility, assesses a missile's depressed trajectory capability, and generates trajectory state vectors. |

**FIGURE 2-2 MPS CSC DESCRIPTIONS**

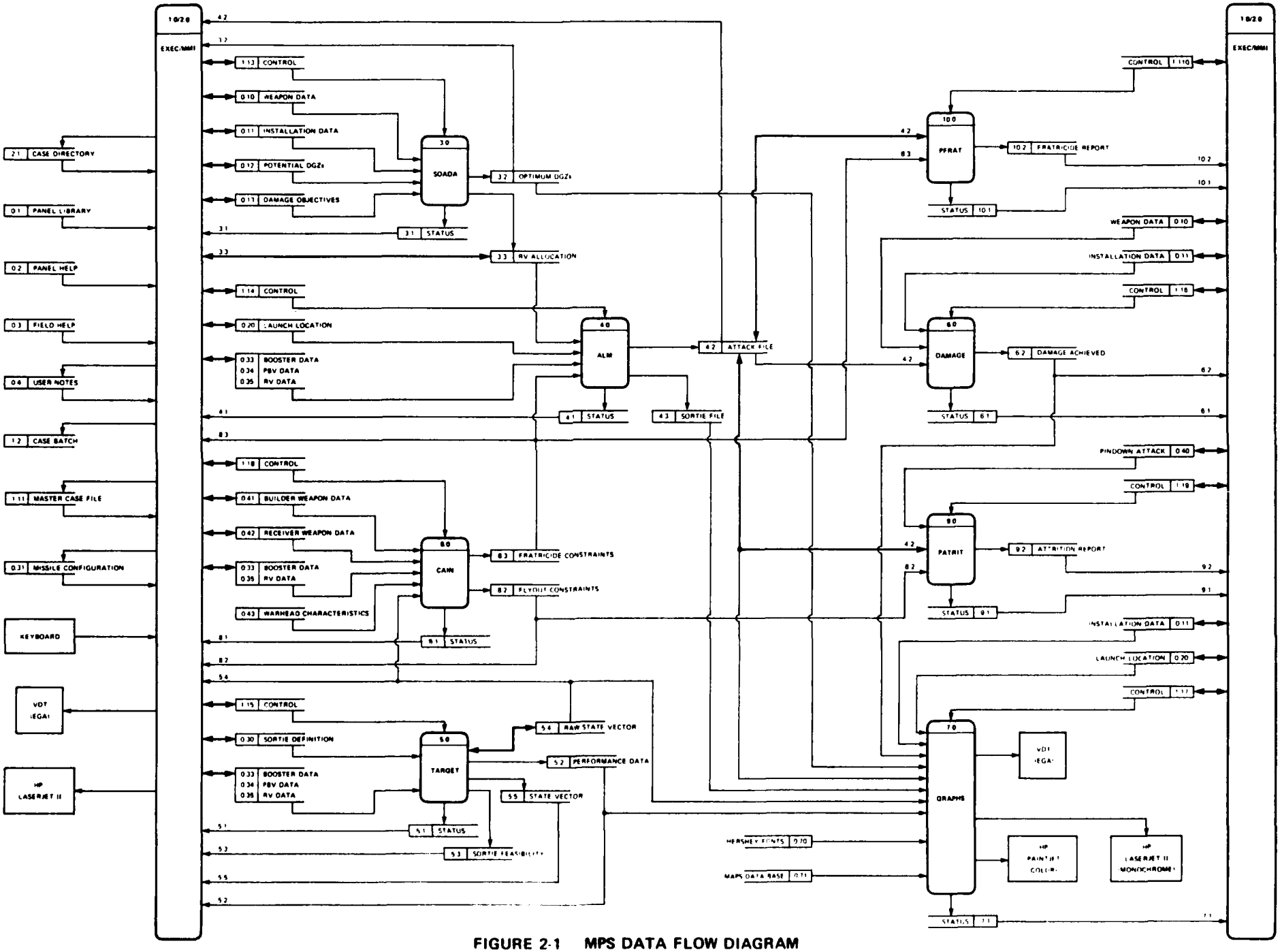



**FIGURE 2-1 MPS DATA FLOW DIAGRAM**

## 2.4 MPS CASES

Fundamental to the informed employment of the MPS is an understanding and appreciation of the concept of a case. The importance of this concept is reflected in the MPS main menu which, with only one notable exception (build data sets), provides the user with a set of operations (define, load, execute, graph, and review) for manipulating cases.

Within the MPS, a case refers collectively to: (1) a selected set of MPS functions; (2) the ordered sequence of CSC executions needed to perform those functions; (3) all files input to that execution sequence; and (4) all output files produced.

The MPS supports three types of cases:

- Accessibility - an accessibility case involves the autonomous execution of the TARGET CSC to generate missile performance estimates, to assess missile sortie feasibility, to assess a missile's depressed trajectory capability or to report trajectory state vectors.
- Nuclear Constraints - a nuclear constraints case involves the autonomous execution of the CAIN CSC to generate deterministic or probabilistic nuclear environment exclusion regions for RVs (fratricide) or missiles (flyout attrition).
- Laydown Case - a laydown case provides the user with the option of running subsets of the ALM, DAMAGE, PATRIT, PFRAT and SOADA CSCs in an end-to-end mode to plan, generate, and evaluate timed laydowns of ballistic missile forces on target sets, subject to operational and nuclear vulnerability constraints.

## 2.5 MPS OPERATIONAL ENVIRONMENT

The MPS is designed to execute efficiently on AT class microcomputers, running under an MS-DOS operating system. The baseline hardware configuration for which the MPS is tailored consists of the following elements:

- 80286 or 80386 central processing unit equipped with an 80287 or 80387 numeric coprocessor.
- EGA video display adapter and monitor.
- 640 kilobytes of random access memory (RAM).
- 1 megabyte of additional RAM for use as a graphics data buffer.
- Hewlett Packard Laserjet Series II printer or Hewlett Packard PaintJet printer.
- 20 megabytes of hard disk storage.
- One 1.2 megabyte 5¼" floppy disk drive.

Although the MPS is targeted for operation on this baseline hardware, the MPS is capable of being installed and executed on a wider class of machines.

To compile, link and execute the MPS, the following software is required:

- MS-DOS Operating System 3.2 or above. The system must include the VDISK.SYS and ANSI.SYS installable device drivers.
- Microsoft FORTRAN Optimizing Compiler, Version 4.1 or above.
- Microsoft Macro Assembler, Version 5.1.

As one might infer from the preceding list, the MPS is a combination of FORTRAN and assembly code, with over 95% of the software being FORTRAN. Use of assembly code is confined to the EXEC, MMI and GRAPHS CSCs. The majority of the MPS FORTRAN code conforms to the FORTRAN 77 (ANSI X3.9 - 1978) programming language standard.

However, there are a few general exceptions employed by the MPS to increase software efficiency and to improve software maintainability. These exceptions are:

- The use of the INCLUDE statement to include common blocks in subroutines.
- The use of '*' data typing, specifically INTEGER*2 and LOGICAL*1 data types to reduce the size of executable code.
- The use of the Microsoft backslash and $ edit descriptors to improve display quality.
- The use of Hollerith constants in data statements.
- The use of type conversion for character constants specified in data statements.

Specific occurrences of these exceptions are described in Appendix C for each module.

# SECTION 3
# CSC OVERVIEW

## 3.1 PURPOSE

Within the MPS, the ALM (Attack Laydown Model) CSC is responsible for assigning RVs (Reentry Vehicles) to DGZs (Desired Ground Zeros) and for scheduling missile launch times. Specifically, the ALM first forms missile sorties that:

- Collectively fulfill, or fulfill as closely as possible, an input RV allocation specifying the number of RVs to be targeted to each DGZ.
- Are individually feasible in the sense that the sortie is within the missile's performance capabilities.
- Individually satisfy user-specified targeting requirements and constraints, including: the reentry angle trajectory to be employed; the maximum amount of PBV (Post-Boost Vehicle) fuel to be used for range extension; and the minimum exoatmospheric distance between RVs.

Using the feasible missile sorties identified, the ALM next constructs a missile launch schedule that:

- Optimizes a user-supplied attack timing goal (either minimize RV downtime span, or minimize missile launch duration).
- Enforces minimum time separations between RVs targeted to the same DGZ, when those RVs are deployed from different missiles. These times are either specified by the user or read from the input Fratricide Constraints Files.
- Satisfies missile system launch rate constraints.

The ALM CSC can be executed without previously executing any other CSCs, but this requires the user to allocate RVs to DGZs, to generate the RV Allocation file normally written by the Allocation CSC (SOADA), and to specify minimum in-line RV arrival time separations. In most applications, however, the Allocation and CAIN CSCs are run prior to executing ALM. CAIN is run in mode 2, to generate the Fratricide Constraints Files that can be input to ALM.

## 3.2 OPTIONS

Operation of the ALM CSC is controlled by user specification of the following six options:

- In-line RV time separations. These times can be provided by the user or the user can direct the ALM to read

these times for user-specified Fratricide Constraints Files. If the user elects the input option, then the user inputs the desired minimum in-line RV spacing ($\Delta T$) as a function of RV number. For example, the user might specify a $\Delta T$ of 2 seconds between the first and second RVs targeted to a DGZ, and a $\Delta T$ of 2 minutes between the second and third RVs. This single set of $\Delta T$s is then employed for all DGZs, regardless of the height of burst (HOB) associated with a DGZ. Alternatively, the user can supply the name of a CAIN-produced Fratricide Constraints File for each HOB employed in an attack, and direct the ALM to read the minimum, fratricide-free, in-line RV time spacings ($\Delta TF$'s) contained in those files. Based on the HOB employed at a DGZ, the ALM will then: (1) identify the applicable $\Delta TF$ for that DGZ; and (2) enforce that $\Delta TF$ uniformly between every consecutive pair of RVs targeted to that DGZ. $\Delta TF$ is defined as the minimum, intra-DGZ RV arrival time separation that ensures avoidance of RV fratricide due to the prompt nuclear environments (thermal and nuclear radiation environments and blast wave environments) that are produced by an RV HOB detonation.

- Maximum fraction of PBV fuel to be used for PBV range extension.
- Reentry angle to be employed for each missile participating in an attack. If zero is entered, minimum-energy trajectories are to be used.
- Rotating or non-rotating earth.
- Attack timing goal. Either minimize RV downtime span, or minimize missile launch duration.
- Space or don't space RVs a minimum distance apart exoatmospherically.

## 3.3 CONCEPTUAL LIMITATIONS

The conceptual limitations of the ALM CSC are as follows:

- All missiles employed in the attack are identical.
- The number of RVs in an attack must be evenly divisible by the number of RVs on a missile; i.e., an integer number of missiles must be employed in an attack.
- A single HOB must be used for all RVs targeted to a given DGZ.

## 3.4 DATA FLOW DIAGRAM

The data flow diagram (DFD) presented in Figure 3-1 identifies the files that are input to and output from the ALM CSC. Furthermore, this figure indicates which CSCs can generate data for, or use data generated by, the ALM CSC. A brief discussion of DFDs is provided in Appendix H.

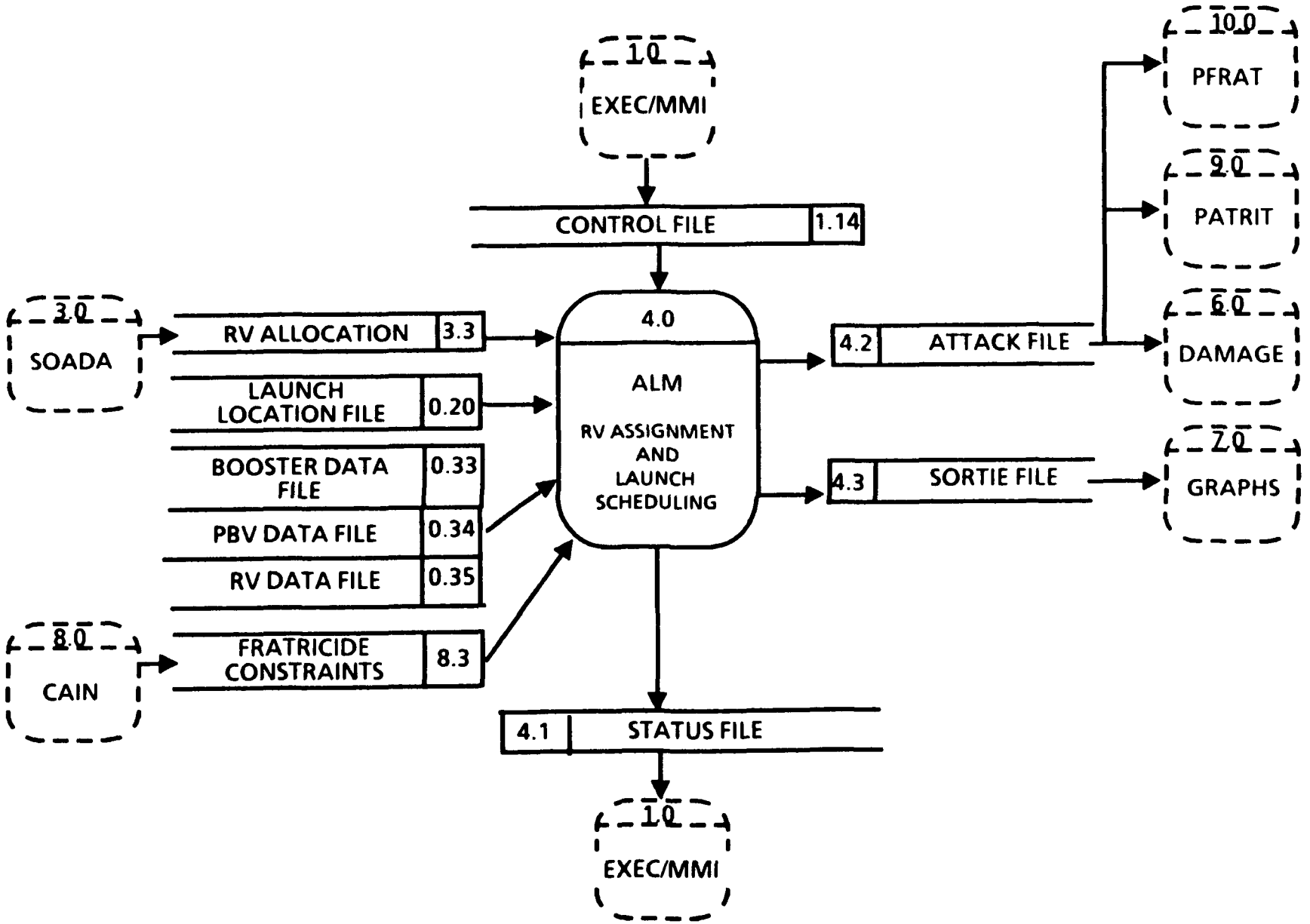



**FIGURE 3-1 ALM DATA FLOW DIAGRAM**

Inspection of Figure 3-1 reveals that the ALM accepts as input the following seven files:

- Control File (1.14). Prepared by the EXEC/MMI, this file contains values that control the performance of the ALM CSC.
- RV Allocation File (3.3). This file specifies the number of DGZs, their locations, and the number of RVs allocated to each DGZ. This file is normally written by the SOADA CSC.
- Launch Location File (0.20). This file contains the latitude and longitude of each attacking missile. The file is generated by the user via the MMI.
- Booster Data File (0.33). Supplies booster model parameters for the missile system employed in the attack.
- PBV Data File (0.34). Supplies PBV model parameters for the attacking missile system, if that system has a PBV.
- RV Data File (0.35). Supplies RV model parameters for the attacking missile system.
- Fratricide Constraints Files (8.3). These files are generated by the CAIN CSC and contain the fratricide-free, in-line RV time separations used by the ALM.

The following files are generated by the ALM CSC:

- Status File (4.1). This file informs the EXEC/MMI and the user of the success or failure of the ALM execution. Information, warning and error messages describe the cause of any failure.
- Attack File (4.2). This file contains missile launch time and RV downtime statistics, as well as individual missile and RV data. ALM initializes the probability of attrition, and the probabilities of fratricide for each RV. The Attack File can be passed to the DAMAGE, PATRIT, and PFRAT CSCs.
- Sortie File (4.3). This file describes the sorties performed by each missile; i.e., it provides the ordered list of DGZs targeted by each missile.

### 3.5 FUNCTION CHART

Figure 3-2 illustrates the functional structure of the ALM CSC. This figure presents the four-step process that ALM employs to assign RVs to DGZs and schedule missile launches. These steps are:

- Read input files. ALM reads the Control, RV Allocation, Launch Location, Booster Data, PBV Data, RV Data, and (optionally) Fratricide Constraints files.
- Form valid sorties. For each missile, ALM identifies an ordered sequence of DGZs that can be targeted. Collectively, these

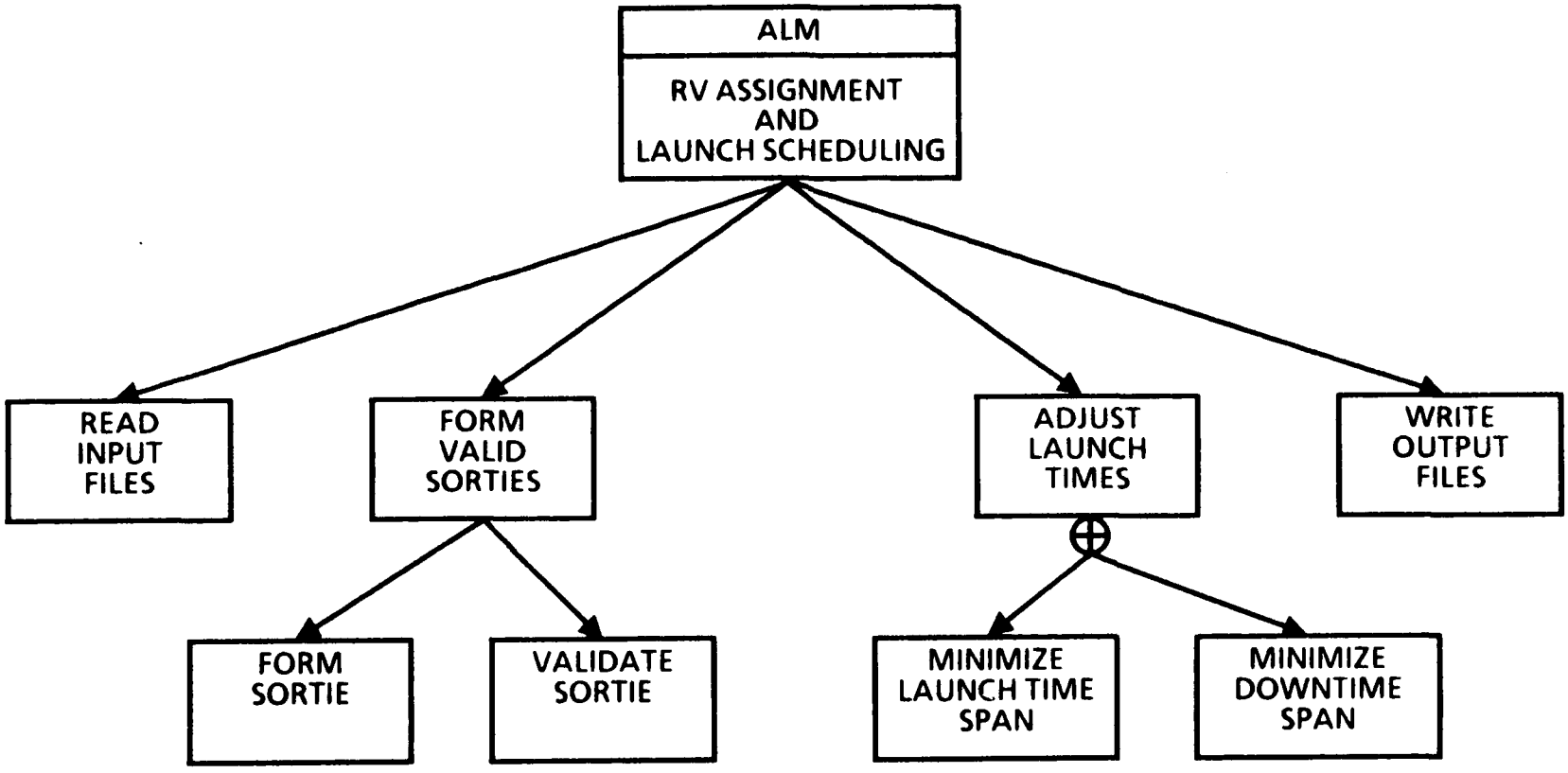



**FIGURE 3-2 ALM FUNCTION CHART**

sorties are designed to optimally fulfill the input RV allocation.

- Adjust launch times. For each missile, ALM identifies a launch time. The resultant launch schedule is designed to optimize the attack timing objective, and satisfy weapon system and targeting constraints.
- Write output files. ALM writes the Attack and Sortie Files.

See Appendix I for a brief tutorial on function charts, if needed.

# SECTION 4
# INPUTS

All of the input data required to execute ALM are provided via a set of data files. These files are either: output files generated by a prior execution of another CSC, user-modified versions of such output files, output files produced via the MPS "BUILD DATA SETS" option, or user-generated (via the MMI) input files. Figure 4-1 identifies the seven files input to ALM and indicates the source of those files.

The first three files listed in Figure 4-1 can all be created by the user through the MMI, but the RV Allocation File will normally be written by SOADA, the RV Allocation CSC.

The next three files (Booster, PBV and RV Data) are Missile Data Base files. These files must be created by the user prior to executing the ALM CSC. More specifically, the user must have created a record in each of these files corresponding to the missile system to be used in the ALM execution. These files/records are created by the user via the MMI when the "BUILD DATA SETS" option is selected in the MPS main menu.

The last file, the Fratricide Constraints File, is always written by the CAIN CSC; it is needed only if the user doesn't want to specify the time between layers.

In order to amend the set of input data to the ALM CSC, the maintenance programmer must change one or more of

| FILE | FILE NO. | CREATED BY |
|---|---|---|
| Control File | 1.14 | User via MMI |
| RV Allocation File | 3.3 | SOADA CSC (or user via MMI) |
| Launch Location File | 0.20 | User via MMI |
| Booster Data File | 0.33 | Missile Data Base |
| PBV Data File | 0.34 | Missile Data Base |
| RV Data File | 0.35 | Missile Data Base |
| Fratricide Constraints File | 8.3 | CAIN CSC (optional) |

**FIGURE 4-1 ORIGINS OF ALM INPUT FILES**

these input files. To assist him in this endeavor, Figures 4-2 through 4-8 are provided. These figures, called Input-Output Parameter Tables (IOPTs), show the detailed format and content of the seven ALM input files.

FILE NAME: ALM Control File ('case'.CF4)

FILE NUMBER: 1.14

DATE: 06/29/89

FILE TYPE

☒ FORMATTED ☒ SEQUENTIAL
☐ UNFORMATTED ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | 1 | Case name ( = 'case') | A8 | | | |
| | 2 | Case description | 2X, A40 | | | |
| 2 | 1 | Name of RV Allocation File; i.e. 'case'.RVA (file #3.3). Includes drive and path | A60 | | | |
| 3 | 1 | Name of Launch Location File; i.e., 'case'.LLF (file #0.20) | A60<br>A20 | | | |
| 4 | 1 | Booster file record number | I4 | | | |
| | 2 | Name of Missile Data Set Booster File (MPS.BST, file #0.33) | 2X,A60 | | | |
| 5 | 1 | PBV file record number | I4 | | | |
| | 2 | Name of Missile Data Set PBV File (MPS.PBV, file #0.34) | 2X,A60 | | | |
| 6 | 1 | RV file record number | I4 | | | |
| | 2 | Name of Missile Data Set RV File (MPS.RV, file #0.35) | 2X,A60 | | | |
| 7 | 1 | Number of CAIN fratricide models to use | I4 | [0, 1] | | |
| | 2 | Drive and path for file(s) containing fratricide model(s) | 2X, A60 | | | |
| 8 | 1 | Name of first fratricide model file | 2X, A12 | | | |
| | 2 | Name of 2nd fratricide model file | 2X, A12 | | | |
| | 3 | Name of 3rd fratricide model file | 2X, A12 | | | |

FIGURE 4-2 ALM CONTROL FILE IOPT

FILE NAME: ALM Control File ('case'.CF4)

FILE NUMBER: 1.14

DATE: 06/29/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 8 (cont'd) | 4 | Name of 4th fratricide model file | 2X, A11 | | | |
| 9 | 1 | Name of Attack File; i.e. 'case'. AF; (file #4.2). Includes drive and path. | A60 | | | |
| 10 | 1 | Name of Sortie File; i.e. 'case'.SF (file #4.3). Includes drive and path | A60 | | | |
| 11 | | RUN OPTIONS: | | | | |
| | 1 | FRF4RX. Fraction of fuel for PBV range extension. | F6.3 | [0.000, 1.000] | - | - |
| | 2 | GAMMA. Reentry angle (> 0) or value indicating minimum energy reentry angle ( = 0) should be used. | F6.2 | [0.00, + 50.00] | deg | deg |
| | 3 | ROTRTH. Flag: if .TRUE., rotating earth should be used. | L4 | [T, F] | - | - |
| | 4 | SYNCRO. Flag: if .TRUE., arrival times of first layer RVs should be synchronized. | L4 | [T, F] | - | - |
| | 5 | SPACED. Flag: if.TRUE., RVs must be spaced at least a minimum distance apart. The minimum distance depends on the weapon type. | L4 | [T, F] | - | - |

**FIGURE 4-2 ALM CONTROL FILE IOPT (Continued)**

FILE NAME: ALM Control File ('case'.CF4)

FILE NUMBER: 1.14

DATE: 06/29/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 12 | | TIMES BETWEEN LAYERS | | | | |
| | 1 | Time between layer 1 and layer 2. | F6.1 | [0.0, 999.0] | secs | secs |
| | 2 | Time between layer 2 and layer 3. | F6.1 | [0.0 999.0] | secs | secs |
| | 3 | Time between layer 3 and layer 4. | F6.1 | [0.0, 999.0] | secs | secs |
| | 4 | Time between layer 4 and layer 5. | F6.1 | [0.0, 999.0] | secs | secs |
| | 5 | Time between layer 5 and layer 6. | F6.1 | [0.0, 999.0] | secs | secs |
| | 6 | Time between layer 6 and layer 7. | F6.1 | [0.0, 999.0] | secs | secs |
| | 7 | Time between layer 7 and layer 8. | F6.1 | [0.0, 999.0] | secs | secs |
| | 8 | Time between layer 8 and layer 9. | F6.1 | [0.0, 999.0] | secs | secs |
| | 9 | Time between layer 9 and layer 10. | F6.1 | [0.0, 999.0] | secs | secs |

**FIGURE 4-2 ALM CONTROL FILE IOPT (Concluded))**

FILE NAME: RV Allocation File ('case'.RVA)

FILE NUMBER: 3.3

DATE: 6/19/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | DEFAULT | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|---|
| | | | | | | ENG. | MET. |
| 1 | 1 | Target island label | - | 1x, A24 | - | - | - |
| 2 | 1 | Target island elevation | 0 | 1x, A40, F8.0 | [-10000, 30000] ft | ft | m |
| 3 | 1 | Height of burst 1 | 0 | 1x, A40, F8.1 | [0, 32000] ft | ft | m |
| 4 | 1 | Height of burst 2 | 0 | 1x, A40, F8.1 | [0, 32000] ft | ft | m |
| 5 | 1 | Number of missiles in attack | 0 | 1x, A40, I4 | [1, 10] | - | - |
| 6 | 1 | Number of RVs allocated | - | 1x, A40, I4 | [1, 250] | - | - |
| 7-9 | 1 | RV allocation table labels | | A80 | - | - | - |

**FIGURE 4-3 RV ALLOCATION FILE IOPT**

FILE NAME: RV Allocation File ('case'.RVA)

FILE NUMBER: 3.3

DATE: 6/19/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | DEFAULT | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|---|
| | | | | | | ENG. | MET. |
| 10-259 | | Allocation record | | | | | |
| | 1 | Index descriptor | 0 | 1x, I4 | [1, 9999] | - | - |
| | 2 | DGZ type<br>0 = installation<br>1 = grid point<br>2 = user input | 2 | 1x, I2 | [0, 2] | - | - |
| | 3 | Latitude (°N) decimal degrees | 0 | 5x, F8.3 | [-90, 90] | deg | deg |
| | 4 | Longitude (°E) decimal degrees | 0 | 4x, F8.3 | [-180, 180] | deg | deg |
| | 5 | Height of burst index | 1 | 8x, I1 | [1, 2] | - | - |
| | 6 | Number of RVs allocated to DGZ<br>(Warning message will appear if exceeds 10, ALM should check this)<br><br>(Order implies values; i.e. record 10 describes the most valuable DGZ, record 259, the least valuable) | 0 | 11x, I3 | [1, 250] | - | - |

**FIGURE 4-3 RV ALLOCATION FILE IOPT (Concluded)**

FILE NAME: Launch Location File ('case'.LLF)

FILE NUMBER: 0.20

DATE: 06/20/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS

| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | - | LAUNCH COMPLEX RECORD | - | | - | - |
| | 1 | Launch Complex Label; e.g. "Nureyev, USSR" | A24 | | | |
| | 2 | Number of missile silos | I4 | [1, 10] | - | - |
| | 3 | Co-located silos | L4 | [T, F] | - | - |
| 2 | - | MISSILE RECORD If Record 1, Parameter 3 is TRUE, Records 2-11 contain the following: | | | | |
| | 1 | Latitude (°N) | F8.3 | [-90.000, + 90.000] | deg | deg |
| | 2 | Longitude (°E) | F8.3 | [-180.000, + 180.000] | deg | deg |
| | 3 | Altitude | F8.0 | [-10000.0, + 30000.0] | ft | m |
| 2-11 | - | MISSILE RECORD If Record 1, Parameter 3 is FALSE, Record 2 contains the following: | | | | |
| | 1 | Missile Number (Index) | I4 | [1, 10] | | |
| | 2 | Latitude (°N) | F8.3 | [-90.000, + 90.000] | deg | deg |
| | 3 | Longitude (°E) | F8.3 | [-180.000, + 180.000] | deg | deg |
| | 4 | Altitude | F8.0 | [-10000.0, + 30000.0] | ft | m |
| | 5 | Flight (i.e. group missile belongs to): a letter | 1X, A1 | [A, Z] | - | - |

**FIGURE 4-4 LAUNCH LOCATION FILE IOPT**

FILE NAME: Booster Data File (MPS.BST)

FILE NUMBER: 0.33

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | 1 | Last Record | I4 | [1, 31] | - | - |
| | 2 | Number of open records | I4 | [0, 30] | - | - |
| | 3 | Array of open record numbers size = 30 | I4 | [2, 31] | - | - |
| 2-31 | 1 | Booster name | A20 | | | |
| | 2 | Number of stages | I4 | [1, 4] | - | - |
| | 3 | Energy management stage | I4 | [1, 4] | - | - |
| | 4 | Number of drag coefficients | I4 | [0, 10] | - | - |
| | 5 | Number of PBV's on booster | I4 | [0, 1] | - | - |
| | 6 | Number of RV's on booster | I4 | [1, 25] | - | - |
| | 7 | Flag:LSOLID - .TRUE. if solid | I4 | [T, F] | - | - |



FIGURE 4-5 BOOSTER DATA FILE IOPT

FILE NAME: Booster Data File (MPS.BST)

FILE NUMBER: 0.33

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 8 | Weight - stage 1 | R4 | (0.0, 1.0E + 06) | lb | kg |
| | 9 | Weight - stage 2 | R4 | [0.0, 1.0E + 06) | lb | kg |
| | 10 | Weight - stage 3 | R4 | [0.0, 1.0E + 06) | lb | kg |
| | 11 | Weight - stage 4 | R4 | [0.0, 1.0E + 06) | lb | kg |
| | 12 | Vacuum specific impulse - stage 1 | R4 | (0.0, 1000.0) | sec | N-s/kg |
| | 13 | Vacuum specific impulse - stage 2 | R4 | [0.0, 1000.0) | sec | N-s/kg |
| | 14 | Vacuum specific impulse - stage 3 | R4 | [0.0, 1000.0) | sec | N-s/kg |
| | 15 | Vacuum specific impulse - stage 4 | R4 | [0.0, 1000.0) | sec | N-s/kg |
| | 16 | Vacuum thrust - stage 1 | R4 | (0.0, 2.25E + 06) | lb | kN |
| | 17 | Vacuum thrust - stage 2 | R4 | [0.0, 2.25E + 06) | lb | kN |



**FIGURE 4-5 BOOSTER DATA FILE IOPT (Continued)**

FILE NAME: Booster Data File (MPS.BST)

FILE NUMBER: 0.33

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 18 | Vacuum thrust - stage 3 | R4 | [0.0, 2.25E + 06) | lb | kN |
| | 19 | Vacuum thrust - stage 4 | R4 | [0.0, 2.25E + 06) | lb | kN |
| | 20 | Base area - stage 1 | R4 | (0.0, 1000.0) | $ft^2$ | $m^2$ |
| | 21 | Base area - stage 2 | R4 | [0.0, 100.0) | $ft^2$ | $m^2$ |
| | 22 | Base area - stage 3 | R4 | [0.0, 100.0) | $ft^2$ | $m^2$ |
| | 23 | Base area - stage 4 | R4 | [0.0, 100.0) | $ft^2$ | $m^2$ |
| | 24 | Exit area - stage 1 | R4 | (0.0, 1000.0] | $ft^2$ | $m^2$ |
| | 25 | Exit area - stage 2 | R4 | [0.0, 1000.0] | $ft^2$ | $m^2$ |
| | 26 | Exit area - stage 3 | R4 | [0.0, 1000.0] | $ft^2$ | $m^2$ |
| | 27 | Exit area - stage 4 | R4 | [0.0, 1000.0] | $ft^2$ | $m^2$ |
| | 28 | Maximum burntime - stage 1 | R4 | (0.0, 1000.0) | sec | sec |
| | 29 | Maximum burntime - stage 2 | R4 | [0.0, 1000.0) | sec | sec |
| | 30 | Maximum burntime - stage 3 | R4 | [0.0, 1000.0) | sec | sec |
| | 31 | Maximum burntime - stage 4 | R4 | [0.0, 1000.0) | sec | sec |
| | 32 | Coast duration - stage 1 | R4 | (0.0, 100.0) | sec | sec |
| | 33 | Coast duration - stage 2 | R4 | [0.0, 100.0) | sec | sec |

**FIGURE 4-5 BOOSTER DATA FILE IOPT (Continued)**

FILE NAME: Booster Data File (MPS.BST)

FILE NUMBER: 0.33

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 34 | Coast duration - stage 3 | R4 | [0.0, 100.0) | sec | sec |
| | 35 | Coast duration - stage 4 | R4 | [0.0, 100.0) | sec | sec |
| | 36 | Minimum burntime of energy management stage | R4 | [0.0, 1000.0) | sec | sec |
| | 37 | Vertical launch duration | R4 | [0.0, 100.0) | sec | sec |
| | 38 | Time of flight to initiate attitude hold | R4 | [0.0, 1000.0) | sec | sec |
| | 39 | Shroud jettison time | R4 | [0.0, 1000.0) | sec | sec |
| | 40 | Shroud weight | R4 | [0.0, 2.25E + 04) | lb | kg |
| | 41 | Launch sabot velocity | R4 | [0.0, 1000.0) | ft/s | m/s |
| | 42 | Boost phase acceleration limit | R4 | (0.0, 100.0) | g | g |
| | 43 | CD (1) | R4 | [0.0, 1.0] | - | - |
| | 44 | CD (2) | R4 | [0.0, 1.0] | - | - |
| | 45 | CD (3) | R4 | [0.0, 1.0] | - | - |
| | 46 | CD (4) | R4 | [0.0, 1.0] | - | - |
| | 47 | CD (5) | R4 | [0.0, 1.0] | - | - |
| | 48 | CD (6) | R4 | [0.0, 1.0] | - | - |
| | 49 | CD (7) | R4 | [0.0, 1.0] | - | - |
| | 50 | CD (8) | R4 | [0.0, 1.0] | - | - |
| | 51 | CD (9) | R4 | [0.0, 1.0] | - | - |
| | 52 | CD (10) | R4 | [0.0, 1.0] | - | - |



**FIGURE 4-5 BOOSTER DATA FILE IOPT (Continued)**

FILE NAME: Booster Data File (MPS.BST)

FILE NUMBER: 0.33

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 53 | Mach number (1) | R4 | [0.0, 100.0] | - | - |
| | 54 | Mach number (2) | R4 | [0.0, 100.0] | - | - |
| | 55 | Mach number (3) | R4 | [0.0, 100.0] | - | - |
| | 56 | Mach number (4) | R4 | [0.0, 100.0] | - | - |
| | 57 | Mach number (5) | R4 | [0.0, 100.0] | - | - |
| | 58 | Mach number (6) | R4 | [0.0, 100.0] | - | - |
| | 59 | Mach number (7) | R4 | [0.0, 100.0] | - | - |
| | 60 | Mach number (8) | R4 | [0.0, 100.0] | - | - |
| | 61 | Mach number (9) | R4 | [0.0, 100.0] | - | - |
| | 62 | Mach number (10) | R4 | [0.0, 100.0] | - | - |
| | 63 | Launch reliability | R4 | [0.0, 1.0] | - | - |
| | 64 | Missile/PBV reliability | R4 | [0.0, 1.0] | - | - |
| | 65 | Minimum time between launches | R4 | [0.0, 6000.0] | sec | sec |

**FIGURE 4-5 BOOSTER DATA FILE IOPT (Concluded)**

FILE NAME: PBV Data File (MPS.PBV)

FILE NUMBER: 0.34

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | 1 | Last Record | I4 | [1, 31] | - | - |
| | 2 | Number of open records | I4 | [0, 30] | - | - |
| | 3 | Array of open record numbers size = 30 | I4 | [2, 31] | - | - |
| 2-31 | 1 | PBV name | A20 | | | |
| | 2 | Flag: LPULL - .TRUE. if puller<br>.FALSE. if pusher | I4 | [T, F] | - | - |
| | 3 | Flag: LGUIDE - .TRUE. if extended stellar guidance after burnout | I4 | [T, F] | - | - |
| | 4 | Flag: LINVRT - .TRUE. if PBV is mounted nose backwards | I4 | [T, F] | - | - |
| | 5 | Flag: LCONT - .TRUE. if main engine burns continuously<br>.FALSE. if main engine can be turned-off | I4 | [T, F] | - | - |
| | 6 | Number of RV's on PBV (NOBJs) | I4 | [1, 25] | - | - |
| | 7 | Number of RV's on tier | I4 | [0, 25] | - | - |
| | 8 | Spacing direction flag:<br>1 - space later in time<br>-1 - space earlier in time | I4 | 1 or -1 | - | - |
| | 9 | Object type indicator (1st object)<br>1 - RV<br>0 - No object | I4 | 0 or 1 | - | - |

**FIGURE 4-6 PBV DATA FILE IOPT**

FILE NAME: PBV Data File (MPS.PBV)

FILE NUMBER: 0.34

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 10 to 33 | Object type indicator (2nd object) to Object type indicator (25th object) | I4 | 0 or 1 to 0 or 1 | | |
| | 34 | PBV structure (dry) weight | R4 | (0.0, 1.0E + 06) | lb | kg |
| | 35 | PBV fuel weight | R4 | (0.0, 1.0E + 05) | lb | kg |
| | 36 | Axial engine thrust (low if dual levels) | R4 | (0.0, 2.25E + 04) | lb | kN |
| | 37 | Specific impulse | R4 | (0.0, 1000.0) | sec | N·S/k |
| | 38 | Average attitude maneuver duration | R4 | (0.0, 60.0) | sec | sec |
| | 39 | Average attitude maneuver flowrate | R4 | (0.0, 100.0) | lb/sec | kg/sec |
| | 40 | Predeployment maneuver duration | R4 | (0.0, 60.0) | sec | sec |

**FIGURE 4-6 PBV DATA FILE IOPT (Continued)**

FILE NAME: PBV Data File (MPS.PBV)

FILE NUMBER: 0.34

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 41 | Post deployment maneuver duration | R4 | (0.0, 60.0) | sec | sec |
| | 42 | Dual level thrust ratio | R4 | [1.0, 100.0] | - | - |
| | 43 | High thrust turn-on criterion | R4 | (0.0, 1000.0) | sec | sec |
| | 44 | PBV tier structure weight | R4 | [0.0, 1000.0] | lb | kg |
| | 45 | High-to-low/low-to-high thrust transition time | R4 | [0.0, 100.0] | sec | sec |
| | 46 | Post booster/PBV separation coast | R4 | [0.0, 1000.0] | sec | sec |
| | 47 | Post Booster/PBV separation burn | R4 | [0.0, 1000.0] | sec | sec |
| | 48 to 72 | Minimum inter-object spacing (1st obj.)<br>Minimum inter-object spacing (25th obj.) | R4 | [0.0, 1.0E + 06] | ft | m |



**FIGURE 4-6 PBV DATA FILE IOPT (Concluded)**

FILE NAME: RV Data File (MPS.RV)

FILE NUMBER: 0.35

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | 1 | Last record | I4 | [1, 31] | - | - |
| | 2 | Number of open records | I4 | [0, 30] | - | - |
| | 3 | Array of open record numbers size = 30 | I4 | [2, 31] | - | - |
| 2-31 | 1 | RV name | A20 | | | |
| | 2 | Number of SHOB | I4 | [0, 5] | - | - |
| | 3 | Number of drag values | I4 | [0, 10] | - | - |
| | 4 | *Number of MaRV AA vs CL values* | I4 | [0, 20] | - | - |
| | 5 | Ballistic coefficient | R4 | (0.0, 2.0E + 04) | $lb/ft^2$ | $kg/m^2$ |
| | 6 | Weight | R4 | (0.0, 1.0E + 04) | lb | kg |
| | 7 | Base Area | R4 | (0.0, 1000.0) | $ft^2$ | $m^2$ |
| | 8 | Yield | R4 | (0.0, 30000] | kt | kt |
| | 9 | CEP | R4 | [0.0, 5000.0] | ft | m |
| | 10 | CD (1) | R4 | [0.0, 1.0] | - | - |
| | . | | | | | |
| | . | | | | | |
| | . | | | | | |
| | 19 | CD (10) | | | | |

**FIGURE 4-7 RV DATA FILE IOPT**

FILE NAME: RV Data File (MPS.RV)

FILE NUMBER: 0.35

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 20 | Mach (1) | R4 | [0.0, 100.0] | | |
| | . | . | | | | |
| | . | . | | | | |
| | . | . | | | | |
| | 29 | Mach (10) | | | | |
| | 30 | RV reliability | R4 | [0.0, 1.0] | | |
| | 31 | SHOB (1) | R4 | [0.0, 1000.0] | $ft/kT^{1/3}$ | $m/kT^{1/3}$ |
| | . | . | | | | |
| | . | . | | | | |
| | . | . | | | | |
| | 35 | SHOB (5) | | | | |
| | 36 | Nose radius | R4 | (0.0, 100.0) | in | cm |
| | 37 | Base radius | R4 | (0.0, 100.0) | in | cm |
| | 38 | Cone half-angle | R4 | (0.0, 90.0) | deg | deg |
| | 39 | Surface roughness | R4 | (0.0, 1.0) | in | cm |
| | 40 | MaRV hypersonic drag coefficient | R4 | (0.0, 1.0) | - | - |
| | 41 | MaRV maximum lift-to-drag ratio | R4 | (0.0., 1.0 E + 04) | - | - |
| | 42 | MaRV maximum attack angle | R4 | (0.0, 90.0) | deg | deg |
| | 43 | MaRV minimum attack angle | R4 | (0.0, 90.0) | deg | deg |

**FIGURE 4-7 RV DATA FILE IOPT (Continued)**

FILE NAME: RV Data File (MPS.RV)

FILE NUMBER: 0.35

DATE: 10/13/89

FILE TYPE

☐ FORMATTED ☐ SEQUENTIAL

☒ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 2-31 | 44 | 1st entry in attack angle table | R4 | (0.0, 90.0) | deg | deg |
| | . | | | | | |
| | . | | | | | |
| | . | | | | | |
| | 63 | 20th entry in attack angle table | | | | |
| | 64 | 1st entry in CL table | R4 | [0.0, 1.0] | - | - |
| | . | | | | | |
| | . | | | | | |
| | . | | | | | |
| | 83 | 20th entry in CL table | R4 | | | |

**FIGURE 4-7 RV DATA FILE IOPT (Concluded)**

FILE NAME: Fratricide Constraints File ('case'.FCF)

FILE NUMBER: 8.3

DATE: 12/8/89

FILE TYPE

☒ FORMATTED ☐ SEQUENTIAL

☐ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | 1 | Title | A50 | | | |
| 2 | 1 | Mode of operation<br>1 deterministic<br>2 fast conservative<br>3 probabilistic | I2 | | | |
| | 2 | Flyout or fratricide flag<br>2 flyout<br>4 fratricide (should always be 4) | I2 | | | |
| 3 | 1 | $SHOB_1$ - builder scaled height of burst | E10.4 | | $kft/(kT)^{\frac{1}{3}}$ | $km/(kT)^{\frac{1}{3}}$ |
| | 2 | $SHOB_2$ - receiver scaled height of burst | E10.4 | | $kft/(kT)^{\frac{1}{3}}$ | $km/(kT)^{\frac{1}{3}}$ |
| | 3 | Yield | E10.4 | | kT | kT |
| 4 | | | | | | |
| | 1 | $\sigma_{TOT}$ for builder | E10.4 | | sec | sec |
| | 2 | dummy variable | E10.4 | | | |
| | 3 | CEP for builder | E10.4 | | kft | km |
| | 4 | ALLOB - allowable fratricide fraction | E10.4 | | | |
| 5 | 1 | builder launch complex reliability | E10.4 | | | |
| | 2 | builder missile/RV reliability | E10.4 | | | |
| | 3 | builder RV/Warhead reliability | E10.4 | | | |
| | | (parameters 4, 5, & 6 are from the MDS and are passed to PFRAT & PATRIT, CAIN does not use them) | | | | |

**FIGURE 4-8 FRATRICIDE CONSTRAINTS FILE IOPT**

FILE NAME: Fratricide Constraints File ('case'.FCF)

FILE NUMBER: 8.3

DATE: 12/8/89

FILE TYPE

☒ FORMATTED ☐ SEQUENTIAL

☐ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 6 | 1 | NA - number of angles. (The angles are then calculated as: 180° • (I-1)/(NA-1) for the Ith angle) | I5 | | | |
| | 2 | NT - number of times | I5 | | | |
| | 3 | Reserved for future use | | | | |
| | 4 | D2GMAX, for all times ≧ TOFF (NT), all receiver with a greater target range will be safe - at any angle | E10.4 | | kft | km |
| | 5 | SSTIM1, a receiver aimed at the same DGZ as the builder will be safe at a time > SSTIM1 | E10.4 | | sec | sec |
| | 6 | SSTIM2 a receiver is safe at a time > SSTIM2 | E10.4 | | sec | sec |
| 7 | 1 | Sure safe up-range distance, a receiver farther up-range than this target value is safe for all times ≦ TOFF (NT) | E10.4 | | kft | km |
| | 2 | Sure safe down-range distance, a receiver farther down range than this target value is safe for all times ≦ TOFF (NT) | E10.4 | | kft | km |
| | 3 | Sure safe cross-range distance, a receiver at a greater cross-range distance is safe for all times ≦ TOFF (NT) | E10.4 | | kft | km |

**FIGURE 4-8 FRATRICIDE CONSTRAINTS FILE IOPT (Continued)**

FILE NAME: Fratricide Constraints File ('case'.FCF)

FILE NUMBER: 8.3

DATE: 12/8/89

FILE TYPE

☒ FORMATTED ☐ SEQUENTIAL

☐ UNFORMATTED ☒ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| | | [(repeat the following cards for each time (IT = 1, NT)] | | | | |
| 8 + (IT-1) * (NA + 1) | 1 | TOFF (IT) - time since intended burst time | E10.4 | | sec | sec |
| | 2 | XCNTR (IT) - up/down-range center of footprint at this time | E10.4 | | kft | km |
| | 3 | D1MIN (IT) - at all target ranges ≤ D1MIN at TOFF, the receiver will be killed | E10.4 | | kft | km |
| | 4 | D2MAX (IT) - at all target ranges > D2MAX at TOFF, the receiver will be safe | E10.4 | | kft | km |
| | | [repeat the following card for each angle (IA = 1, NA)] | | | | |
| 8 + (IT-1) * (NA + 1) + IA | 1 | $D_1$ - target range, above which survival is possible | E10.4 | | kft | km |
| | 2 | $D_2$ - target range above which survival is certain | E10.4 | | kft | km |
| | 3 | A - radius of exclusion zone | E10.4 | | kft | km |
| | 4 | B - diffusion parameter for density fall-off | E10.4 | | kft | km |
| | 5 | C - strength parameter | E10.4 | | | |
| | | together these variables provide a fit to data to determine the probability of kill using a Wood-Saxon form: $p(r) = \begin{cases} 1.0 & 0 \leq r \leq D_1 \\ \frac{C}{1 + \exp((r-A)/B)} & D_1 < r \leq D_2 \\ 0.0 & D_2 < r \end{cases}$ | | | | |

**FIGURE 4-8 FRATRICIDE CONSTRAINTS FILE (Concluded)**

# SECTION 5
# PROCESSING

In this section, a comprehensive discussion is provided describing how the ALM CSC accomplishes its function. Specifically, a detailed overview of the processing logic, implemented in ALM, is provided using flowcharts as the presentation medium. Following that, a brief tutorial on the underlying concepts and models of the ALM CSC is presented. Finally, an identification and explanation of any program limitations is reported. For a discussion of flowchart symbology, see Appendix J.

## 5.1 PROCESSING FLOW

The process used to assign RVs to DGZs and to schedule missile launches is shown in Figures 5-1 through 5-12 using top-level, functional flowcharts. However, before proceeding to a discussion of these flowcharts a brief overview of the ALM's methodology is needed.

The ALM employs a heuristic, decision-making algorithm to identify feasible missile sorties that fulfill (or nearly fulfill) the input RV allocation and that satisfy user-prescribed targeting requirements and constraints. This algorithm repeatedly sweeps across the target set from right to left forming accessible missile sorties until all missiles have been assigned. Each sortie is initially constructed so that it has a number of DGZs within it (that have not yet received their full RV allotment) that is equal to the number of RVs on a missile. The algorithm then attempts to construct a feasible missile sortie that targets each of these DGZs once and that emphasizes downrange-to-uprange PBV travel. If a feasible sortie using these DGZs is not possible, then the algorithm defers DGZs nearest the target centroid and tries to form a feasible sortie using successively smaller numbers of DGZs until either a feasible sortie is found or it is decided, based on user-specified rules, that those DGZs are "outcasts" that cannot be targeted effectively using the input weapon system.

To accommodate situations in which multiple RVs are targeted to DGZs, the ALM defines and uses the concept of layers. The n-th RV assigned to each DGZ is referred to as an nth layer RV, and the collection of such RVs is referred to as the n-th layer. Missile sorties are similarly assigned a layer number. Specifically, a missile sortie's layer number is equal to the lowest layer number of its RVs.

Missile launch locations are assigned to missile sorties as follows:

- The number of n-th layer missile sorties, Nn, is identified.
- The Nn downrange-most, unassigned missile launch locations are identified.
- Missile launch location are assigned to missile sorties in a right-to-left direction. This implies near-parallel, as opposed to crossing, missile trajectories.

Once the missile sorties have been defined, a launch schedule is constructed that attempts to minimize either the RV downtime or missile launch time span. If the objective is to minimize launch span, then the first-layer missiles are launched as quickly as possible (either simultaneously or at the maximum launch rate), and subsequent layer missiles are launched to satisfy RV inter-arrival time and launch rate constraints. If the objective is to minimize downtime span, then the first-layer missile launch times are set (subject to launch rate constraints) to achieve simultaneous downtimes for the first RV deployed from each first-layer missile, and subsequent layer missiles are again launched so as to satisfy RV inter-arrival time and launch rate constraints.

### 5.1.1 Program ALM

The main program (ALM: Figure 5-1) performs 12 major steps. Steps 1 through 6 involve reading ALM input files and checking the input data. ALM control data, RV allocation data, launch location data, booster data, PBV data, and RV data are read from the first six ALM input files. Then ALM calls TRYFLY to see if the missile closest to the target island can drop all its RVs on the DGZ farthest from the launch complex. If not, ALM stops immediately; there's no point in allocating RVs if missiles can't reach the target island.

If the trial flight succeeds, however, ALM checks the number of Fratricide Constraints Files supplied in the Control File. If greater than zero, ALM reads up to two (optional) input files, extracting the sure-safe time from each file and associating it with a height of burst. Next ALM calls ASR2TI to assign RVs to the target island: this subroutine is the nucleus of the program and calls the greatest number of other subroutines in the program. Finally, ALM calls PREPAR to prepare data for the Attack File, writes it, and also writes the Sortie File.

The three major subroutines called by ALM will be described next, in the

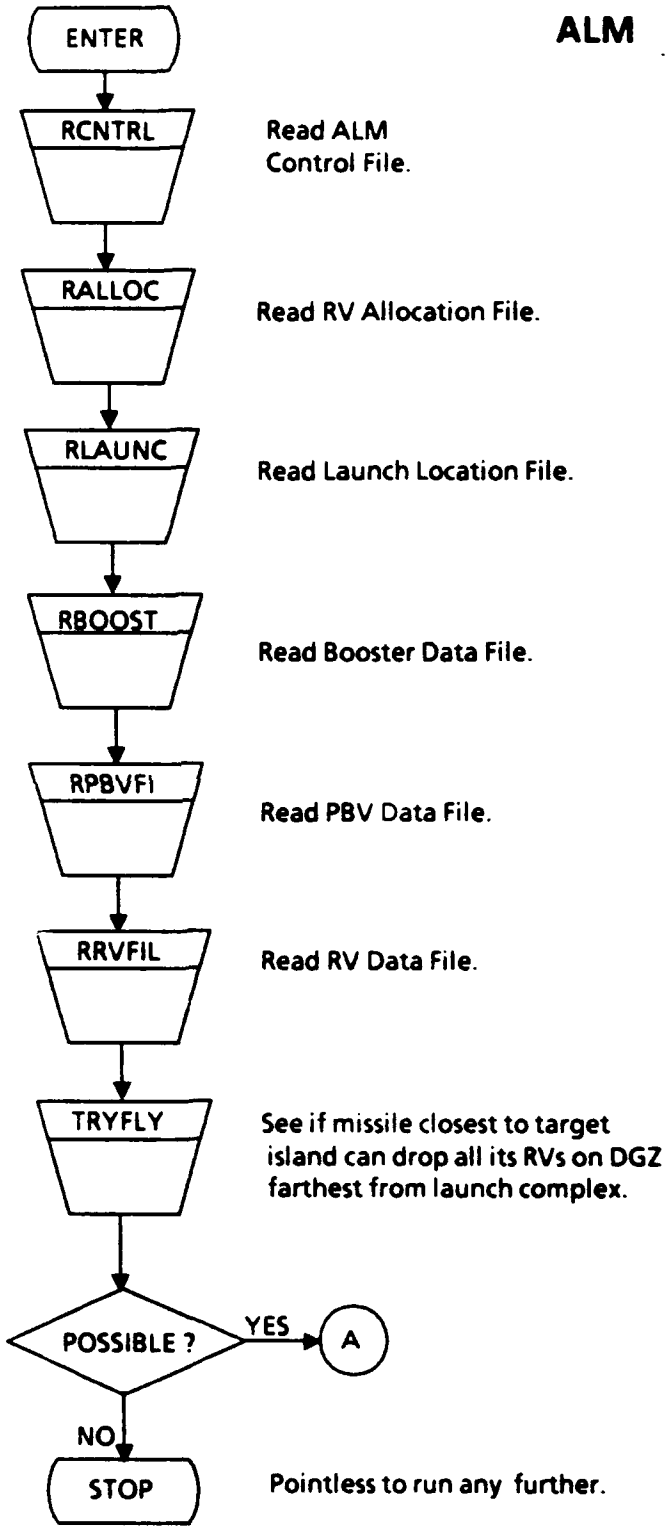



**FIGURE 5-1 PROGRAM ALM PROCESS FLOW**

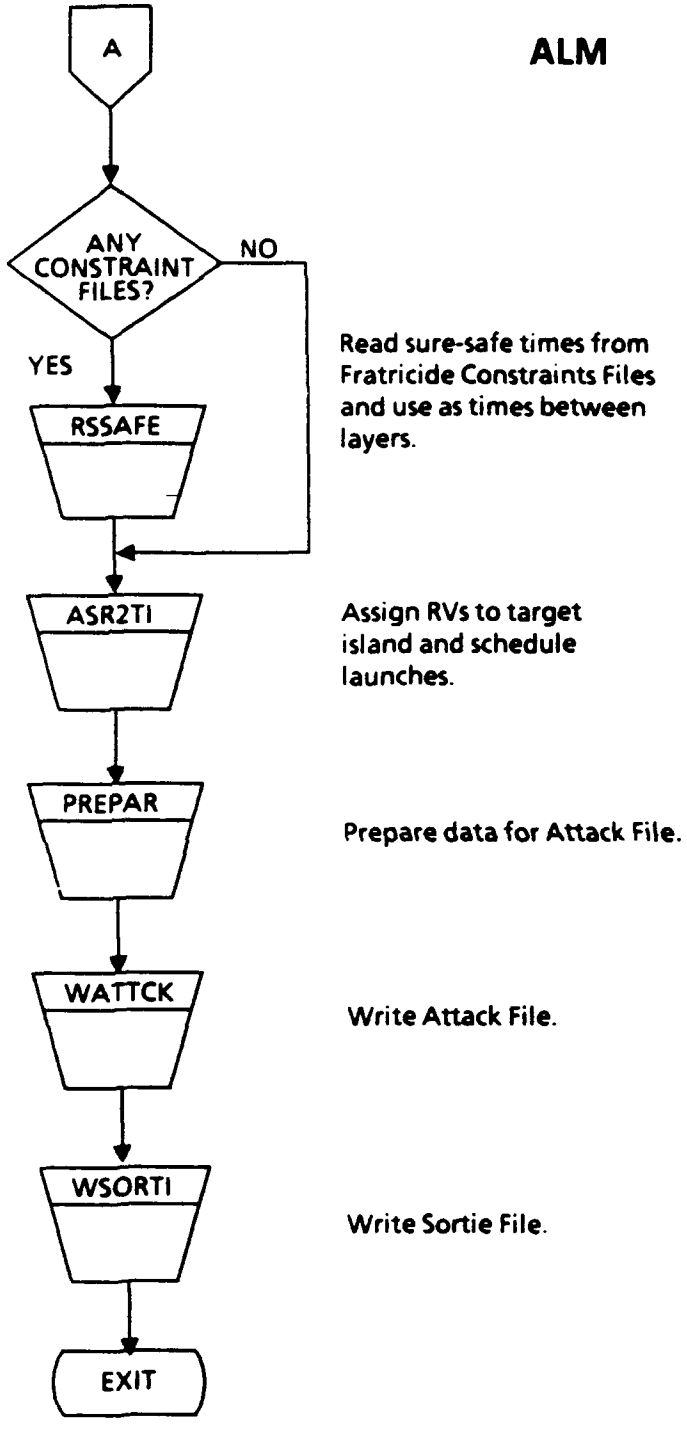



**FIGURE 5-1 PROGRAM ALM PROCESS FLOW (Concluded)**

order they're called: TRYFLY, ASR2TI, and PREPAR.

### 5.1.2 Subroutine TRYFLY

TRYFLY (Figure 5-2) projects the missile silos, and DGZs, onto a plane tangent to the earth at the target island's centroid. The coordinate system imposed on the plane has its positive X-axis pointing downrange. The silo with the maximum X-coordinate is therefore the one closest to the target island, and the DGZ with the maximum X-coordinate is the one farthest from the launch complex. TRYFLY forms a sortie consisting of the farthest DGZ, repeated as many times as there are RVs on a missile. Then TRYFLY calls CHKSRT to see if the missile can perform the sortie. TRYFLY returns a logical flag that's TRUE if the missile can perform the sortie, FALSE if it can't.

### 5.1.3 Subroutine ASR2TI

ASR2TI (Figure 5-3) does some preliminary work before actually assigning RVs to DGZs. The most important steps in this phase are to define the right-to-left order in which DGZs will be scanned, and the order in which missiles will be used. The latter step is performed by subroutine ORDMSS, described later. ASR2TI calls ASR2T2 to construct the sorties that target RVs to DGZs. If requested, ASR2TI calls SYNTIM to "synchronize" the downtimes of first-layer RVs. Note that true synchronization is possible only if the time between launches is zero. Then ASR2TI calls ADJST1 to adjust launch times and downtimes for subsequent layers. Finally, ASR2TI calls FXLYRS to fix the layer number associated with each RV; i.e., to ensure that layer numbers and downtimes are both in ascending order for each string of RVs going to the same DGZ.

### 5.1.4 Subroutine PREPAR

PREPAR (Figure 5-4) prepares data for the Attack File by sorting missile and RV data. Missile launch times and RV downtimes are in ascending order. PREPAR also computes the launch duration and laydown duration. Finally, PREPAR zeros the array containing probabilities of fratricide for each RV.

This concludes the description of major subroutines called by ALM. The major subroutines called by ASR2TI will be described next, in the order they're called: ORDMSS, ASR2T2, and ADJST1.

### 5.1.5 Subroutine ORDMSS

ORDMSS (Figure 5-5) projects the missile silos onto a plane tangent to the earth at the launch complex's centroid. The coordinate system imposed on the plane has its positive X-axis pointing

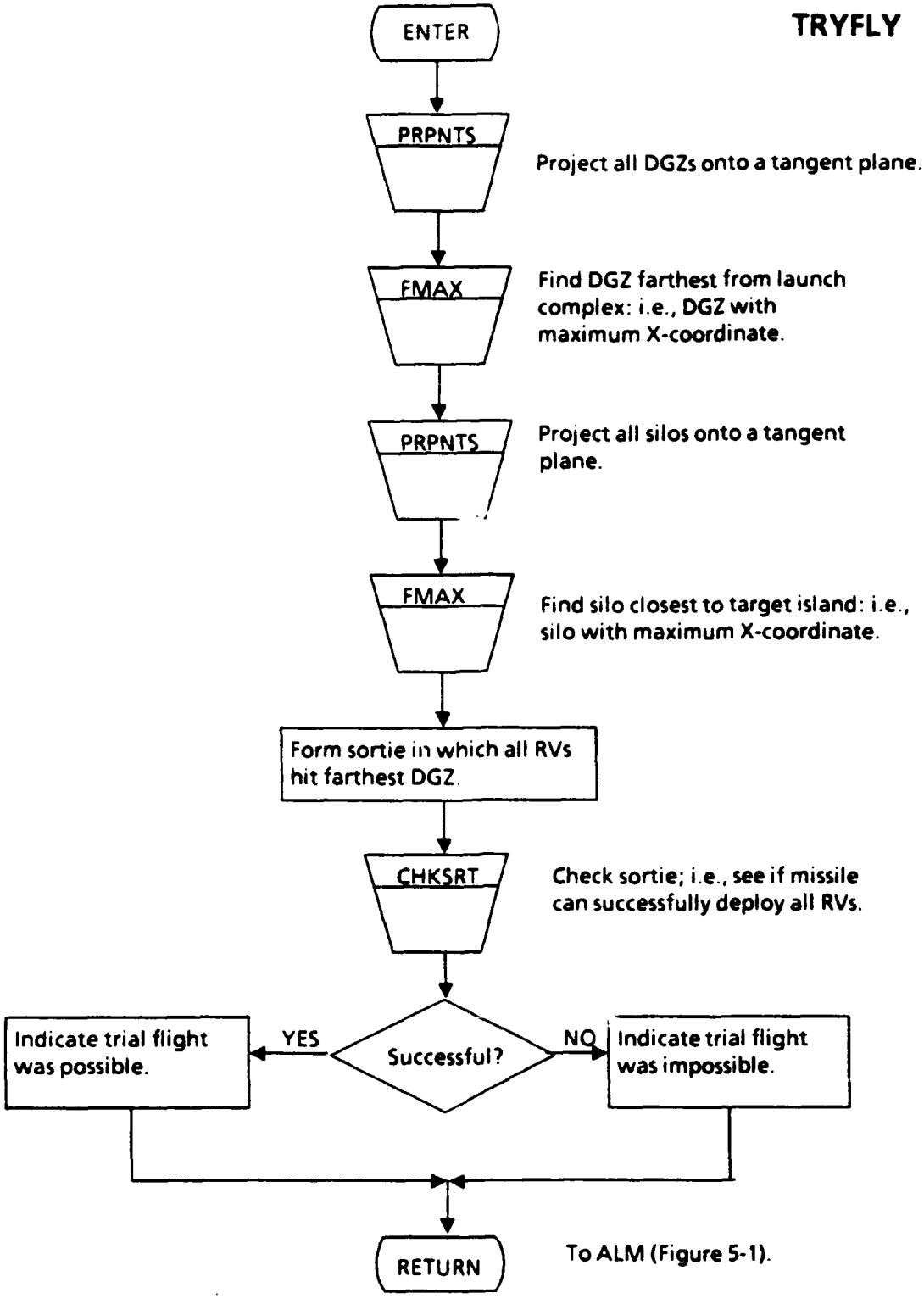



**FIGURE 5-2 SUBROUTINE TRYFLY PROCESS FLOW**

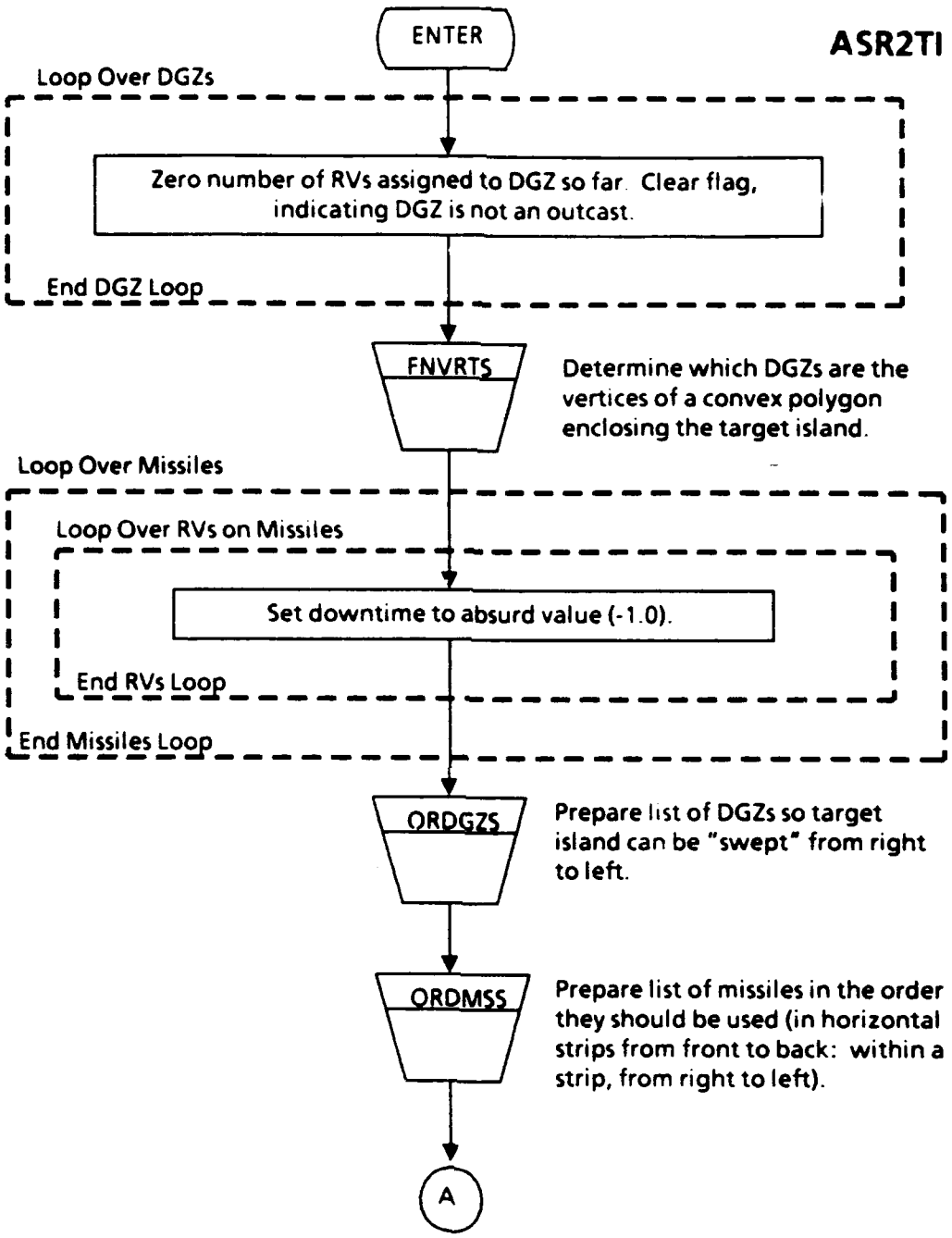



**FIGURE 5-3 SUBROUTINE ASR2TI PROCESS FLOW**

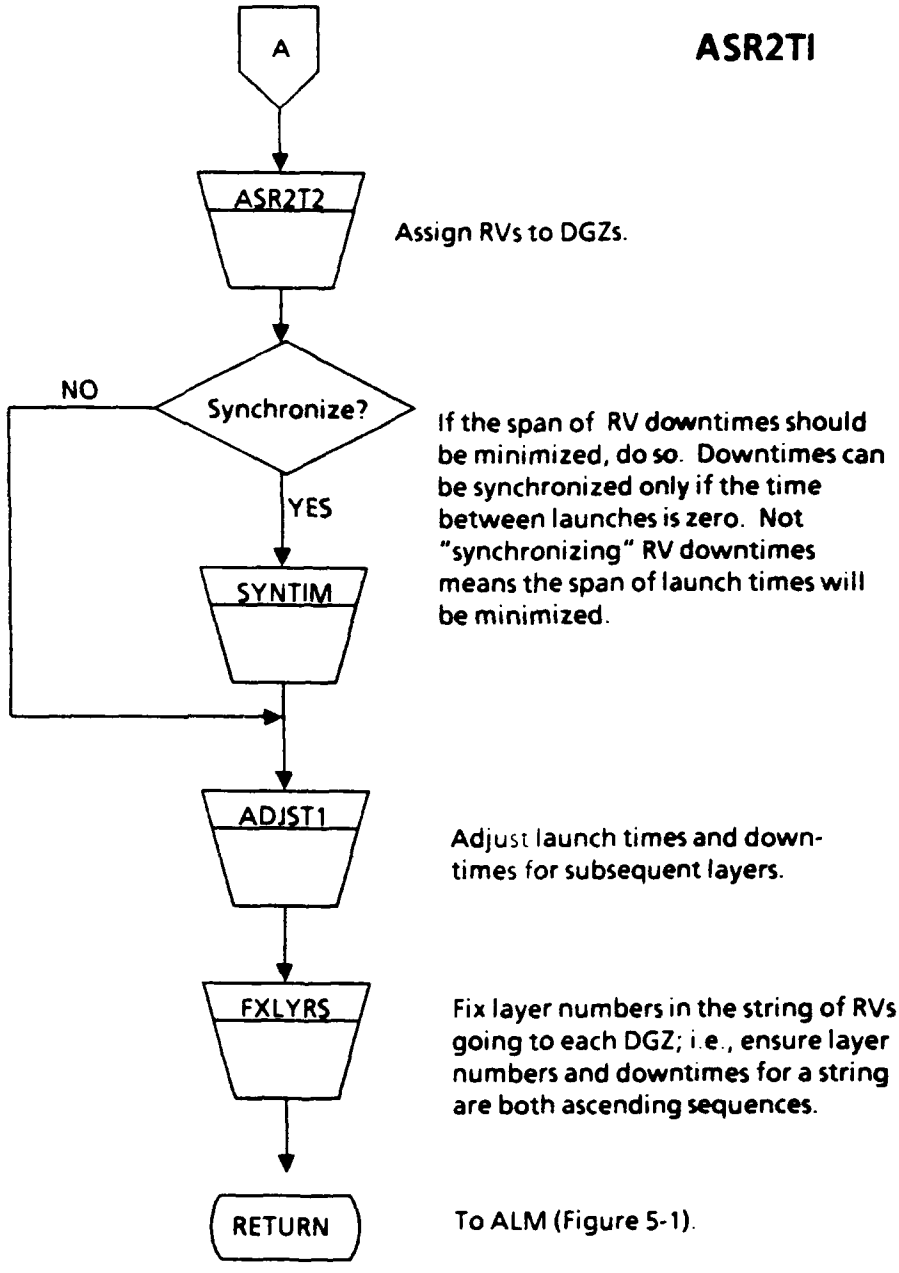



**FIGURE 5-3 SUBROUTINE ASR2TI PROCESS FLOW (Concluded)**

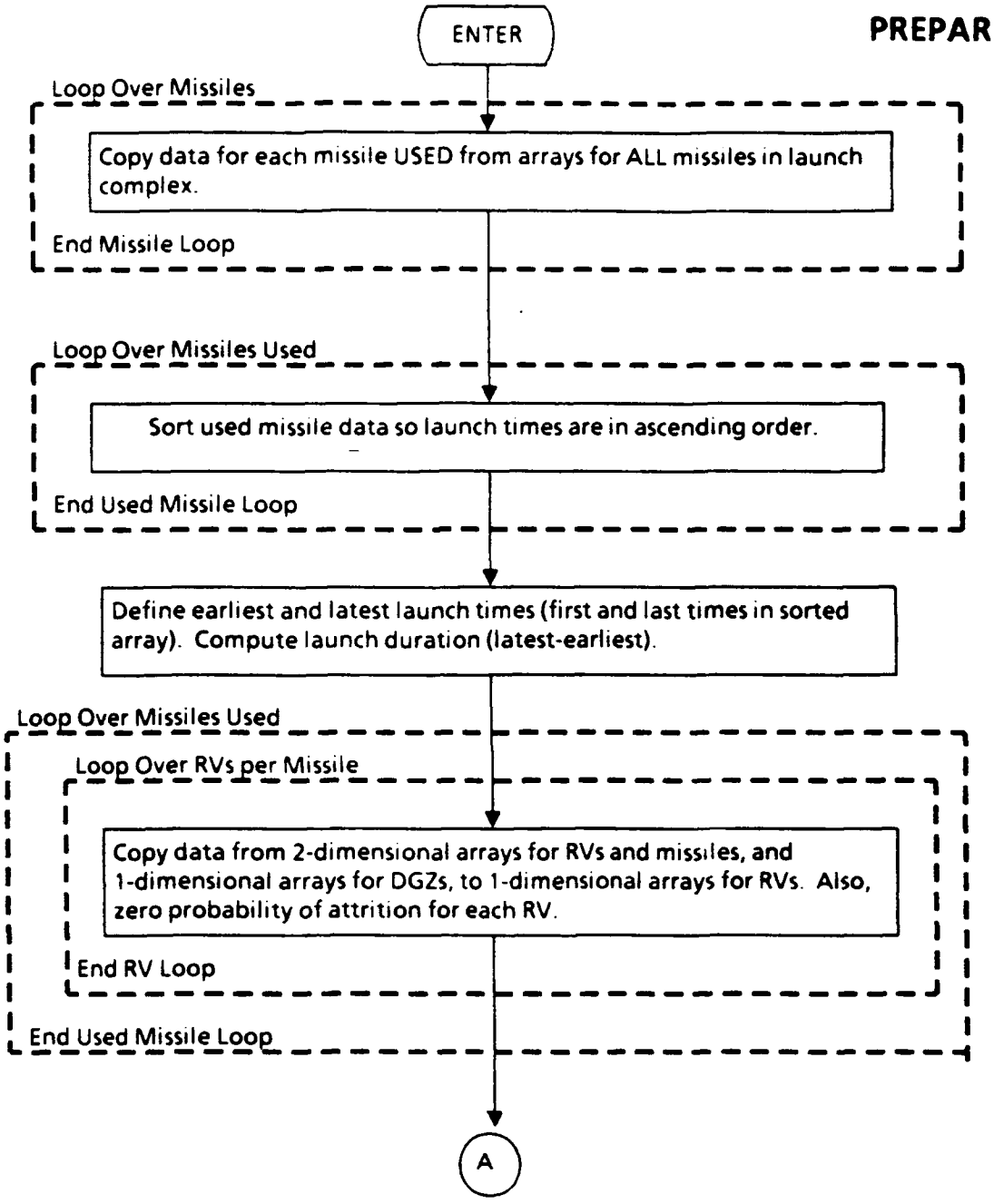



**FIGURE 5-4 SUBROUTINE PREPAR PROCESS FLOW**

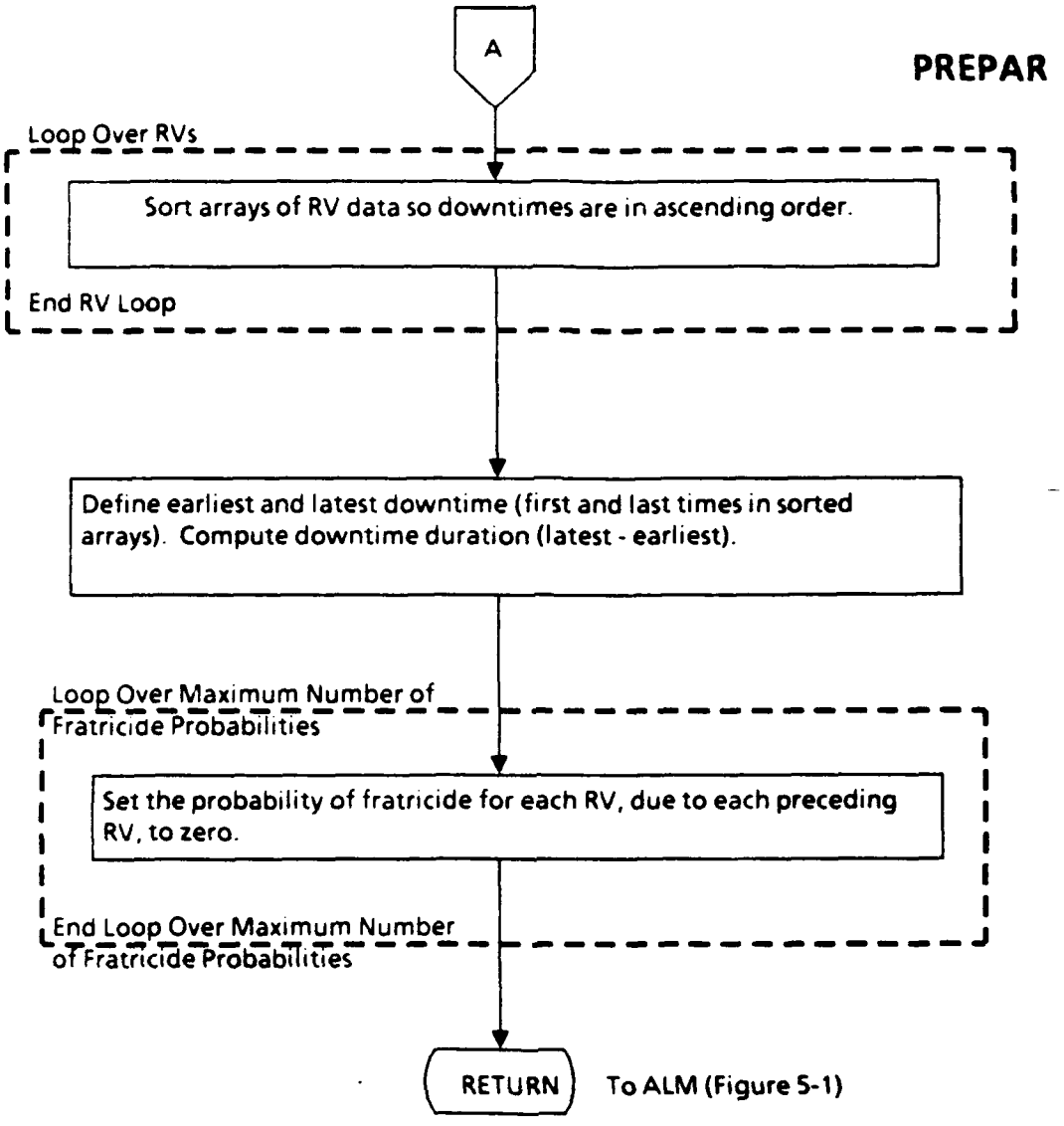



**FIGURE 5-4 SUBROUTINE PREPAR PROCESS FLOW (Concluded)**

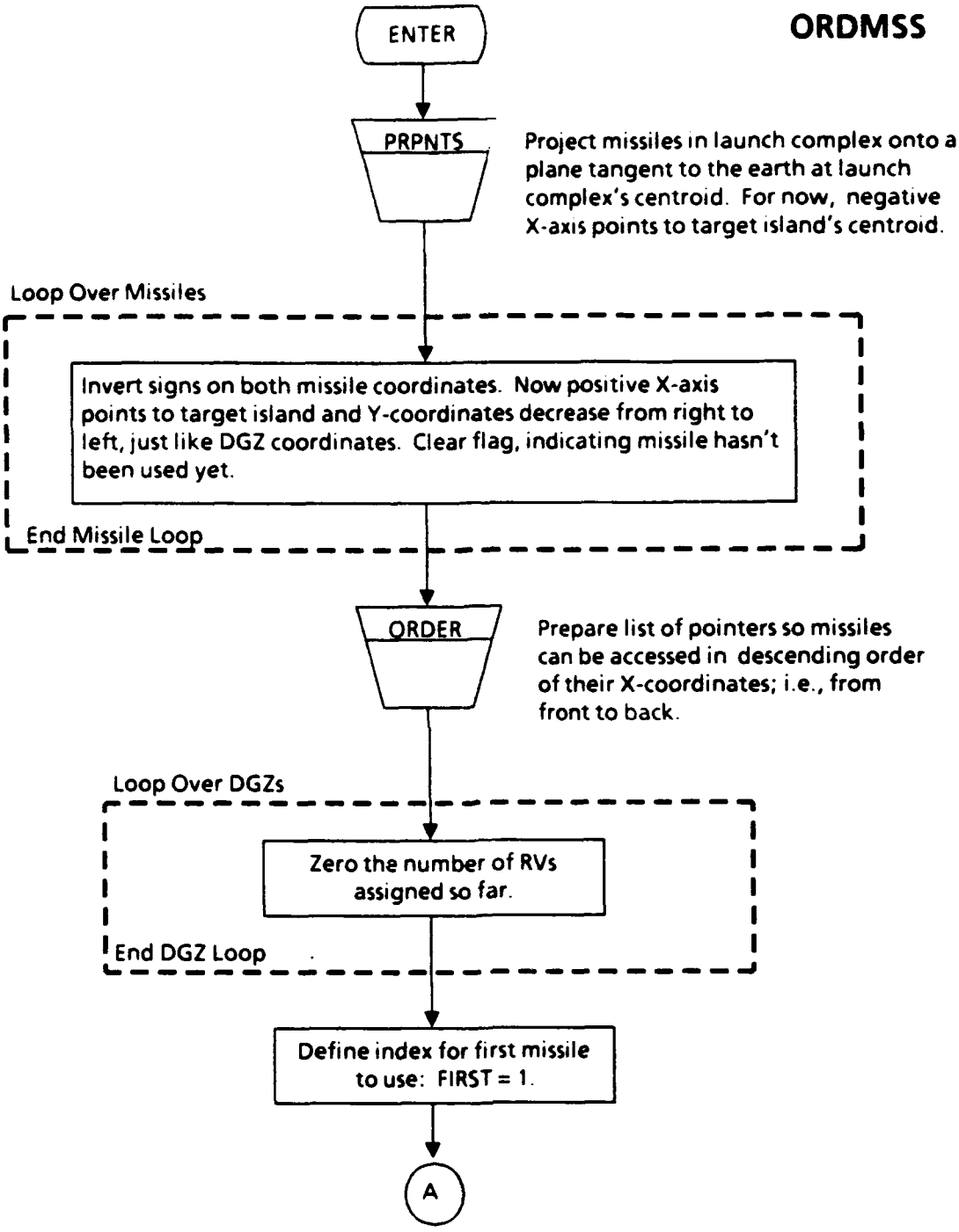



**FIGURE 5-5 SUBROUTINE ORDMSS PROCESS FLOW**

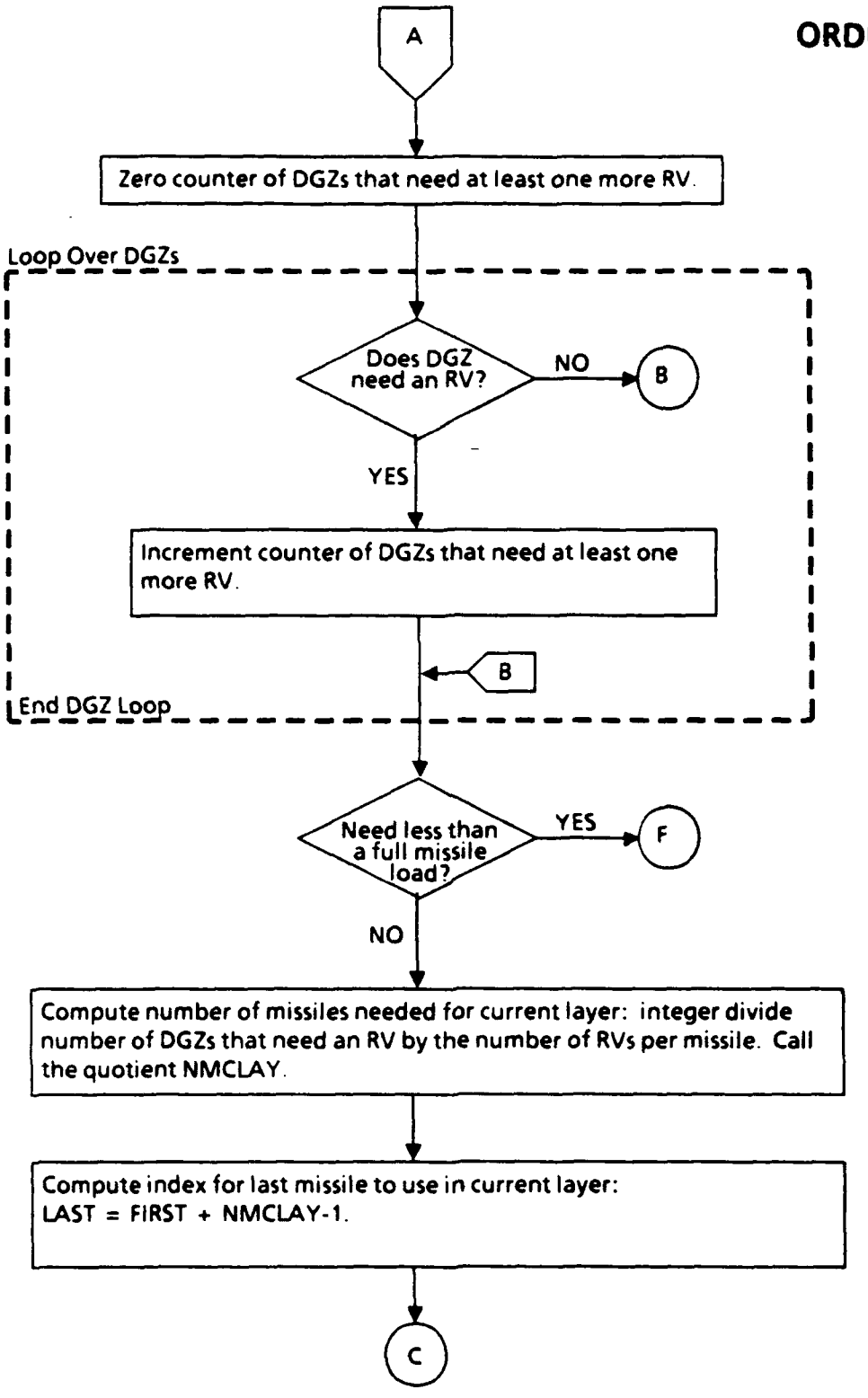



**FIGURE 5-5 SUBROUTINE ORDMSS PROCESS FLOW (Continued)**

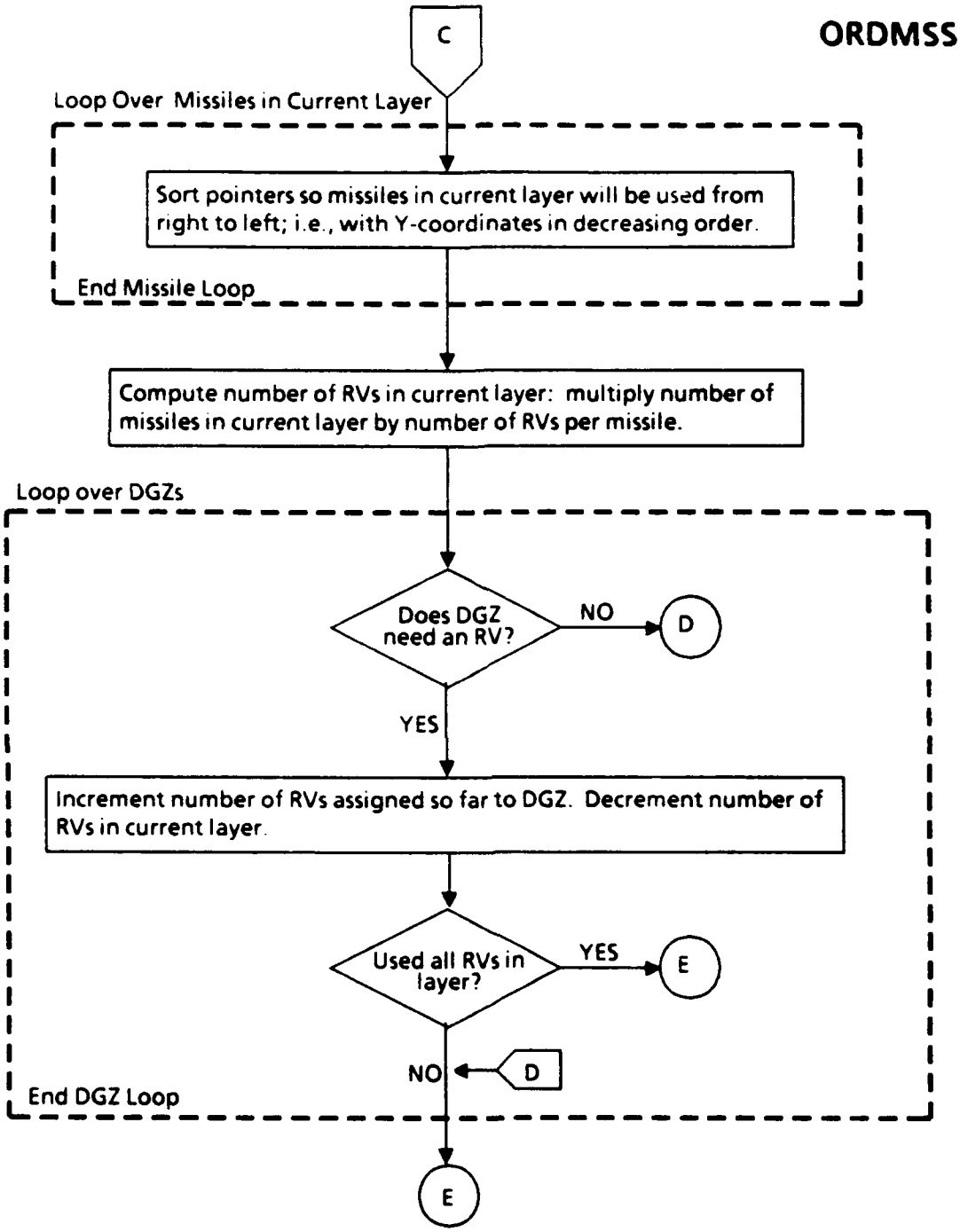



**FIGURE 5-5 SUBROUTINE ORDMSS PROCESS FLOW (Continued)**

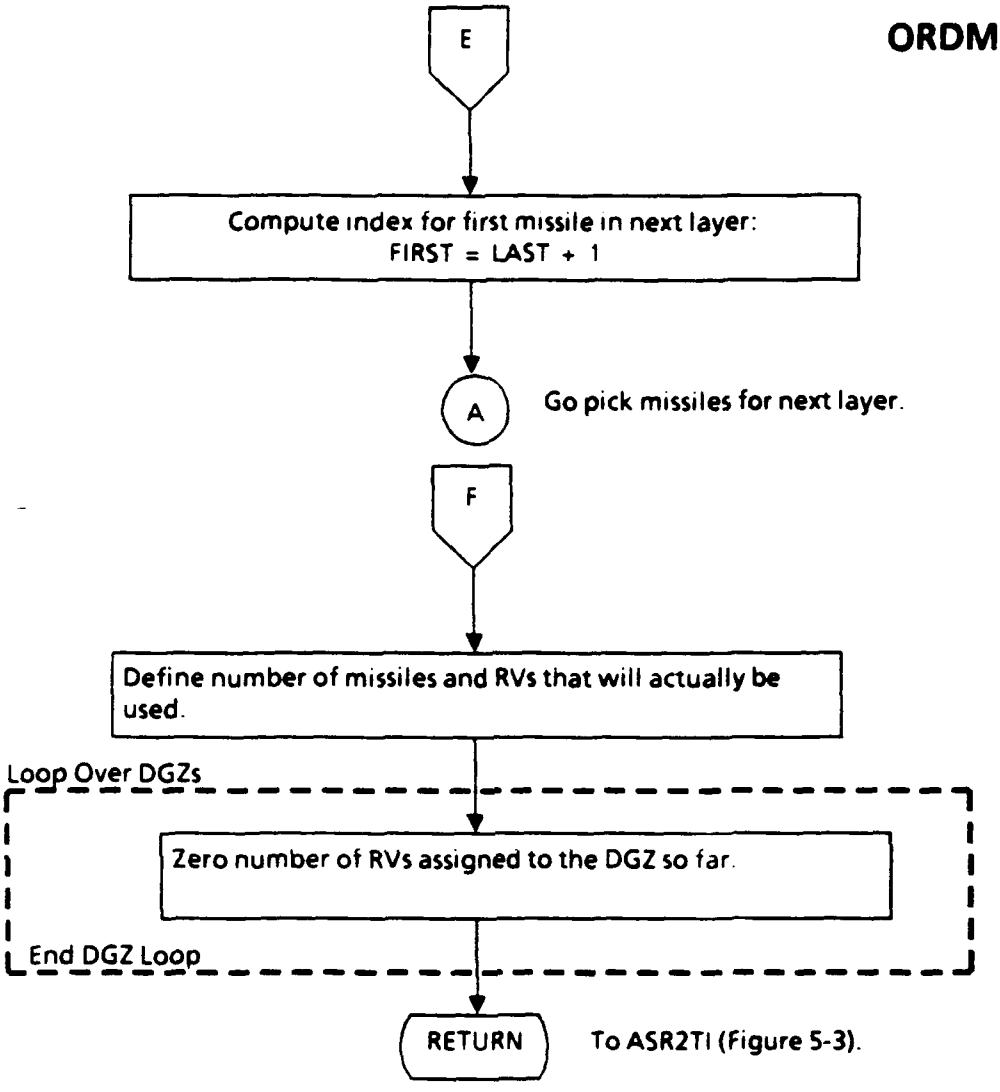



**FIGURE 5-5 SUBROUTINE ORDMSS PROCESS FLOW (Concluded)**

downrange (towards the target island), and the edge of the launch complex closest to the target island is considered the "front" edge. ORDMSS prepares a list of pointers to the missi so they can be accessed in front-to-back order.

ORDMSS assigns missiles layer-by-layer. The missiles used for a layer lie in a "horizontal" strip (i.e., a strip parallel to the Y-axis) and are used in right-to-left order. The number of RVs in a layer equals the number of DGZs that still need at least one more RV. The number of missiles in a layer is the integer quotient of the number of RVs in the current layer divided by the number of RVs per missile. A counter associated with each DGZ is initially cleared, and incremented each time an RV is assigned to the DGZ. ORDMSS resets these counters to zero so they can be used again to actually assign RVs to DGZs later on.

### 5.1.6 Subroutine ASR2T2

ASR2T2 (Figure 5-6) forms sorties for all missiles to be used, by repeatedly sweeping across the target island from right to left. The candidates for a sortie consist of the first $n$ DGZs it encounters that need at least one more RV, where $n$ is the number of RVs on a missile. There are two main types of sortie: single-layer and multi-layer. In a single-layer sortie, each DGZ appears only once, meaning it is attacked by an RV from one and only one layer. In a multi-layer sortie, at least one DGZ appears more than once, meaning it is attacked by RVs from two or more layers.

If ASR2T2 forms a single-layer sortie, it sequentially checks to see whether the next missile to use can perform one of the following four variations of a single-layer sortie: uprange, split, crossrange, or downrange. If the missile can't do any of these sorties, ASR2T2 forms a multi-layer sortie from the single-layer sortie by eliminating a DGZ and reassigning its RV to one of the remaining DGZs.

If ASR2T2 forms a multi-layer sortie, it checks to see whether the next missile to use can perform the sortie (or a couple of variations of it). If not, ASR2T2 forms a new multi-layer sortie by eliminating a DGZ and checks the new sortie.

If a sortie is reduced to just two DGZs, or if a sortie wastes RVs, one of the DGZs is marked as an outcast. Then, ASR2T2 tries to form another sortie. With the outcast no longer considered, perhaps a feasible sortie can be formed after all. If a partial missile load is required at the end, ASR2T2 calls MOPUP to use the remaining RVs as efficiently as possible.

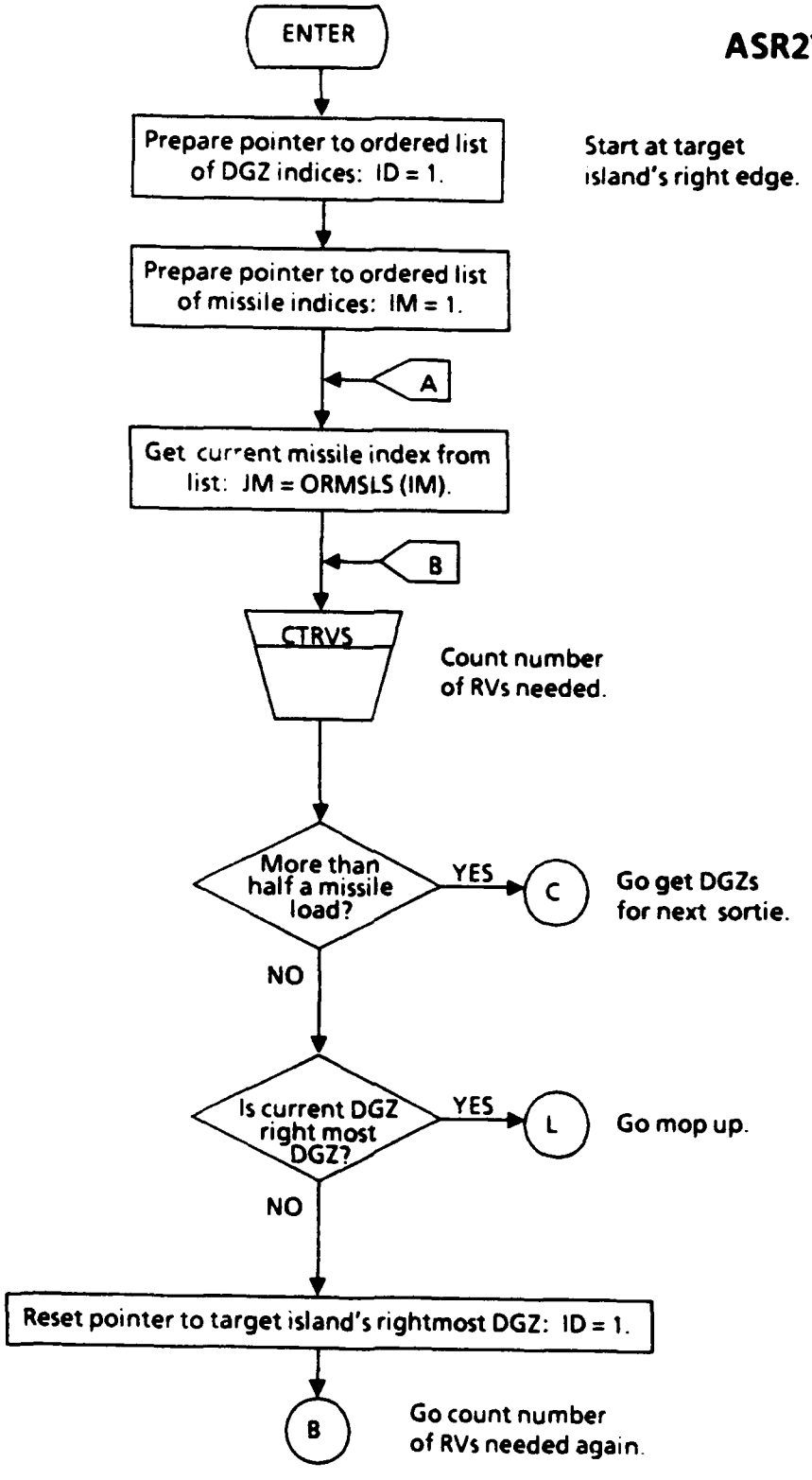



**FIGURE 5-6 SUBROUTINE ASR2T2 PROCESS FLOW**

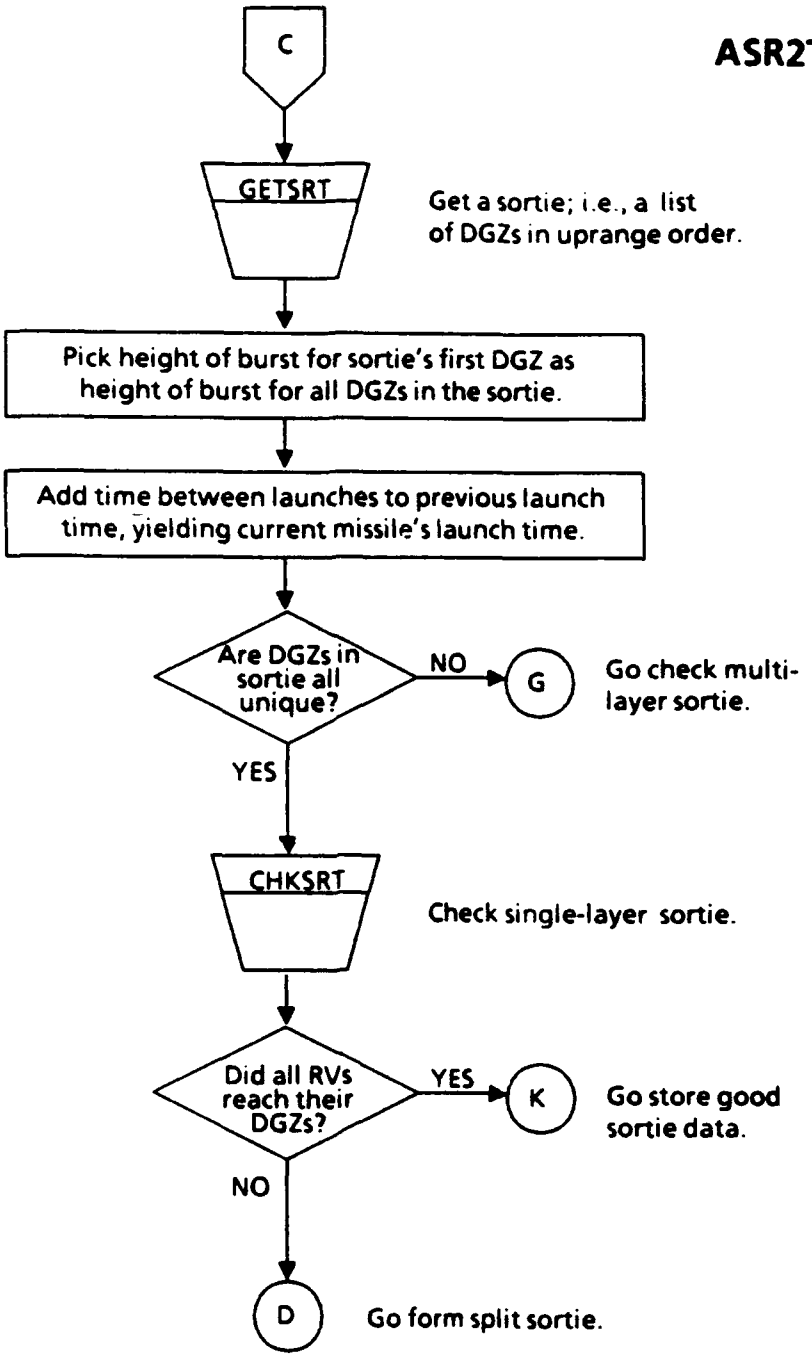



**FIGURE 5-6 SUBROUTINE ASR2T2 PROCESS FLOW (Continued)**

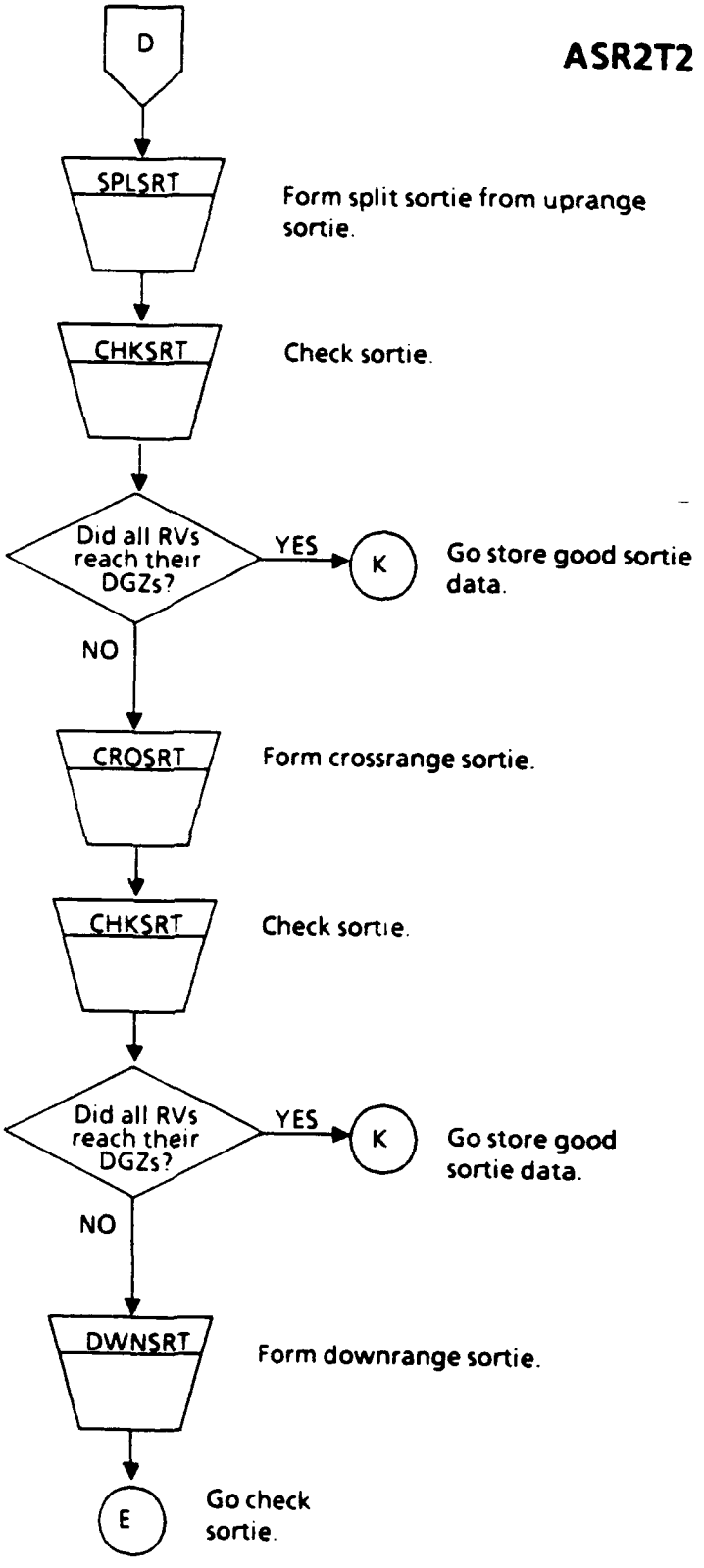



**FIGURE 5-6 SUBROUTINE ASR2T2 PROCESS FLOW (Continued)**

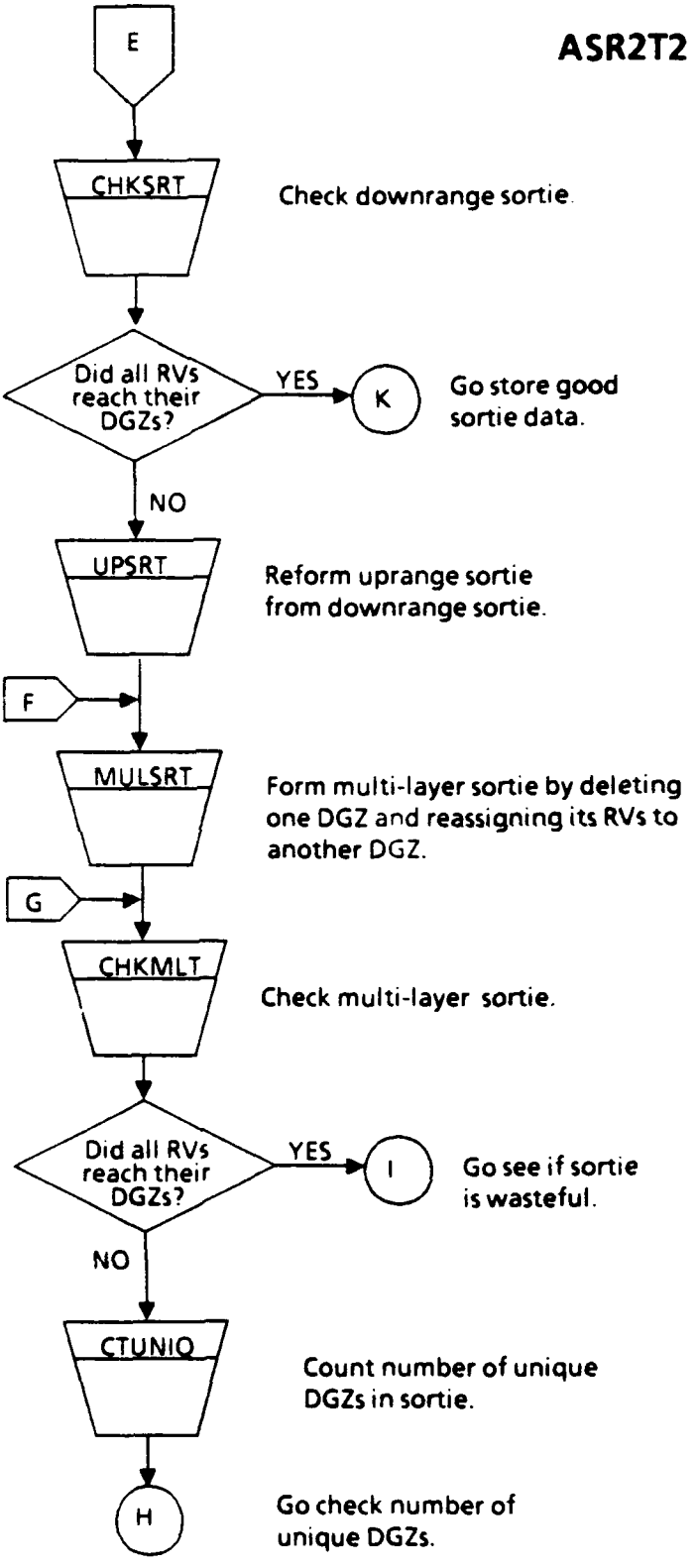



**FIGURE 5-6 SUBROUTINE ASR2T2 PROCESS FLOW (Continued)**

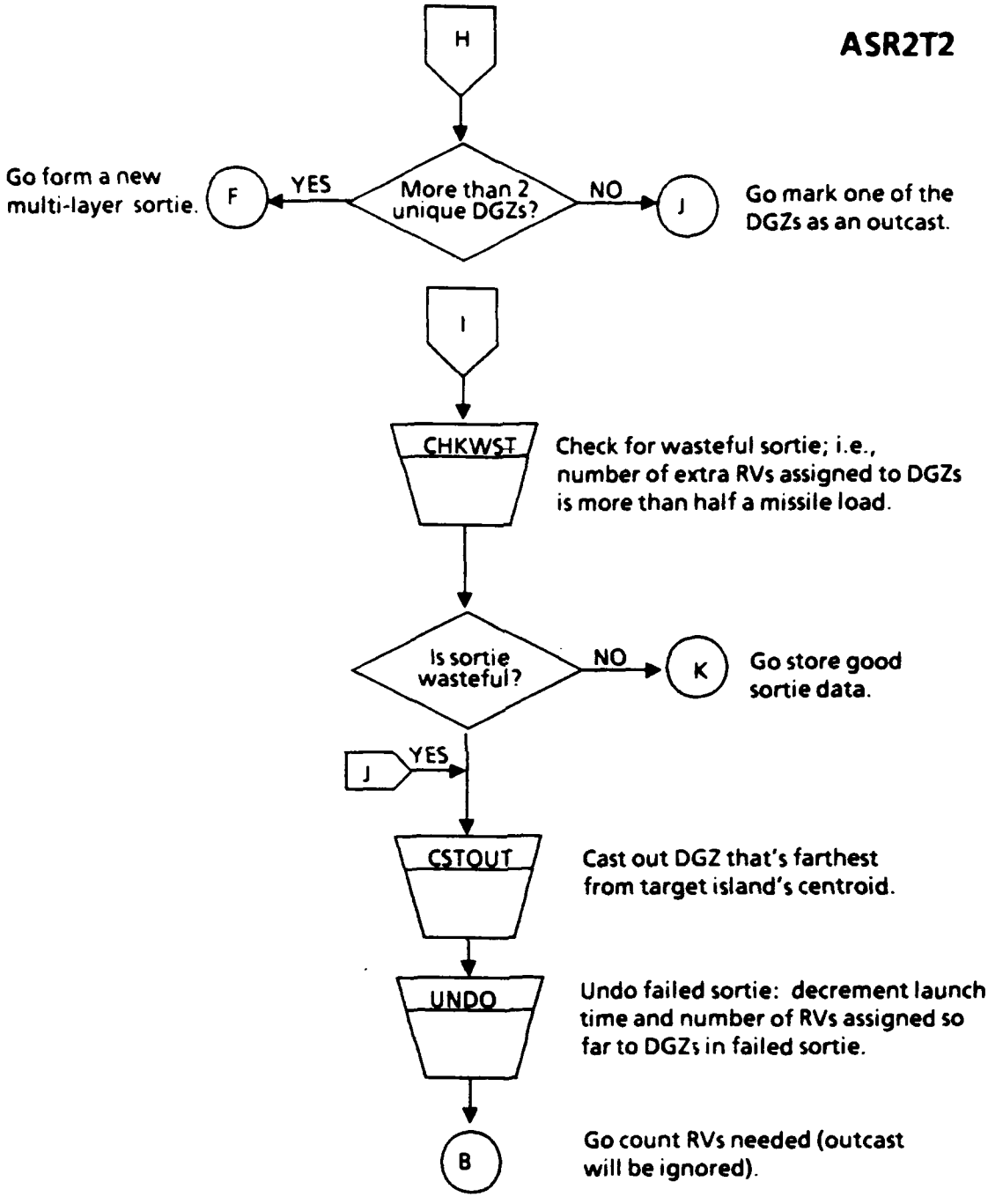



**FIGURE 5-6 SUBROUTINE ASR2T2 PROCESS FLOW (Continued)**

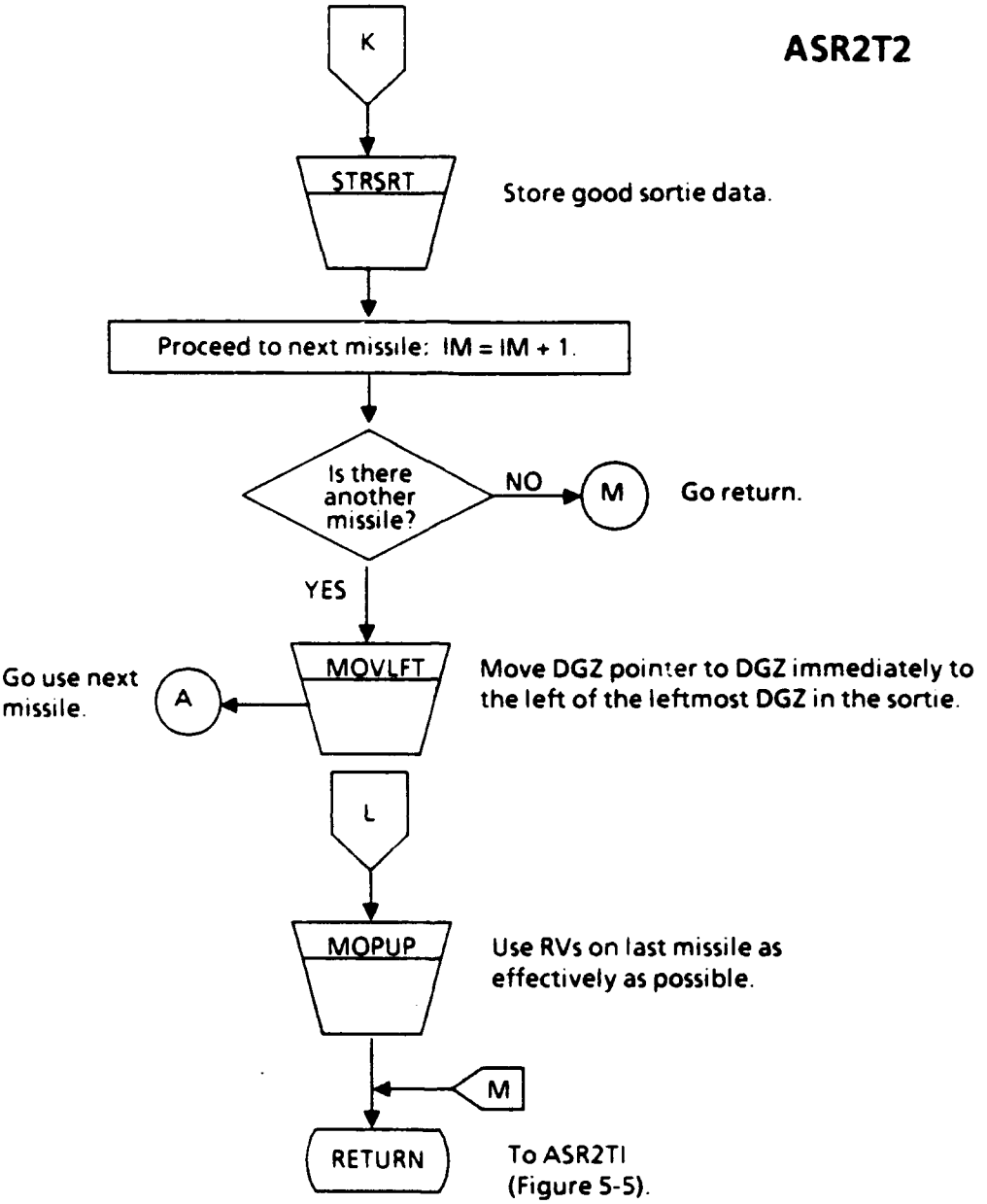



**FIGURE 5-6 SUBROUTINE ASR2T2 PROCESS FLOW (Concluded)**

### 5.1.7 Subroutine ADJST1

ADJST1 (Figure 5-7) adjusts launch times and downtimes for all layers beyond the first. When the user elects to minimize RV downtimes, then just before ADJST1 is called, ASR2TI minimizes the span of downtimes for first-layer RVs (if the time between launches isn't zero), or synchronizes the downtimes (if the time between launches is zero). ADJST1 examines downtimes layer by layer, postponing launches if necessary, so the correct time between layers is maintained.

This concludes the description of major subroutines called by ASR2TI. The major subroutines called by ASR2T2 will be described next, in the order they're called: GETSRT, MULSRT, CHKMLT, and CSTOUT.

### 5.1.8 Subroutine GETSRT

GETSRT (Figure 5-8) repeatedly sweeps across the target island, gathering DGZs that need at least one more RV. The number of DGZs needed for a sortie equals the number of RVs on a missile. DGZs marked as outcasts are ignored during sweeps. If there aren't enough DGZs to complete the final sortie, GETSRT assigns extra RVs to the most valuable DGZs in the partial sortie just formed.

### 5.1.9 Subroutine MULSRT

MULSRT (Figure 5-9) forms a multi-layer sortie from a single-layer (or simpler multi-layer) sortie. It does so by eliminating one DGZ and reassigning its RV(s) to the remaining DGZs. The DGZ eliminated is the one furthest from the sortie's vertical midline.

### 5.1.10 Subroutine CHKMLT

CHKMLT (Figure 5-10) checks to see if a missile can perform a multi-layer sortie. The sortie is first performed by moving uprange; any DGZ assigned more than one RV is hit by consecutive RVs, before proceeding to the next DGZ. If the current missile can't perform the sortie, CHKMLT gives up. If the missile can perform the sortie, CHKMLT checks the time between layers in the sortie. If they're acceptable, CHKMLT returns; if not, CHKMLT tries two variations of the uprange sortie that increase the time between layers.

The first variation moves uprange, hitting each DGZ only once. Then the PBV is instructed to return to the beginning and sweep uprange again, hitting DGZs that get another RV. This is repeated if necessary. If the missile can perform this variation, CHKMLT returns. If not, CHKMLT forms a second variation.

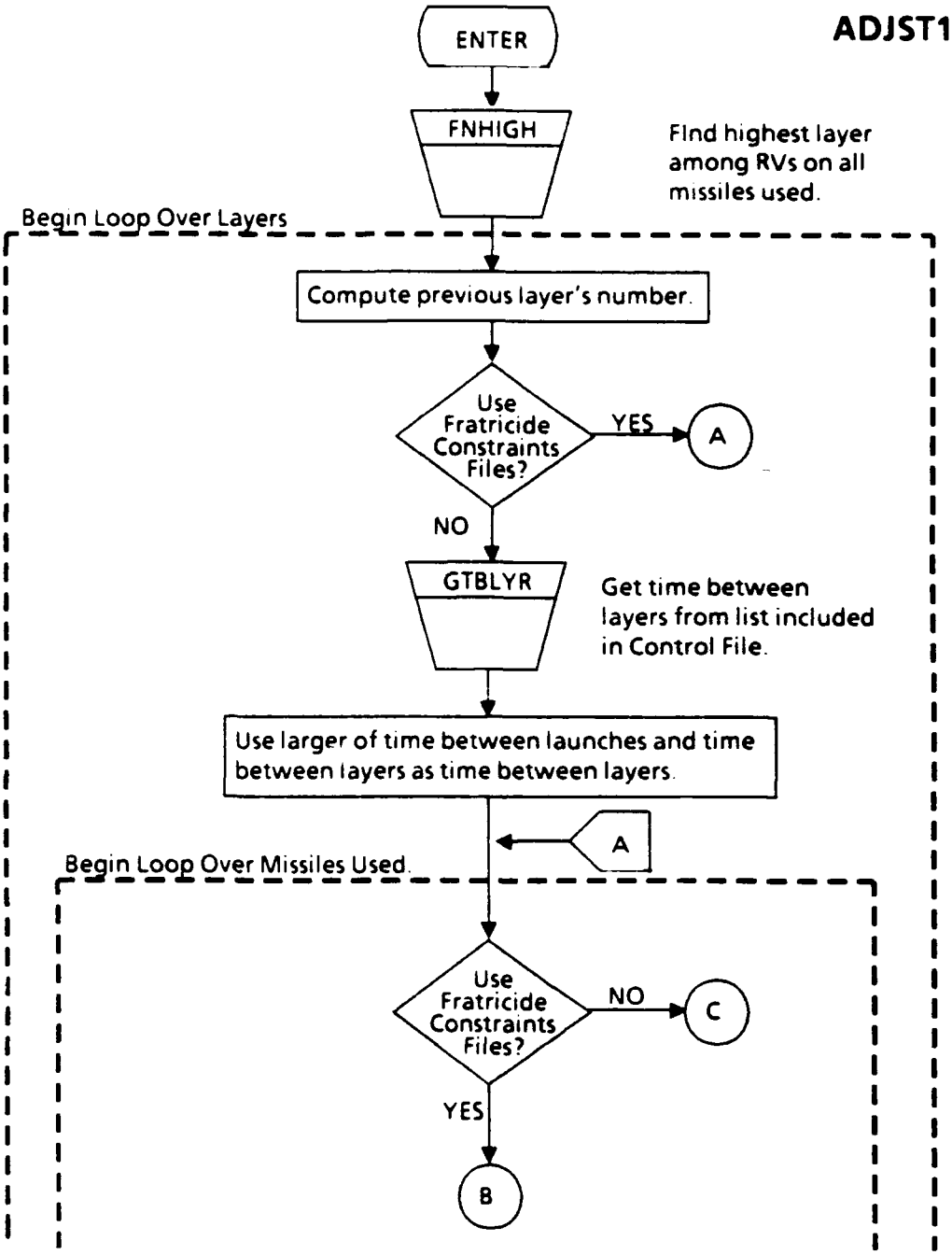



**FIGURE 5-7 SUBROUTINE ADJST1 PROCESS FLOW**

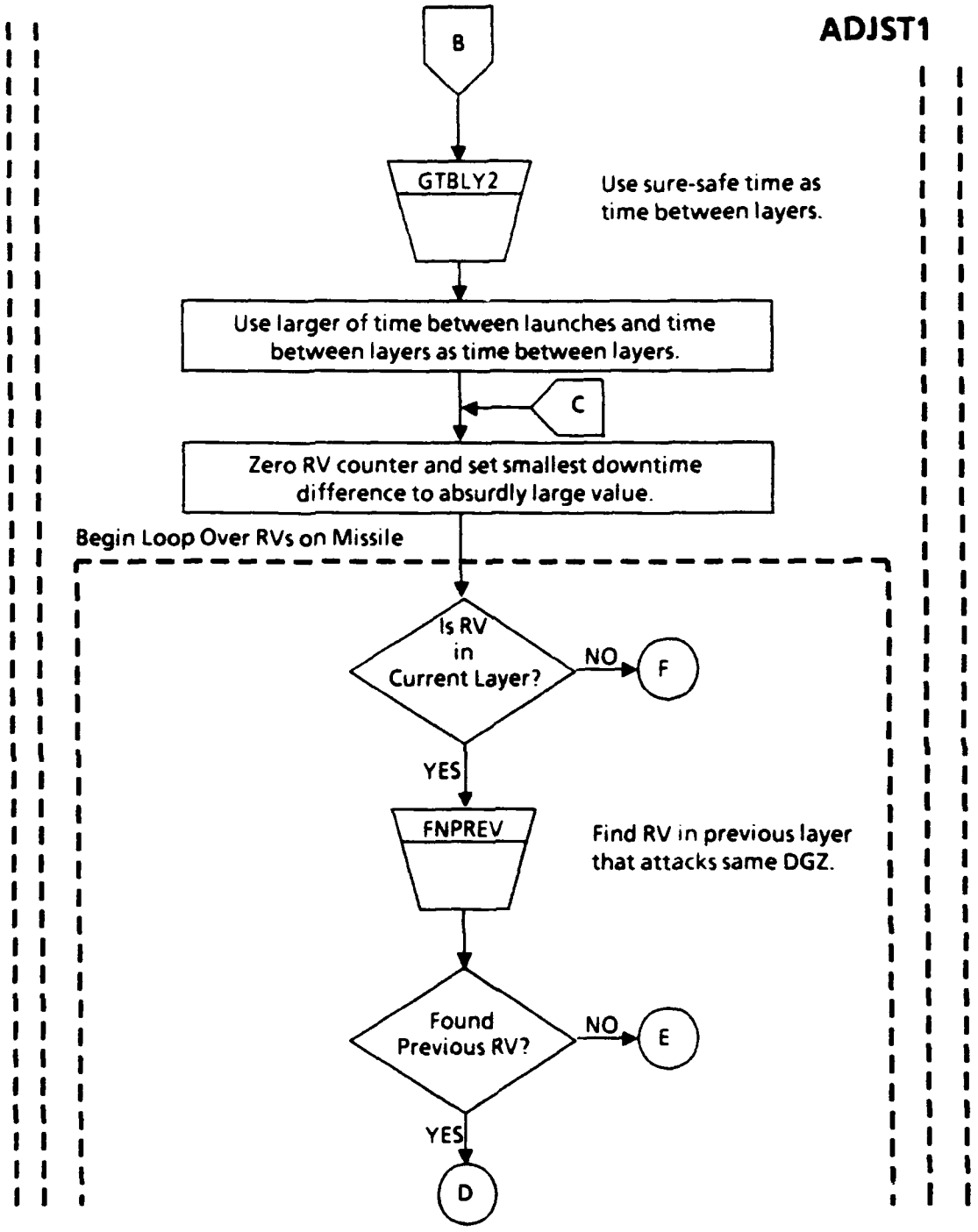



**FIGURE 5-7 SUBROUTINE ADJST1 PROCESS FLOW (Continued)**

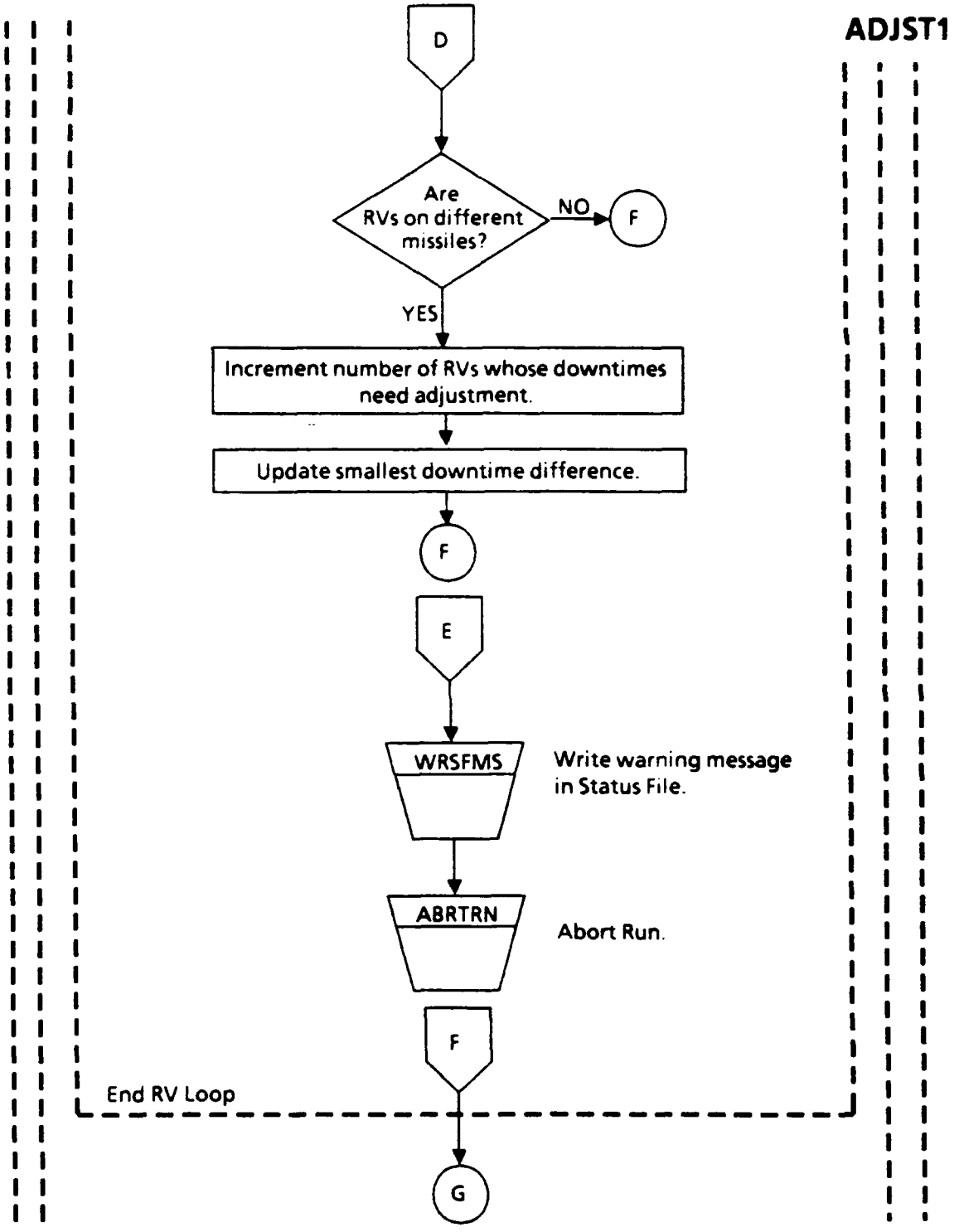



**FIGURE 5-7 SUBROUTINE ADJST1 PROCESS FLOW (Continued)**

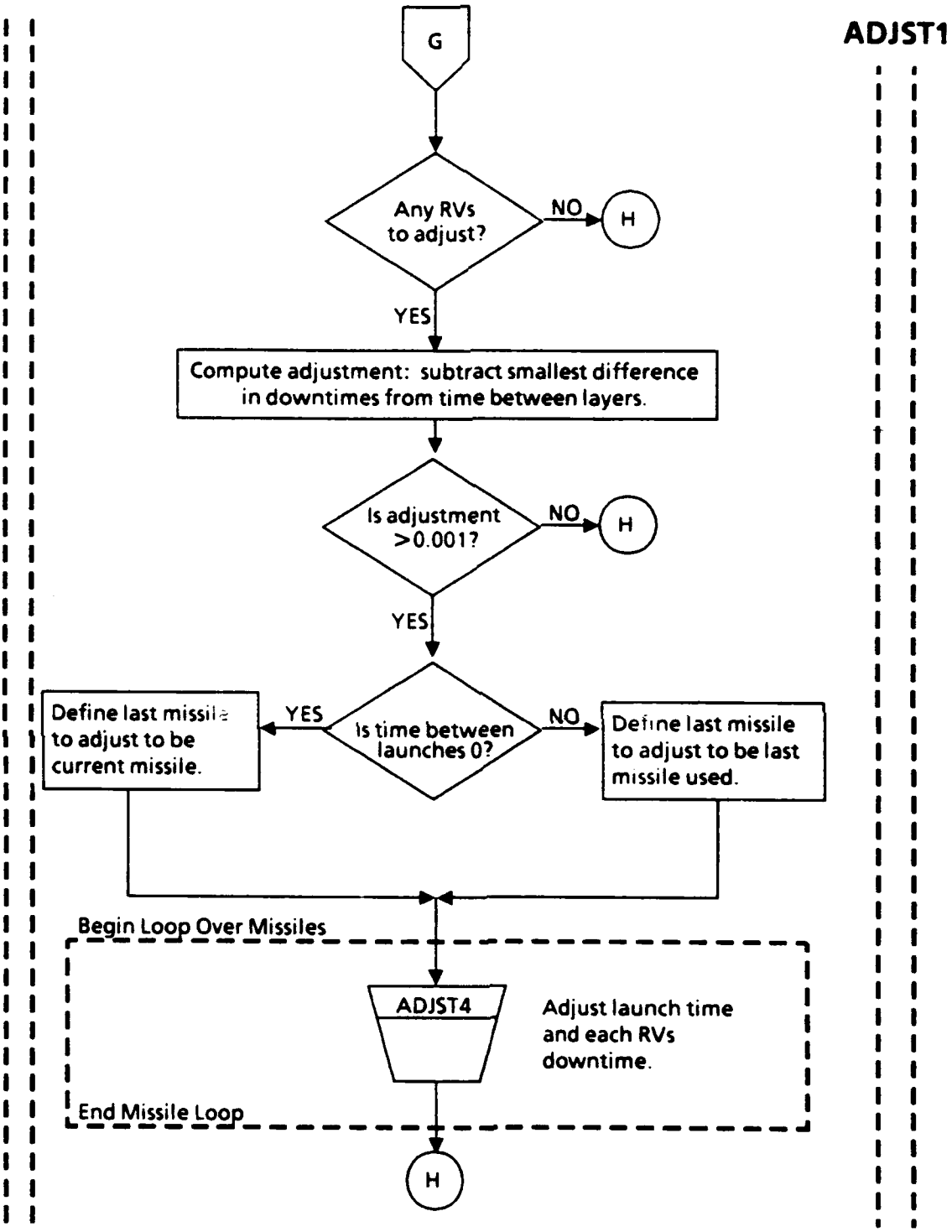



**FIGURE 5-7 SUBROUTINE ADJST1 PROCESS FLOW (Continued)**

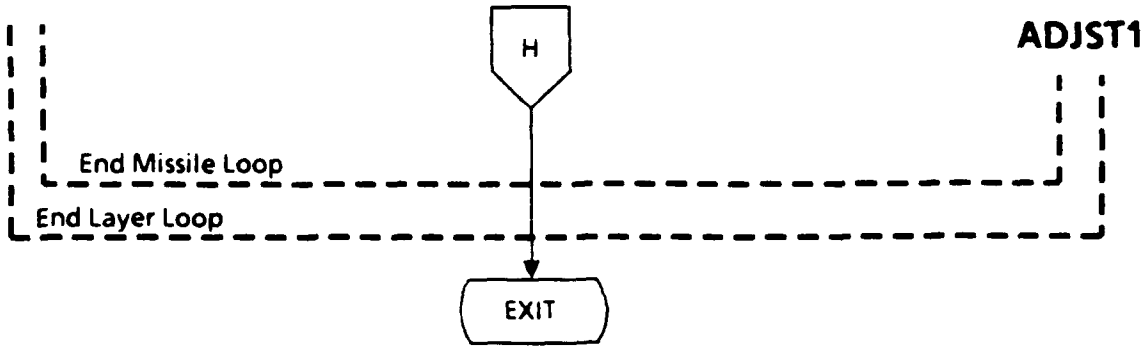



**FIGURE 5-7 SUBROUTINE ADJST1 PROCESS FLOW (Concluded)**

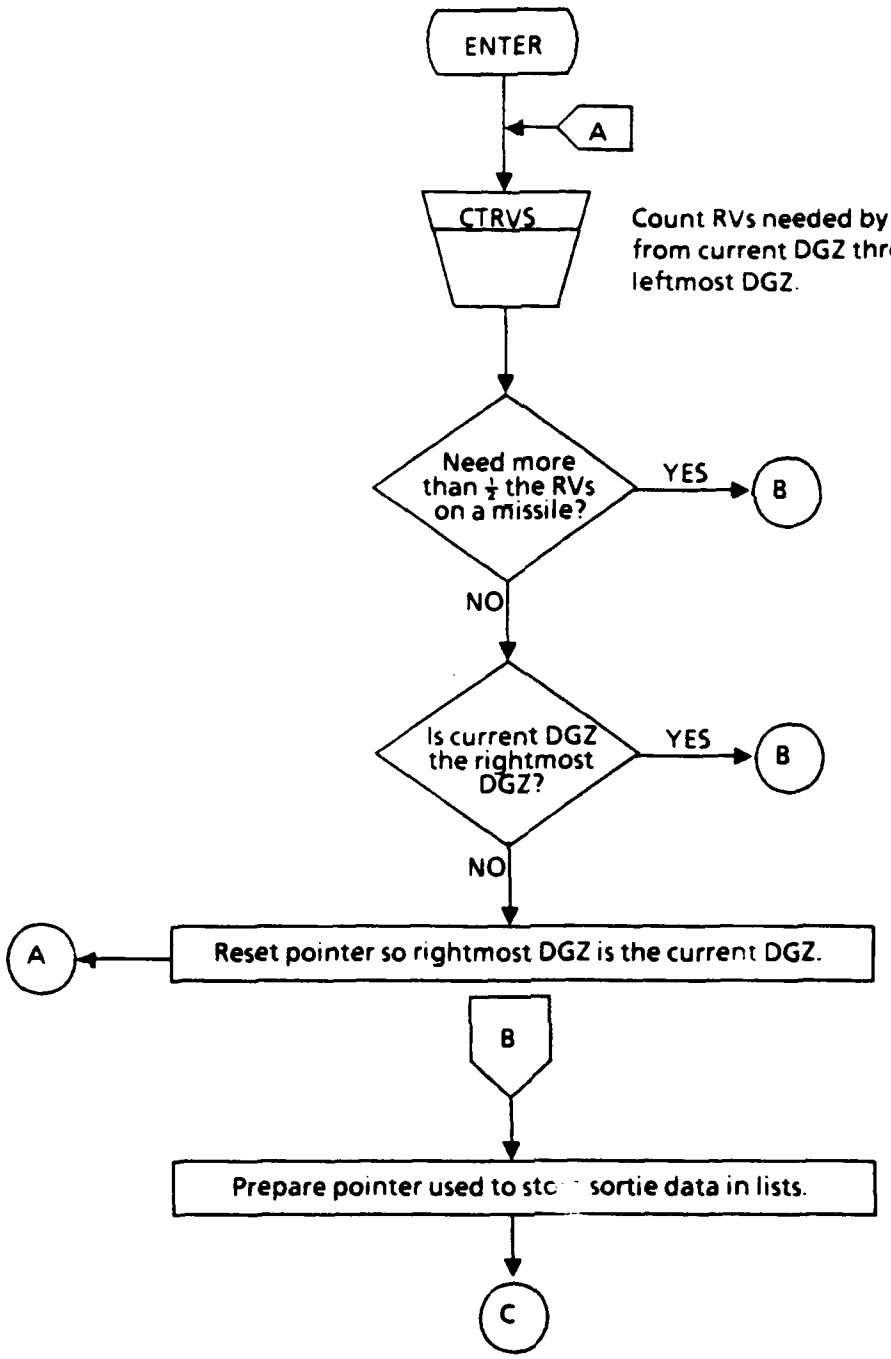



**FIGURE 5-8 SUBROUTINE GETSRT PROCESS FLOW**

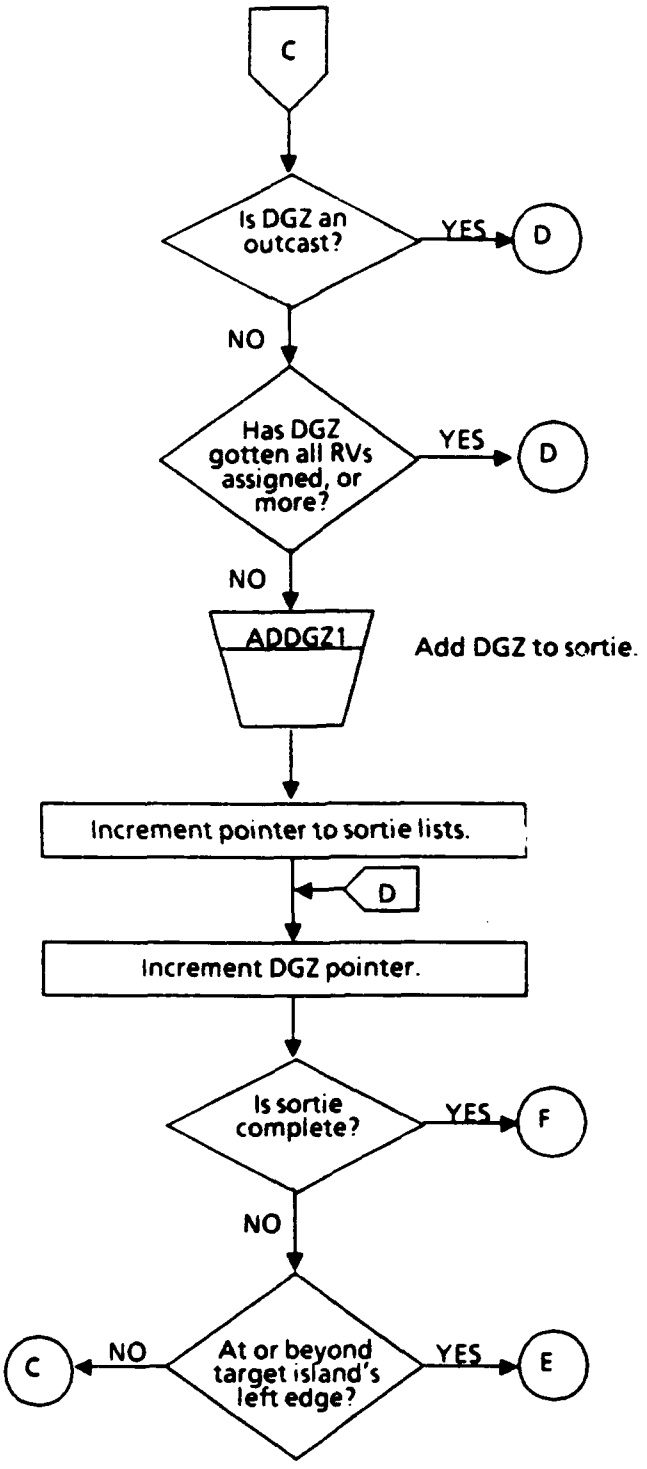



**FIGURE 5-8 SUBROUTINE GETSRT PROCESS FLOW (Continued)**

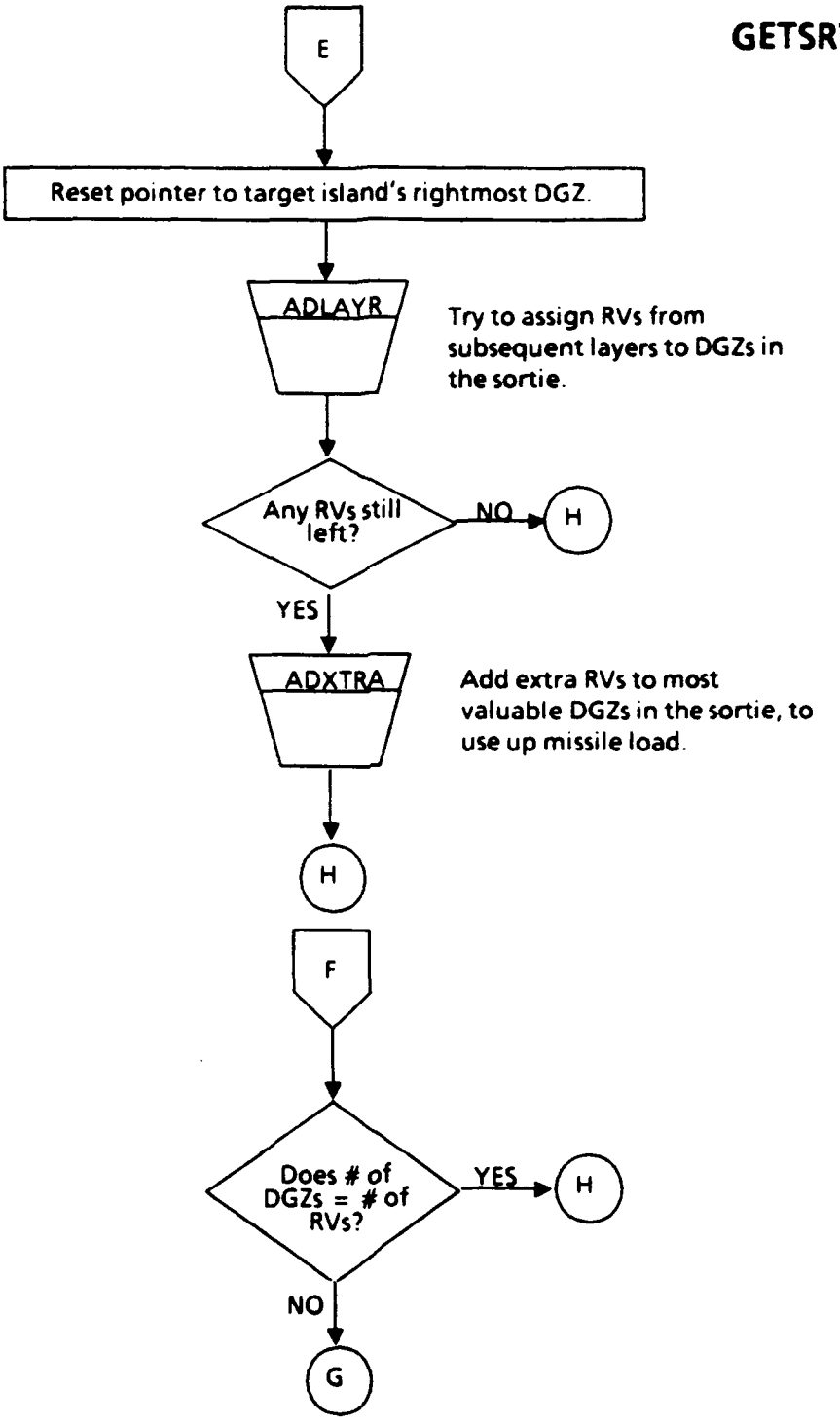



**FIGURE 5-8 SUBROUTINE GETSRT PROCESS FLOW (Continued)**

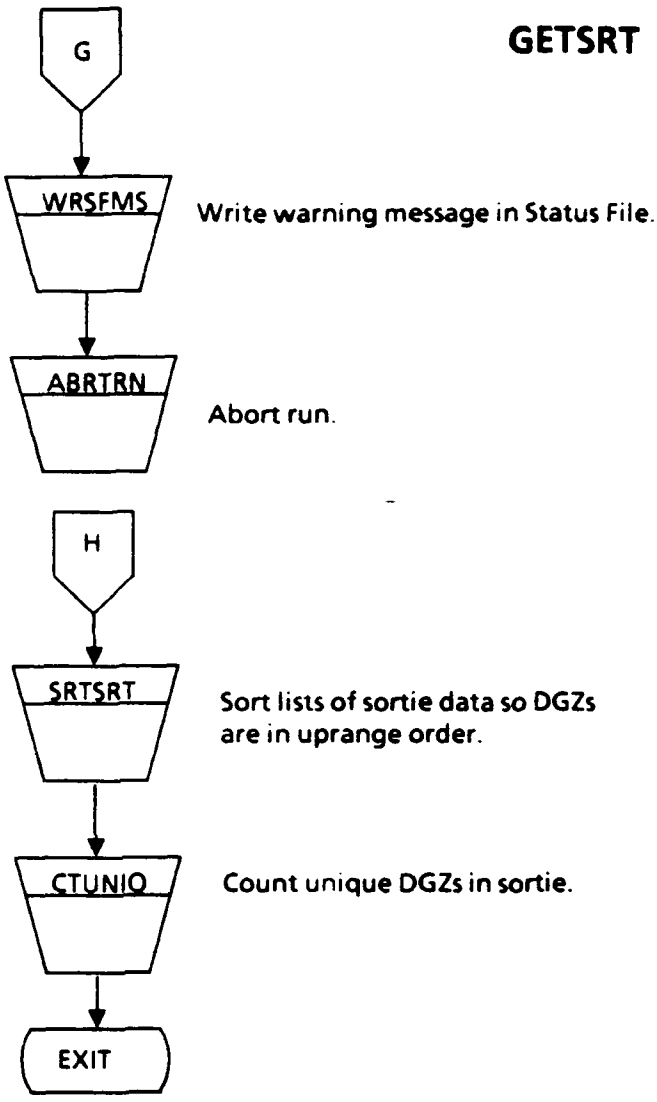



**FIGURE 5-8 SUBROUTINE GETSRT PROCESS FLOW (Concluded)**

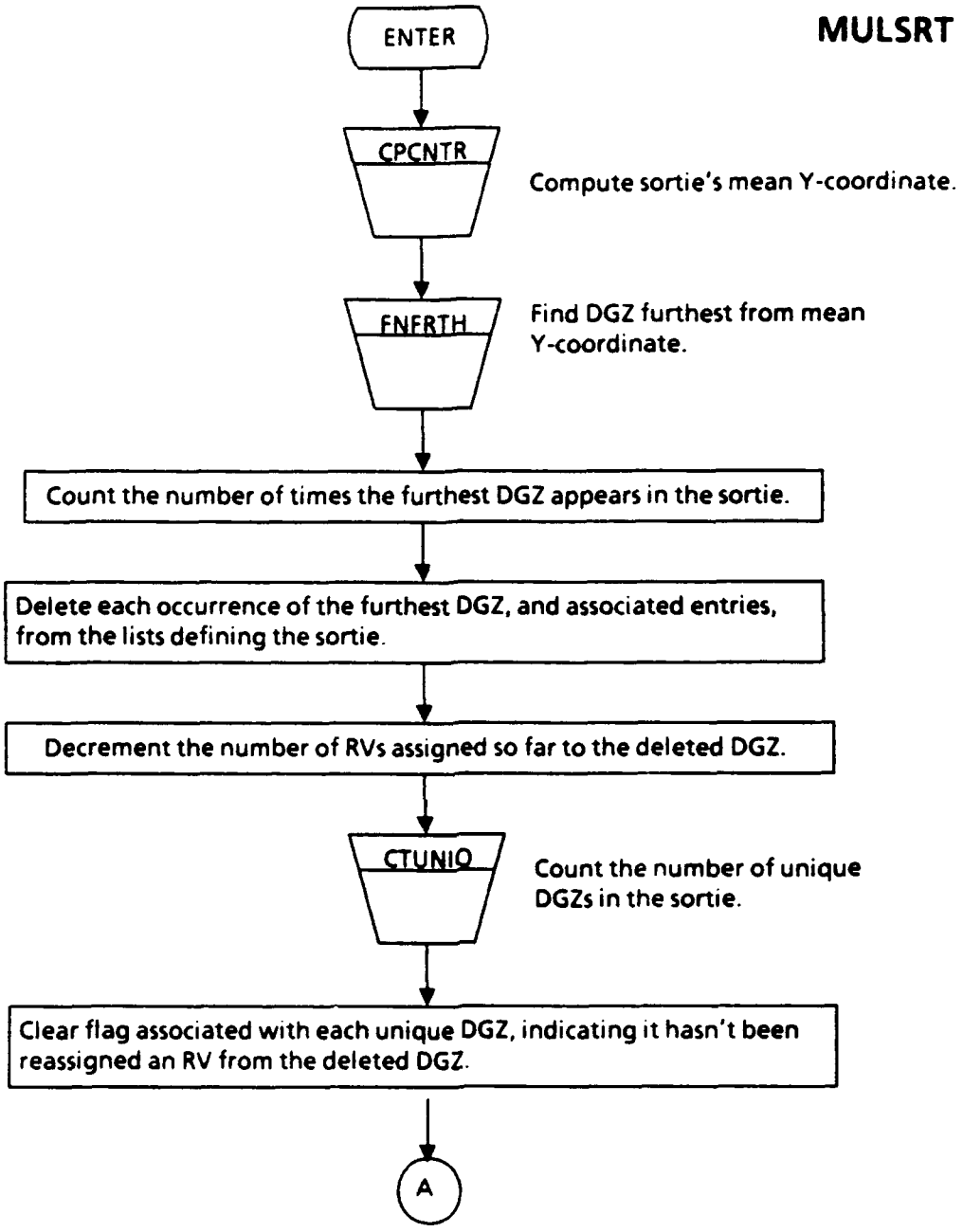



**FIGURE 5-9 SUBROUTINE MULSRT PROCESS FLOW**

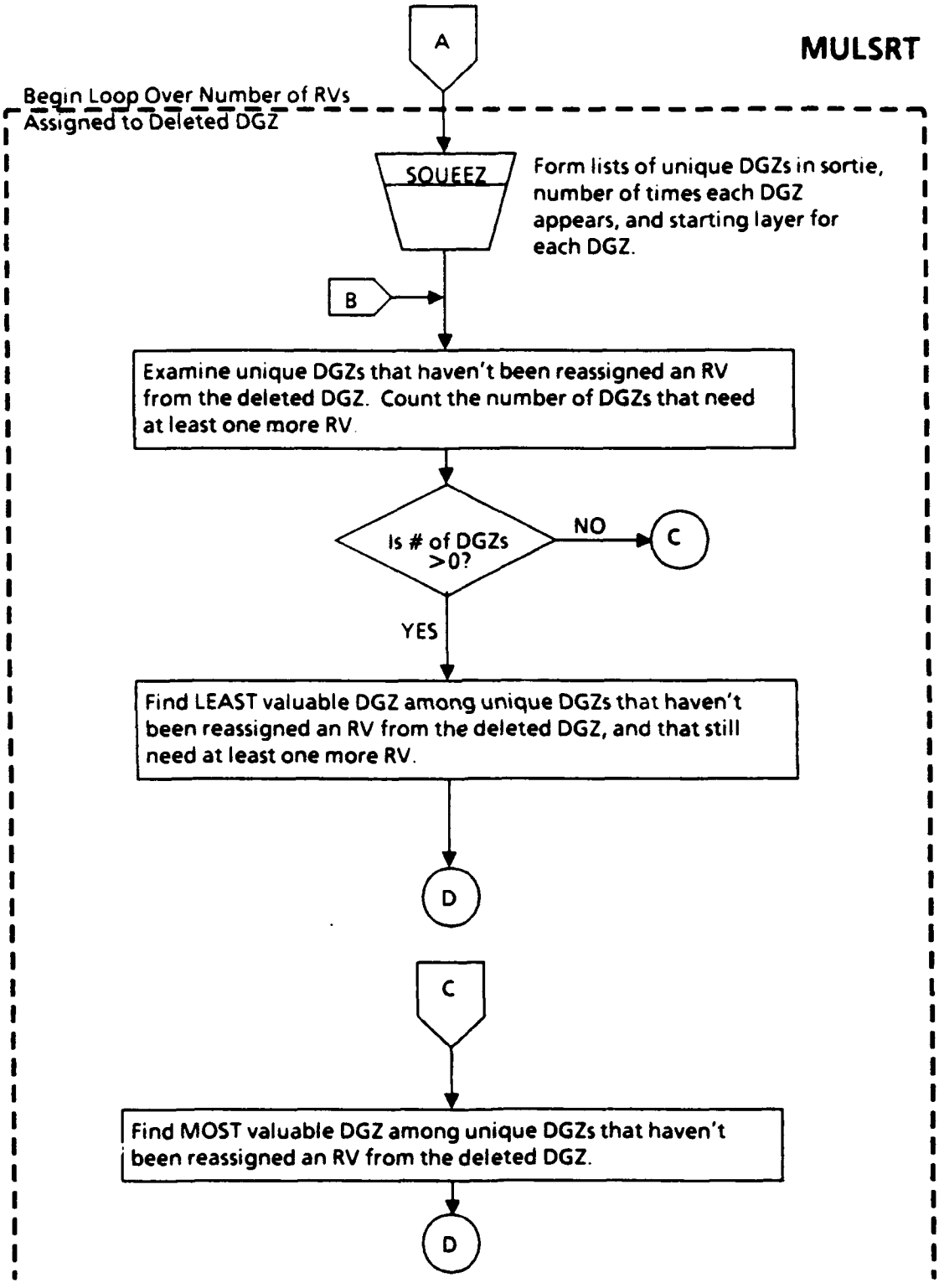



**FIGURE 5-9 SUBROUTINE MULSRT PROCESS FLOW (Continued)**

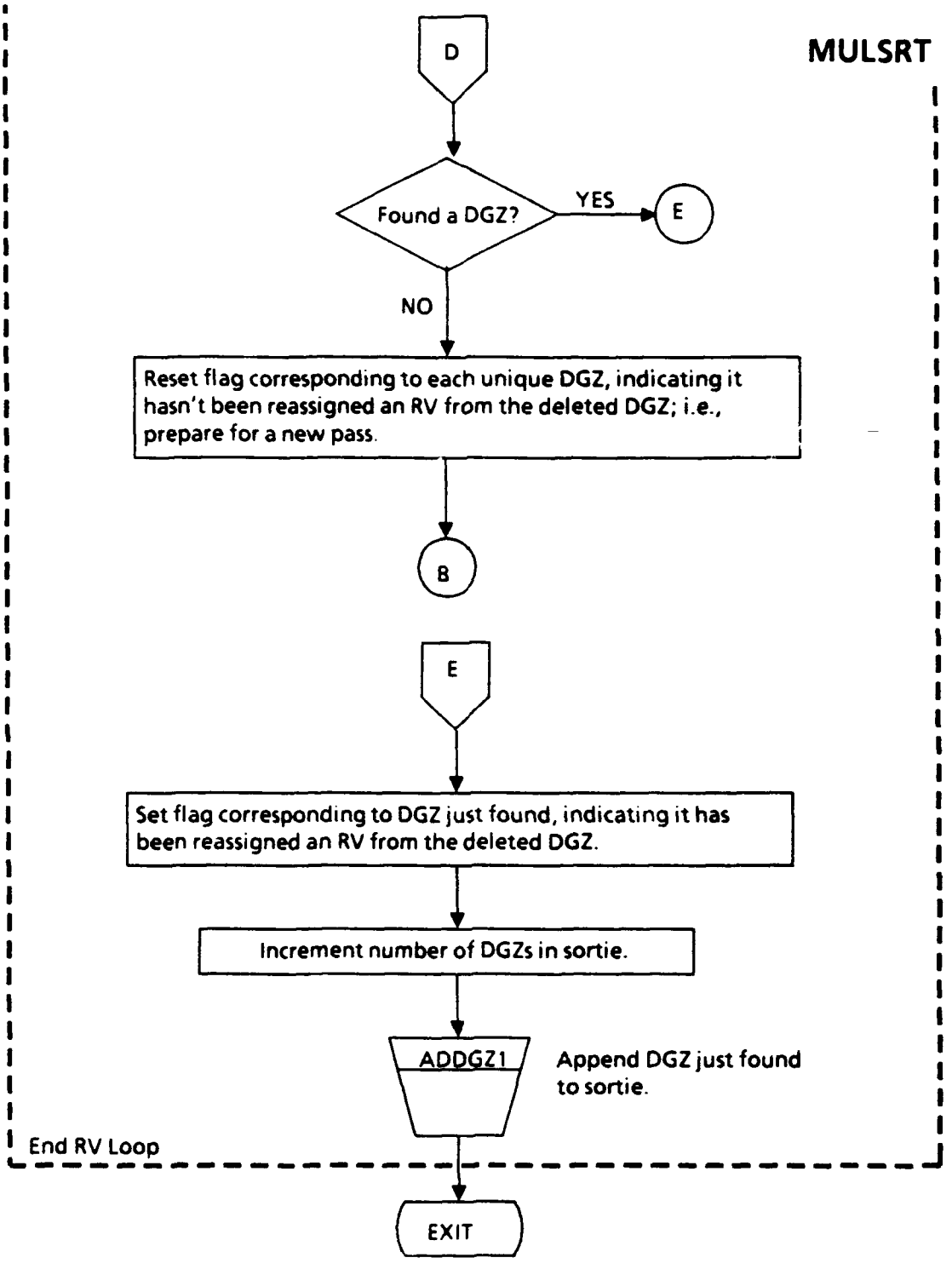



**FIGURE 5-9 SUBROUTINE MULSRT PROCESS FLOW (Concluded)**

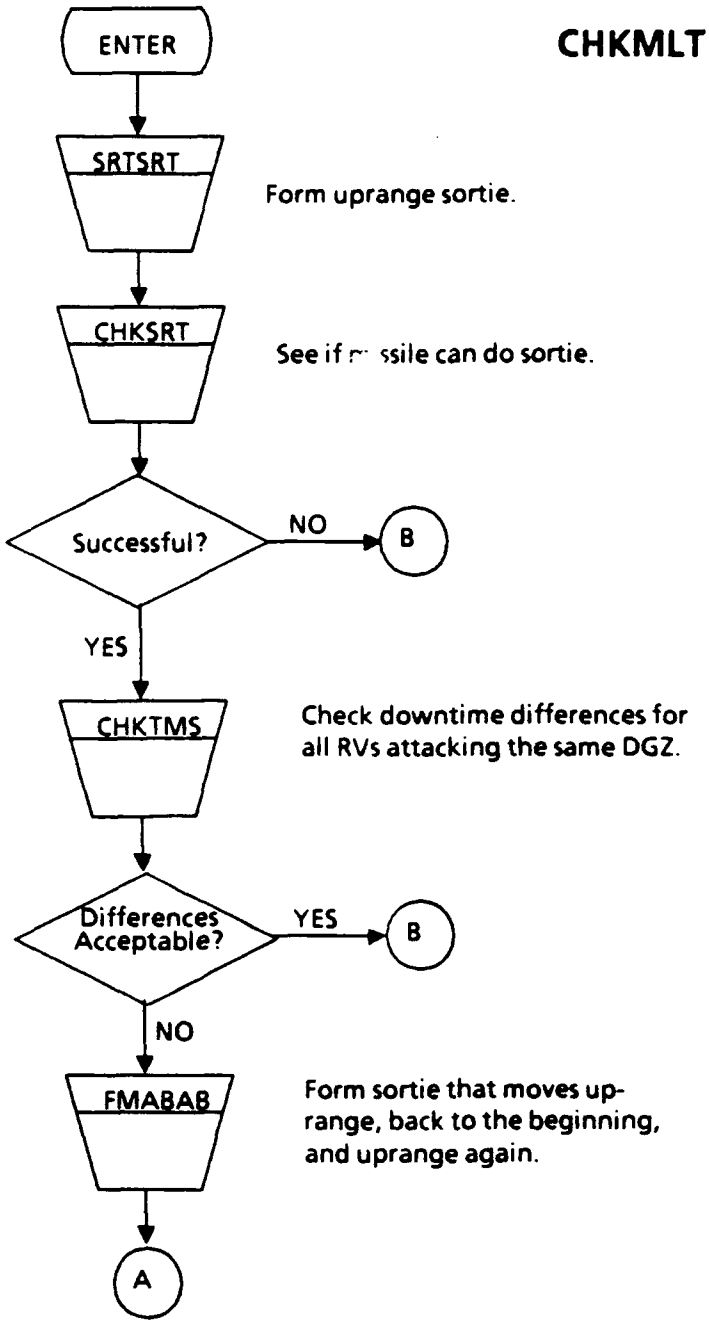



**FIGURE 5-10 SUBROUTINE CHKMLT PROCESS FLOW**

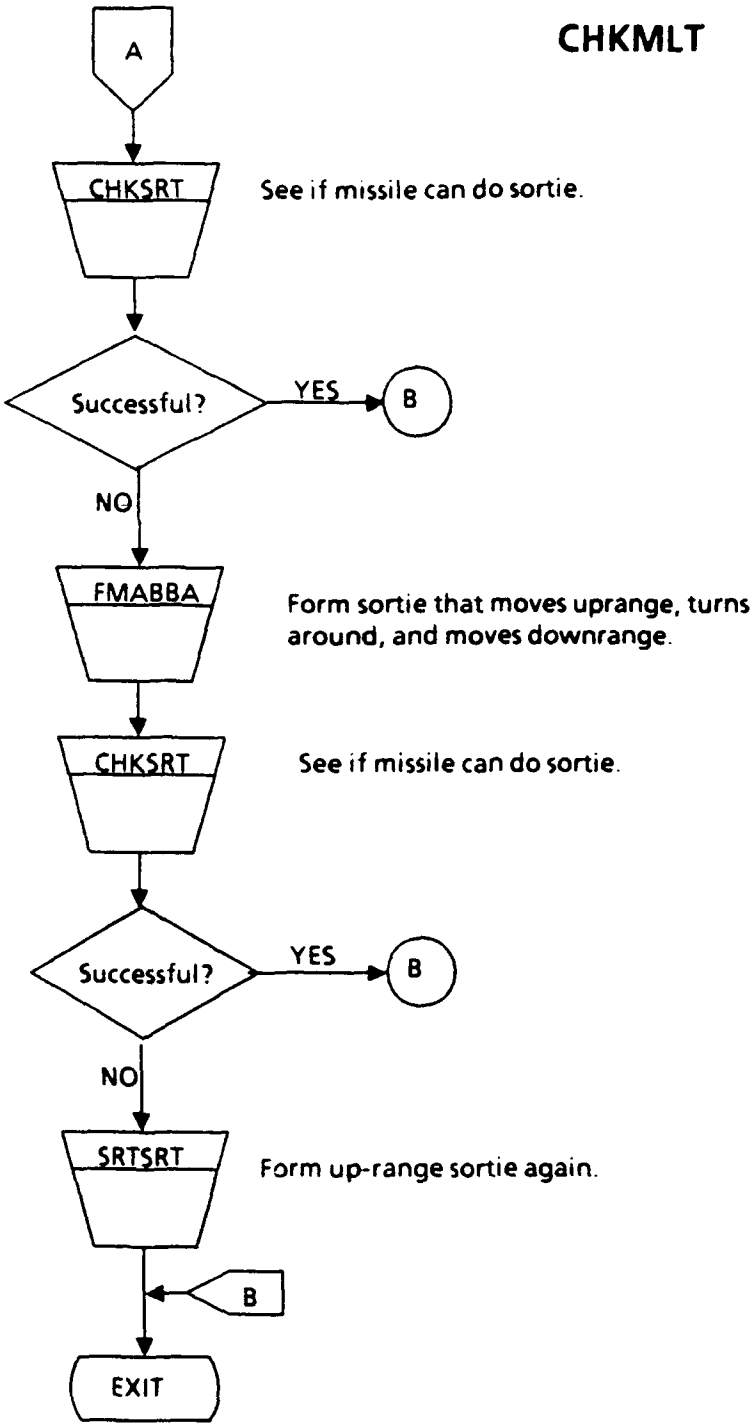



**FIGURE 5-10 SUBROUTINE CHKMLT PROCESS FLOW (Concluded)**

The second variation moves uprange, hitting each DGZ only once. The PBV is instructed to turn around and move downrange, hitting DGZs that get another RV. This is repeated if necessary.

If the missile can perform this variation, CHKMLT returns. If not, CHKMLT reforms the original uprange sortie, indicates it's a successful sortie even through times between layers are incorrect, and returns.

### 5.1.11 Subroutine CSTOUT

Subroutine CSTOUT (Figure 5-11) marks a DGZ as an outcast, so it is ignored in any subsequent sweeps. Within CSTOUT, DGZs are divided into two types: vertices of the convex polygon enclosing the target island, and interior points. CSTOUT scans the current sortie, looking for the vertex furthest from the target island's centroid, and the interior point farthest from the centroid. If CSTOUT finds a vertex, that DGZ is marked as the outcast; otherwise the DGZ that's the furthest interior point becomes the outcast.

This concludes the description of major subroutines called by ASR2T2. The last subroutine to be described by flowchart is ADJST5, which minimizes the span of downtimes for first-layer RVs.

### 5.1.12 Subroutine ADJST5

ADJST5 (Figure 5-12) examines all missiles used, finding the first-layer missiles. These are the missiles whose first RVs to arrive are the first to attack their respective DGZs. ADJST5 forms two lists corresponding to the first-layer missiles it finds. The first list contains launch times and is sorted in DESCENDING order. The second list contains times-of-flight for the first-layer RVs and is sorted in ASCENDING order. By pairing shorter times-of-flight with later launch times, and longer times-of-flight with earlier launch times, the span of downtimes is minimized.

### 5.1.13 Subroutine CHKSRT

CHKSRT provides the ALM interface to the TARGET CSC modules that are linked to, and used by, ALM to CHeck SoRTie feasibility.

This concludes the description of ALM's most important subroutines. Detailed descriptions of all subroutines can be found in Appendix C.

## 5.2 CONCEPTS AND MODELS

The most important concept underlying the ALM is shown in Figure 5-13, which illustrates the tangent-plane coordinate system used to specify the positions of DGZs and silos. This figure defines and illustrates the terms "left", "right", "downrange", etc., that have

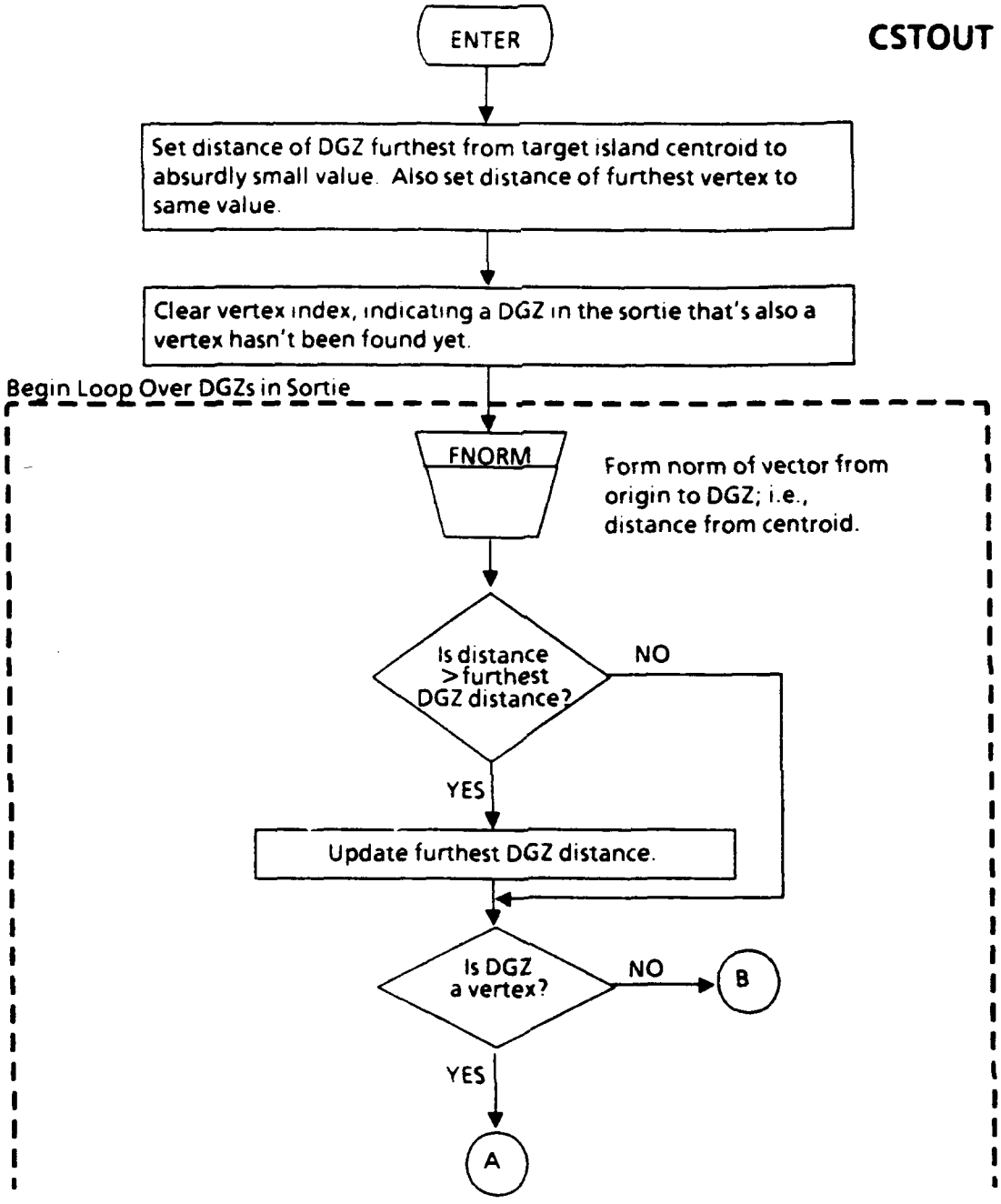



**FIGURE 5-11 SUBROUTINE CSTOUT PROCESS FLOW**

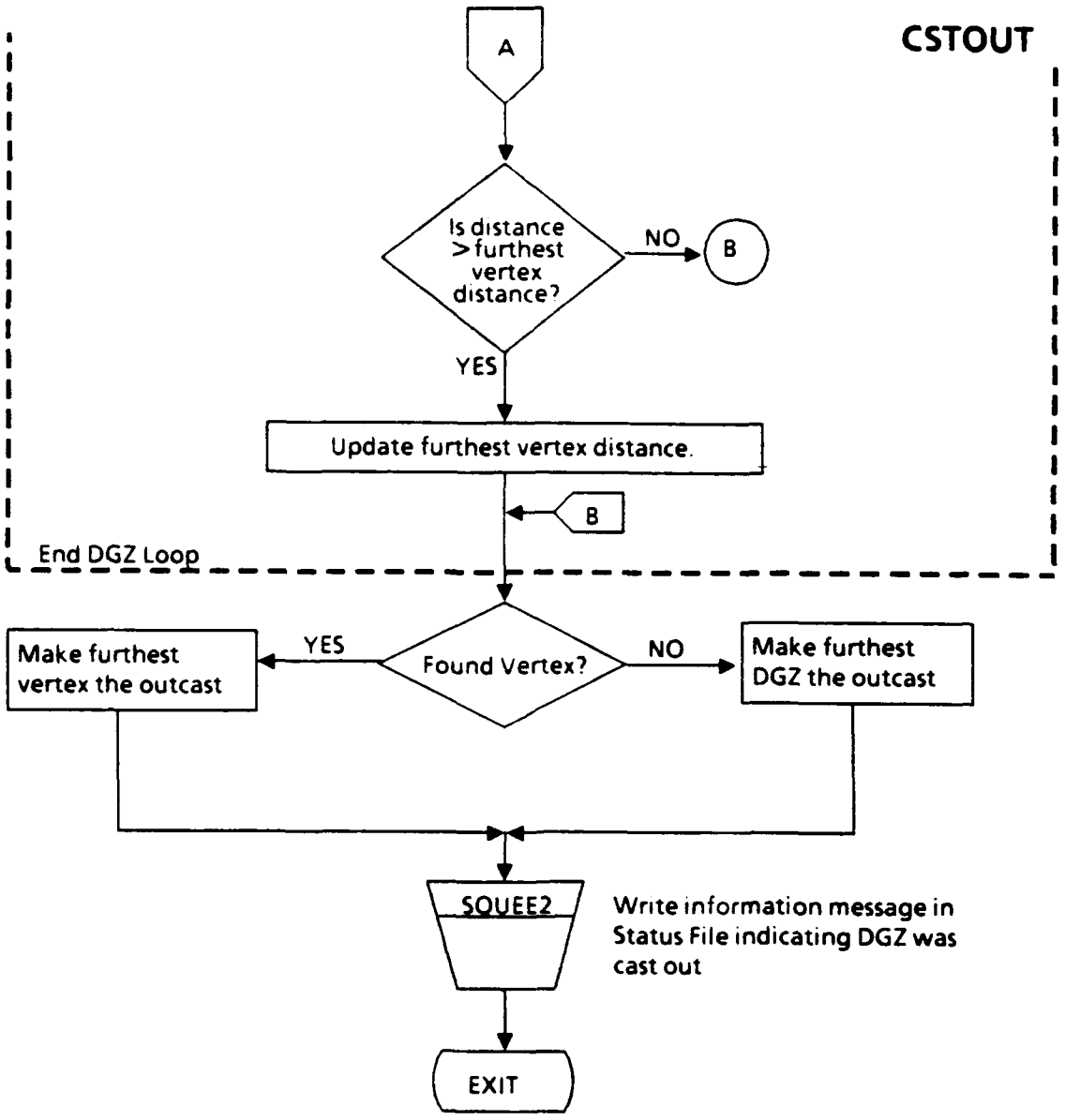



**FIGURE 5-11 SUBROUTINE CSTOUT PROCESS FLOW (Concluded)**

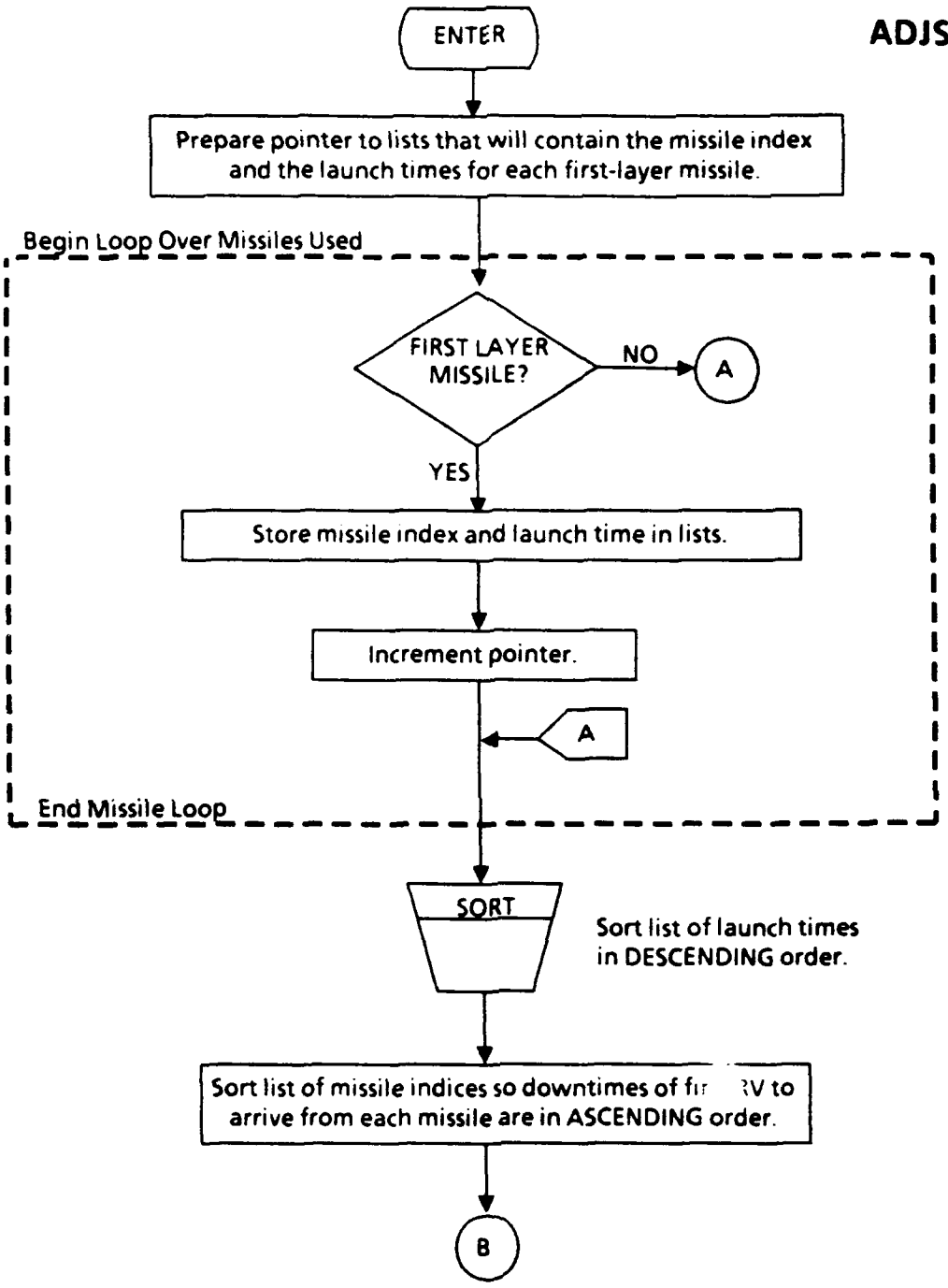



**FIGURE 5-12 SUBROUTINE ADJST5 PROCESS FLOW**

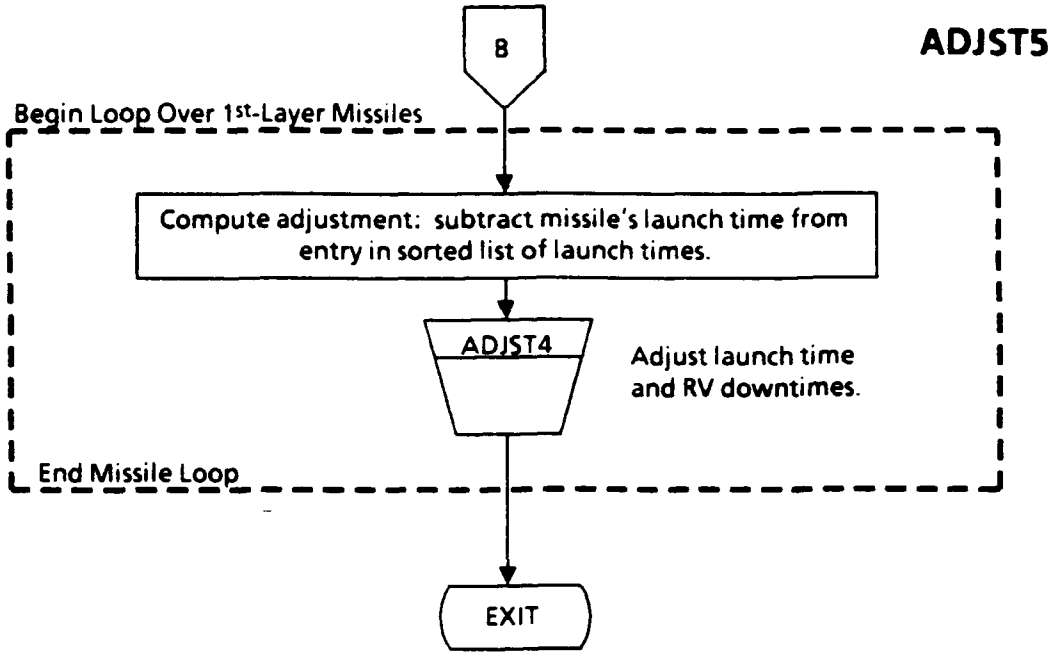



**FIGURE 5-12 SUBROUTINE ADJST5 PROCESS FLOW (Concluded)**

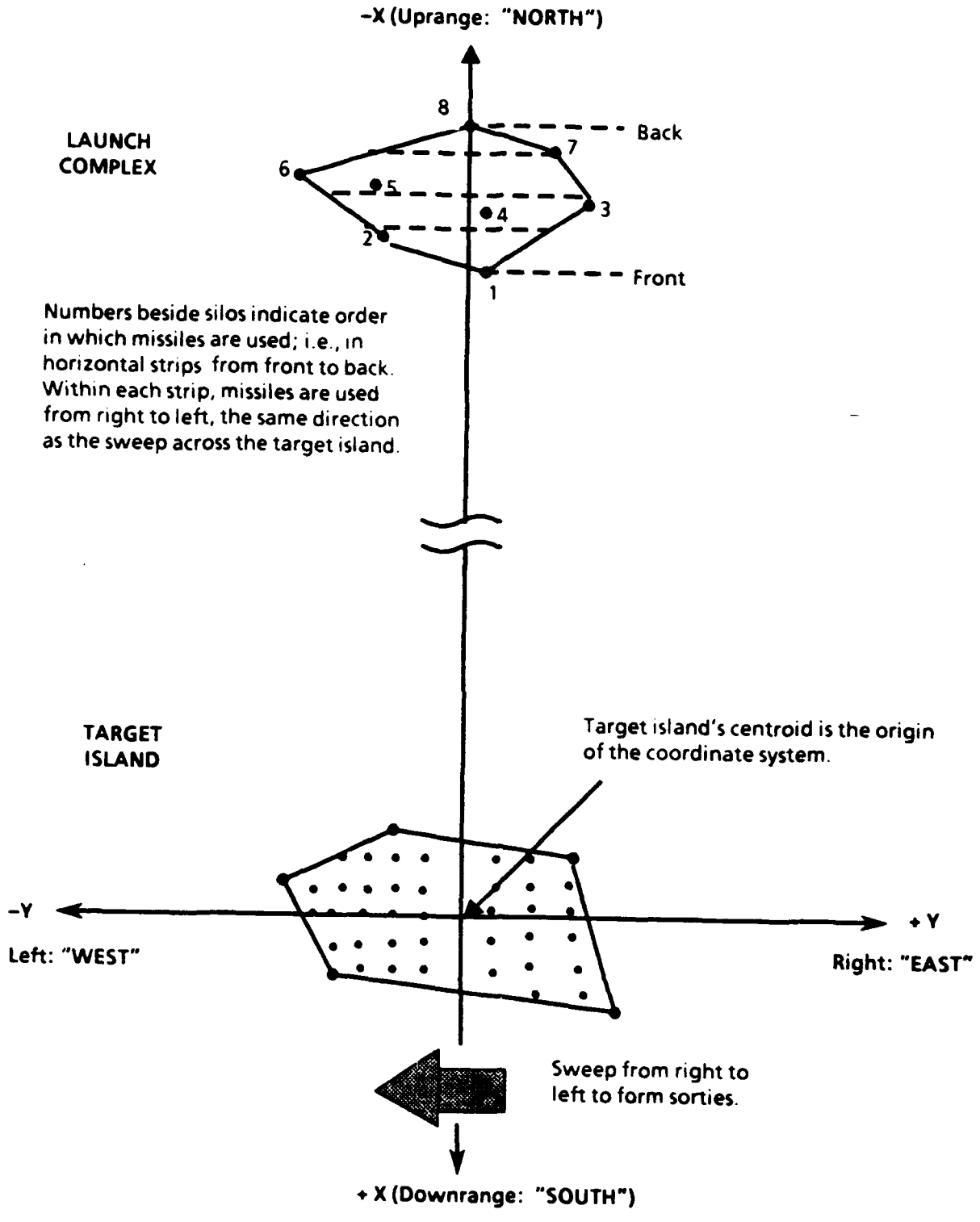



FIGURE 5-13 TANGENT-PLANE COORDINATE SYSTEM

"right", "downrange", etc., that have already been used in subroutine descriptions.

## 5.3 MODEL LIMITATIONS

There are limitations in the ALM CSC, resulting from a variety of considerations: speed of execution, available computer memory, and design compromises.

In ALM itself, the height of burst for the first DGZ in a sortie is considered to be the height of burst for all DGZs in the sortie, and it is used to pick the time between layers.

If Fratricide Constraints Files are used, ALM is indirectly subject to the model limitations of CAIN and NWEM. Because ALM calls modules in TARGET via the interface routine CHKSRT, ALM is subject to the model limitations of TARGET; i.e., limitations on modeling booster, PBV, and RV performance.

# SECTION 6
# OUTPUTS

All data generated by the ALM CSC are placed into three files: the ALM Status File (4.1); the Attack File (4.2); and the Sortie File (4.3). The ALM Status File informs the user as to the success or failure of an ALM execution and presents information, warning, and error messages generated during the execution of ALM. The Attack File contains launch time and downtime statistics, missile launch times, and RV data (downtimes, DGZs attacked, HOBs, and probabilities of attrition and fratricide). The Sortie File describes the kind of missile used in the attack and, for each missile, the DGZs attacked by the RVs on the missile. All three files may be reviewed and/or printed by the user through the MMI, by selecting the 'REVIEW A CASE" option in the MPS main menu. In fact, the Attack File can be reviewed in three ways: by missile, by RV, and by DGZ.

## 6.1 OUTPUT FILES

Figures 6-1 through 6-3 present the IOPTs for the three output files generated by the ALM CSC.

## 6.2 GRAPHICAL

Two maps are produced by the GRAPHS CSC from the ALM output files. The Attack File can be used to draw a Trajectory Projection Map, showing the flight of each missile superimposed on a map of the earth (see Figure 6-4). The Sortie File can be used to draw a Sortie Map (see Figure 6-5). These maps can be drawn on the video monitor, the color PaintJet, or the black and white LaserJet.

## 6.3 REPORTS

Three special reports can be produced by the MMI CSC from the ALM Attack File. The data in the file can be listed by DGZ, by missile, or by RV. Figures 6-6 through 6-8 show examples of these reports.

FILE NAME: ALM Status File ('case'.SF4)

FILE NUMBER: 4.1

DATE: 9/26/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS

| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | 1 | case name | 1X, A8 | - | - | - |
| | 2 | start time | 2X, A8 | - | - | - |
| | 3 | start date | 2X, A8 | - | - | - |
| 2-n<br>n = # of messages | 1 | message type: 0 = information<br>1 = warning<br>2 = success<br>3 = error<br>4 = continuation | I2 | [0, 4] | - | - |
| | 2 | time (message type ≠ 4) | 2X, A8 | - | - | - |
| | 3 | date (message type ≠ 4) | 2X, A8 | - | - | - |
| | 4 | subroutine name (message type ≠ 4) | 2X, A6 | - | - | - |
| | 5 | message text (message type ≠ 4)<br>(message type = 4) | 2X, A48<br>1X, A1, 20X, A48 | - | - | - |

FIGURE 6-1 ALM STATUS FILE IOPT

FILE NAME: Attack File ('case'.AF)

FILE NUMBER: 4.2

DATE: 9/26/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | - | HEADER RECORD | | | | |
| | 1 | Target island label | A24 | | | |
| | 2 | Target island elevation | F8.0 | [-10000, +30000] | ft | |
| | 3 | Missile type | A20 | [1, 100] | | |
| | 4 | $HOB_1$ | F8.1 | [0, 31072.3] | ft | |
| | 5 | $HOB_2$ | F8.1 | [0, 31072.3] | | |
| | 6 | Number of missiles attacking target island | I4 | [1, 10] | | |
| | 7 | Number of RVs attacking target island | I4 | [1, 250] | | |
| 2 | - | LAUNCH TIME STATISTICS | | | | |
| | 1 | Earliest launch time | F8.1 | [0, 86400] | secs | secs |
| | 2 | Latest launch time | F8.1 | [0, 86400] | secs | secs |
| | 3 | Launch duration (latest - earliest) | F8.1 | [0, 86400] | secs | secs |
| | 4 | Minimum time between launches | F8.1 | [0, 86400] | secs | secs |
| 3 | - | DOWN-TIME STATISTICS | | | | |
| | 1 | Earliest down-time | F8.1 | [0, 86400] | secs | secs |
| | 2 | Latest down-time | F8.1 | [0,86400] | secs | secs |
| | 3 | Laydown duration (latest - earliest) | F8.1 | [0,86400] | secs | secs |

**FIGURE 6-2 ATTACK FILE IOPT**

FILE NAME: Attack File ('case'.AF)

FILE NUMBER: 4.2

DATE: 9/26/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED ☒ SEQUENTIAL ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 4 | - | MISSILE RECORD | | | | |
| | 1 | Missile index | I4 | [1, 10] | | |
| | 2 | Launch point latitude (°N) | F8.3 | [-90, + 90] | deg | |
| | 3 | Launch point longitude (°E) | F8.3 | [-180, + 180] | deg | |
| | 4 | Launch time | F8.1 | [0, 86400] | secs | |
| 5 | | Attack record set: each set corresponds to an RV, and the number of records in the set depends on the RV's order of arrival. If i is the RV's ordinal number, then the number of records in the set, j, is given by:<br>j = (i + 14)/10 [integer division]<br>Each record is 80 characters long | | | | |
| | 1 | RV index | I4 | [1, 250] | | |
| | 2 | Down-time | F8.1 | [0, 86400] | secs | secs |
| | 3 | DGZ index descriptor | I6 | [1, 9999] | | |
| | 4 | DGZ type: 0 = installation<br>1 = grid point<br>2 = user input | I2 | [0, 2] | | |
| | 5 | DGZ latitude (°N) | F8.3 | [-90, + 90] | deg | |
| | 6 | DGZ longitude (°E) | F8.3 | [-180, + 180] | deg | |
| | 7 | HOB index | I2 | [1, 2] | | |

**FIGURE 6-2 ATTACK FILE IOPT (Continued)**

FILE NAME: Attack File ('case'.AF)

FILE NUMBER: 4.2

DATE: 9/26/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS



| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 5 | 8 | Missile number | I3 | [1, 10] | | |
| | 9 | RV number (order within missile) | I3 | [1, 25] | | |
| | 10 | Probability of attrition | F7.4 | [0.0, 1.0] | | |
| | 11 | Probability of fratricide by RV #1 | F7.4 | [0.0, 1.0] | | |
| | 12 | Probability of fratricide by RV #2 | F7.4 | [0.0, 1.0] | | |
| | 13 | Probability of fratricide by RV #3 | F7.4 | [0.0, 1.0] | | |
| | 14 | Probability of fratricide by RV #4 | F7.4 | [0.0, 1.0] | | |
| | | For RVs 1-5, one record is sufficient. All subsequent RVs require at least one more record in the set. Each record contains 10 probabilities of fratricide; i.e. the record is written using the format (10F7.4). | | | | |

**FIGURE 6-2 ATTACK FILE IOPT (Concluded)**

FILE NAME: Sortie File ('case'.SF)

FILE NUMBER: 4.3

DATE: 12/8/89

FILE TYPE

☒ FORMATTED ☐ UNFORMATTED

☒ SEQUENTIAL ☐ DIRECT ACCESS

| RECORD NO. | PARAMETER NO. | DESCRIPTION | FORMAT | BOUNDS | UNITS | |
|---|---|---|---|---|---|---|
| | | | | | ENG. | MET. |
| 1 | 1 | Target island label (name) | A24 | - | - | - |
| | 2 | Launch complex name | A24 | - | - | - |
| | 3 | Weapon name | A20 | - | - | - |
| | 4 | Number of PBVs/missile | I3 | [0, 10] | - | - |
| | 5 | Number of RVs/PBV | I3 | [0, 25] | - | - |
| | 6 | Number of RVs/missile | I3 | [1, 25] | - | - |
| | 7 | Number of missiles attacking island | I3 | [1, 10] | - | - |
| 2 | 1 | Missile number | I4 | [1, 10] | - | - |
| | 2 | RV number (order in missile) | I4 | [1, 25] | - | - |
| | 3 | DGZ index descriptor (number/label) | I6 | [1, 99999] | - | - |
| | 4 | DGZ type: 0 = installation<br>1 = rectangular grid point<br>2 = user input | I2 | [0, 2] | - | - |
| | 5 | DGZ latitude (°N) | F8.3 | [-90, +90] | deg | deg |
| | 6 | DGZ longitude (°E) | F8.3 | [-180, +180] | deg | deg |
| | 7 | Layer number [order of arrival at DGZ] | I4 | [1, 10] | - | - |

**FIGURE 6-3 SORTIE FILE IOPT**

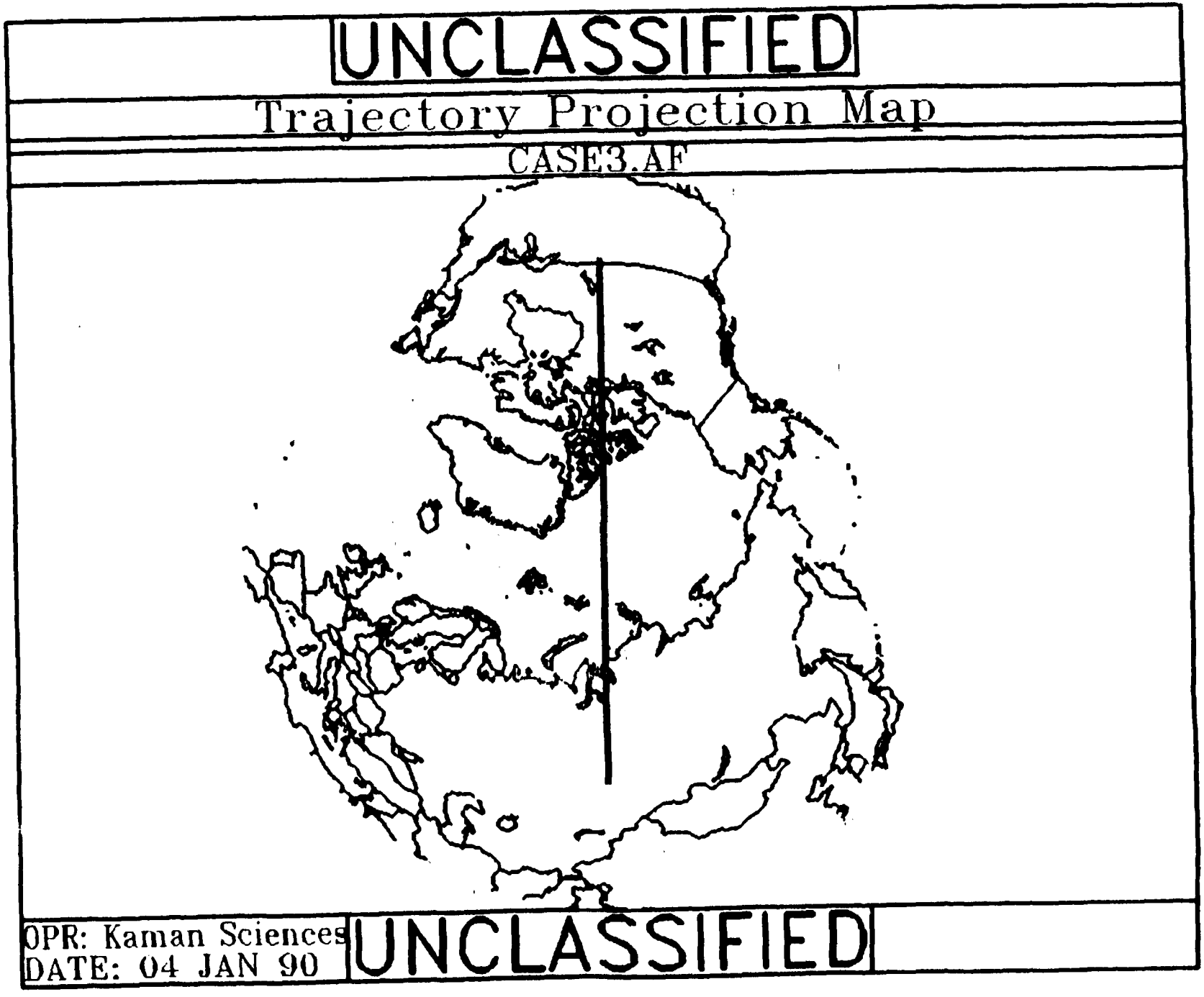



FIGURE 6-4 TRAJECTORY PROJECTION MAP

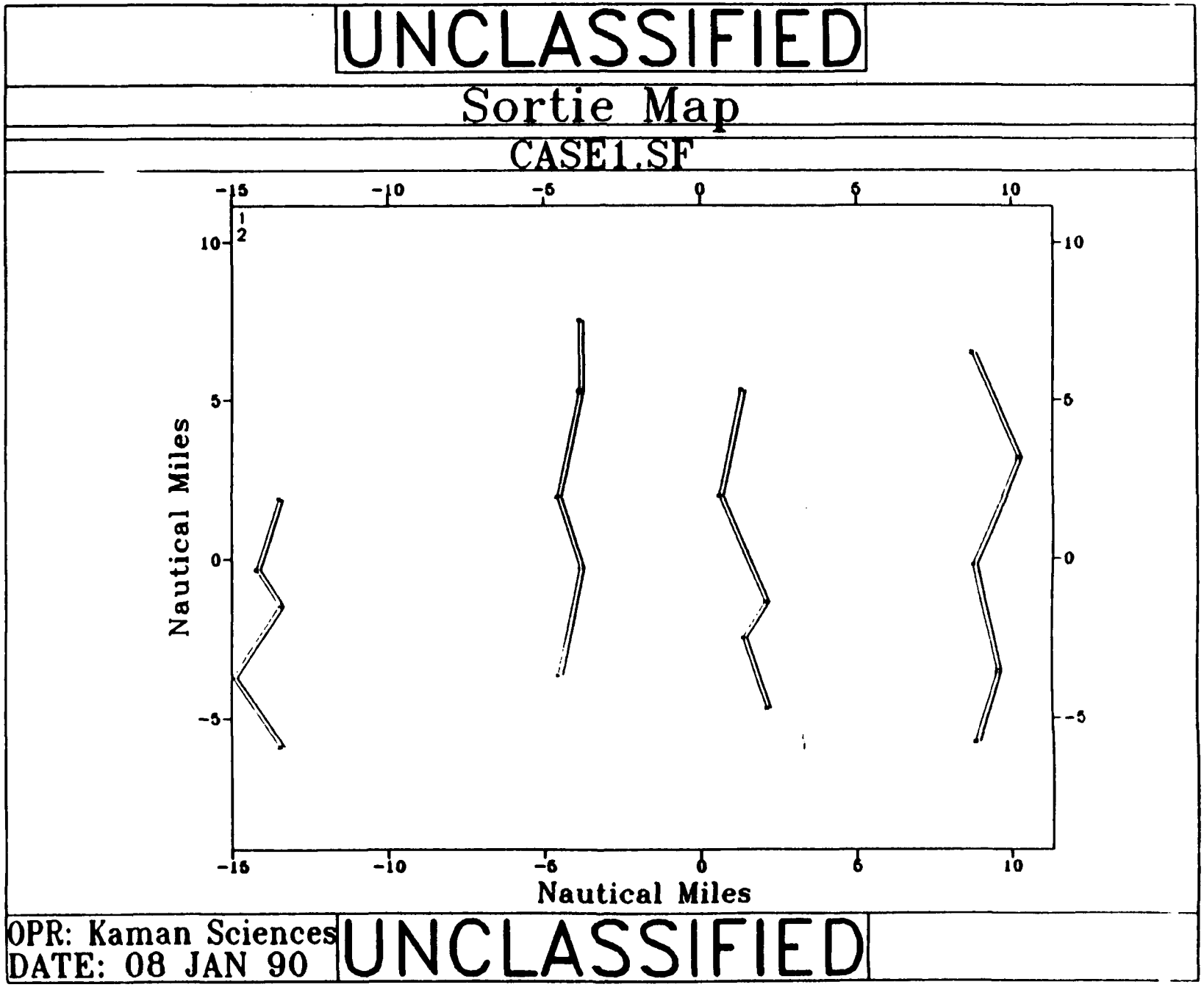



FIGURE 6-5 SC IE MAP

ATTACK DESCRIPTION BY DGZ

Case = CASE1C  Missile System = US-3RV
Number of RVs = 12  Number of Missiles = 4

| DGZ | RV | DOWNTIME | MISSILE | ORDER IN MISSILE |
|---|---|---|---|---|
| 1 | 1 | 1522.9 | 7 | 2 |
| 2 | 1 | 1520.9 | 7 | 1 |
| 3 | 1 | 1525.1 | 6 | 3 |
| 4 | 1 | 1520.9 | 6 | 1 |
| 5 | 1 | 1522.8 | 6 | 2 |
| 5 | 2 | 1526.9 | 4 | 2 |
| 5 | 3 | 1529.4 | 4 | 3 |
| 10 | 1 | 1520.7 | 5 | 1 |
| 12 | 1 | 1525.0 | 5 | 3 |
| 14 | 1 | 1524.8 | 4 | 1 |
| 44 | 1 | 1522.7 | 5 | 2 |
| 60 | 1 | 1525.3 | 7 | 3 |

**FIGURE 6-6 ATTACK DESCRIPTION BY DGZ**

ATTACK DESCRIPTION BY MISSILE

Case = SAMPLE1 Missile System = US-3RV
Number of RVs = 18 Number of Missiles = 6

| MISSILE | LAUNCH TIME | RV | DGZ | DOWNTIME | LAYER |
|---|---|---|---|---|---|
| 3 | .0 | 3 | 1 | 1809.5 | 1 |
| 3 | .0 | 2 | 4 | 1814.0 | 1 |
| 3 | .0 | 1 | 7 | 1818.1 | 1 |
| 2 | .2 | 3 | 2 | 1809.6 | 1 |
| 2 | .2 | 2 | 5 | 1814.1 | 1 |
| 2 | .2 | 1 | 8 | 1818.1 | 1 |
| 1 | .3 | 3 | 3 | 1809.6 | 1 |
| 1 | .3 | 2 | 6 | 1814.1 | 1 |
| 1 | .3 | 1 | 9 | 1818.1 | 1 |
| 6 | .5 | 3 | 1 | 1812.2 | 2 |
| 6 | .5 | 2 | 4 | 1816.7 | 2 |
| 6 | .5 | 1 | 7 | 1820.6 | 2 |
| 5 | .6 | 3 | 2 | 1812.2 | 2 |
| 5 | .6 | 2 | 5 | 1816.7 | 2 |
| 5 | .6 | 1 | 8 | 1820.6 | 2 |
| 4 | .7 | 3 | 3 | 1812.2 | 2 |
| 4 | .7 | 2 | 6 | 1816.7 | 2 |
| 4 | .7 | 1 | 9 | 1820.6 | 2 |

**FIGURE 6-7 ATTACK DESCRIPTION BY MISSILE**

ATTACK DESCRIPTION BY RV DOWNTIME

Case = CASE1C — Missile System = US-3RV
Number of RVs = 12 — Number of Missiles = 4

| RV | DOWNTIME | DGZ | MISSILE | ORDER IN MISSILE |
|---|---|---|---|---|
| 1 | 1520.7 | 10 | 5 | 1 |
| 2 | 1520.9 | 2 | 7 | 1 |
| 3 | 1520.9 | 4 | 6 | 1 |
| 4 | 1522.7 | 44 | 5 | 2 |
| 5 | 1522.8 | 5 | 6 | 2 |
| 6 | 1522.9 | 1 | 7 | 2 |
| 7 | 1524.8 | 14 | 4 | 1 |
| 8 | 1525.0 | 12 | 5 | 3 |
| 9 | 1525.1 | 3 | 6 | 3 |
| 10 | 1525.3 | 60 | 7 | 3 |
| 11 | 1526.9 | 5 | 4 | 2 |
| 12 | 1529.4 | 5 | 4 | 3 |

**FIGURE 6-8 ATTACK DESCRIPTION BY RV**

# SECTION 7
# ERROR PROCESSING

There are two major categories of error processing within the MPS. The first is input data error checking. The second is trapping and reporting run-time errors encountered during CSC execution.

Adhering to the user-friendly design of the MPS, the man-machine interface (MMI) checks numeric data for valid ranges when the data is entered. Thus, for every input data file created by the MPS, the MMI checks all of the numeric data. When entering data, the MMI displays the valid range for a value in the lower section of the panel. If a value is entered outside of this range, the MMI warns the user with a beep and displays the incorrect value, in red, directly below the range specification in the lower section of the panel. In addition to range checks, the MMI also performs some non-specification checks. Specifically, the MMI does not allow a case to be saved unless a valid case name and a missile type have been selected.

The MMI cannot check all of the input character data or combinations of numeric data that may cause errors during CSC execution. Thus, each CSC performs additional checks on the input data it receives. If errors are encountered, the CSC reports the error as described below.

When a CSC is executed, run-time errors can occur. Each CSC is designed to trap as many of these errors as possible, produce helpful information regarding the error, and write the error information to a status file. The EXEC CSC subsequently processes this file when the CSC terminates and displays any errors encountered during the CSC execution. Errors such as those generated during I/O operations, invalid input data, or errors produced by utility routines are processed via the status file mechanism.

Every CSC creates a status file whether or not it generates any run-time errors. The first record in the status file consists of the case name and the time and date the CSC was executed. Subsequent records contain status messages generated during the CSC execution. These messages are of four types or levels of severity: success, information, warning, and error. Each message consists of its type, the time and date it was generated, the module in which it was generated, and the message text. The

CSC must terminate the status file with either a success or error message.

In the following messages, integers printed within messages are represented by "[i]", "[j]", and "[k]"; real numbers by "[x]" and "[y]"; and strings by "sssss".

Note that the case name comes from the Control File's first record. If an error occurs while trying to read it, or if the case name is invalid, or if the Status File can't be opened, a paradox arises. The Status File is the only authorized medium for reporting errors, but an error has occurred before the Status File exists! In this situation, an error message is displayed on the VDT and the CSC stops. These special error messages are as follows:

| | |
|---|---|
| ALM: RCNTRL: | Error reading case name and description. Control File's 1st record is bad. ALM aborted. |
| ALM: RCNTRL: | End-of-file reading case name and description. Control File's 1st record is bad. ALM aborted. |
| ALM: RCNTRL: | Case name is blank. Can't even begin to run. |
| WRSFFR: | Bad logical unit! Must be > 0 and < 100, but not 5 or 6. Logical unit: [i] Case name: "sssss" CSC indicator: [j] |
| WRSFFR: | Bad case name! Length must be > 0 and < 9. Can't be blank. Logical unit: [i] Case name: "sssss" CSC indicator: [j] |
| WRSFFR: | Bad CSC identifier! Must be > 2 but < 11. Logical unit: [i] Case name: "sssss" CSC indicator: [j] |
| WRSFFR: | Couldn't open status file! |
| WRSFFR: | Couldn't write status file's first record! |

After CSC termination, the EXEC CSC reads the status file. Success and information messages are not errors and the EXEC does not process these

messages. If the EXEC does not encounter any warning or error messages, it automatically continues processing the CSC execution chain. If an error message is found, the EXEC interrupts the execution chain, displays the error message, and allows the user to continue the chain or abort and return to the MMI. The user may also view the entire status file before deciding whether to continue. Warning messages are processed in the same manner as errors. However, the user can direct the EXEC to trap or ignore warning messages before the CSC chain is executed. If warnings are ignored, they are treated like information messages. The status file is a formatted file and the user may also view or print the file outside of the EXEC/MMI.

Finally, the EXEC will interrupt the CSC execution chain if no status file is created by the CSC or if no termination (success or error) message is encountered. These cases might occur if a CSC aborts before it can create a status file or if a fatal error is not trapped and reported by the CSC.

Information, warning, and error messages specific to the ALM are listed in Figures 7-1, 7-2, and 7-3, respectively. Beyond an arbitrarily chosen point in its execution, if the ALM aborts, it dumps the contents of all COMMON blocks to the Status File. The name of each variable in COMMON and its value are written as an information message; e.g. LDURAT = 2.5470000E+01. These kinds of messages aren't shown in Figure 7-1.

In the following figures, integers printed within messages are represented by "[i]" through "[n]"; real numbers by "[x]" and "[y]"; and strings by "[s]" and "[t]".

| MODULE | MESSAGE |
|---|---|
| DPATTA | ***** COMMON block /ATTACK/ ***** |
| DPCNTR | ***** COMMON block /CNTROL/ ***** |
| DPLC | ***** COMMON block /LC/ ***** |
| DPOPSHNS | ***** COMMON block /OPSHNS/ ***** |
| DPSORTIE | ***** COMMON block /SORTIE/ ***** |
| DPTI | ***** COMMON block /TI/ ***** |
| DPWEAPON | ***** COMMON block /WEAPON/ ***** |
| ORDMSS | [i] missile(s) ([j] RV(s)) will be used. |
| ORDMSS | 1 missile was specified. |
| ORDMSS | [i] missiles were specified. |
| OUTCST | DGZ [i] is an outcast. |
| RALLOC | Record [i] in the RV Allocation File is an end-of-file. |
| RALLOC | Read RV Allocation File with no errors. |
| RBOOST | Read Booster File with no errors. |
| RCNTRL | [s] (where "[s]" represents the case description). |
| RCNTRL | Since no fratricide models were specified, times between layers will be read from the Control File's last record. |
| RCNTRL | Read Control File with no errors. |
| RCNTRL | Read Launch Location File with no errors. |
| RPBVFI | Read PBV File with no errors. |
| RRVFIL | Read RV File with no errors. |
| RSSAFE | Read sure-safe time(s) from Fratricide Constraint File(s) with no errors. |
| WSORTI | Wrote Sortie File successfully. |

**FIGURE 7-1 ALM INFORMATION MESSAGES**

| MODULE | MESSAGE |
|---|---|
| ADDGZ1 | Bad sortie index!<br>Index = [i].<br>Can't be < 1 or > [j].<br>Upper limit is number of RVs per weapon. |
| ADDGZ1 | Bad DGZ index!<br>Index = [i].<br>Can't be < 1 or > [j].<br>Upper limit is number of DGZs per target island. |
| ADDGZ1 | Too many RVs for one DGZ!<br>Number of RVs = [i].<br>Can't be > [j].<br>DGZ index = [k].<br>DGZ number = [m]. |
| ADDGZ2 | Bad sortie index!<br>Index = [i].<br>Can't be < 1 or > [j].<br>Upper limit is number of RVs per weapon. |
| ADDGZ2 | Bad unique DGZ index!<br>Index = [i].<br>Can't be < 1 or > [j].<br>Upper limit is number of DGZs per target island. |
| ADJST1 | Can't find previous layer's RV!<br>Current layer = [i].<br>Previous layer = [j].<br>Missile = [k].<br>DGZ = [m]. |
| ADLAYR | Bad index for sortie lists!<br>Index = [i].<br>Must be > 1 but not > [j]. |
| ADLAYR | Bad number of RVs!<br>Number of RVs = [i].<br>Must be > 0 but not > [j]. |
| ADXTRA | Bad index for sortie lists!<br>Index = [i].<br>Must be > 1 but not > [j]. |
| ADXTRA | Bad number of RVs!<br>Number of RVs = [i].<br>Must be > 0 but not > [j]. |
| ALM | Can't reach farthest DGZ from silo closest to target island! |

**FIGURE 7-2 ALM WARNING MESSAGES**

| MODULE | MESSAGE |
|---|---|
| ABRTRN | Aborting Allocation CSC (ALM).<br>Contents of COMMON blocks follow. |
| FNPREV | Couldn't find previous RV attacking DGZ!<br>DGZ index = [i].<br>Previous layer = [i]. |
| GETSRT | Number of DGZs in sortie doesn't equal number of RVs on missile!<br>Number of DGZs = [i].<br>Number of RVs = [j]. |
| GTBLYR | Bad layer number!<br>Layer number = [i]. |
| MOPUPC | Couldn't find sortie to repeat! |
| MOVLFT | Couldn't find matching DGZ!<br>Leftmost DGZ in sortie = [i] |
| ORDMSS | Ran out of missiles!<br>NMSLLC = [i]<br>FIRST = [j]<br>LAST = [k]<br>NDNEED = [m]<br>NRVSWP = [n] |
| PREPAR | Bad number of missiles used!<br>Number of missiles used = [i]<br>Should be at least 1. |
| PREPAR | Bad number of RVs!<br>Number of RVs used = [i]<br>Should be at least 1. |
| RALLOC | Bad target island elevation.<br>Can't be < 10000 feet or > 30000 feet. |
| RALLOC | Couldn"t open RV Allocation File!<br>File = "[s]". |
| RALLOC | Target island label is blank. |
| RALLOC | Error reading RV Allocation File's 1st record. |
| RALLOC | End-of-file reading RV Allocation File's 1st record. |
| RALLOC | Error reading RV Allocation File's 2nd record. |
| RALLOC | End-of-file reading RV Allocation File's 2nd record. |
| RALLOC | Error reading RV Allocation File' 3rd record. |
| RALLOC | End-of-file reading RV Allocation File's 3rd record. |

**FIGURE 7-2 ALM WARNING MESSAGES (Continued)**

| MODULE | MESSAGE |
|---|---|
| RALLOC | Error reading RV Allocation File's [i]th record. |
| RALLOC | End-of-file reading RV Allocation File's [i]th record. |
| RALLOC | Bad height-of-burst (HOB)!<br>Can't be < 0 or > 32000 feet. |
| RALLOC | Bad number of missiles attacking target island.<br>Number of missiles = [i].<br>Can't be < 1 or > [j]. |
| RALLOC | Bad number of RVs allocated.<br>Number of RVs allocated = [i].<br>Can't be < 1 or > [j]. |
| RALLOC | Number of RVs isn't an integer multiple of number of missiles!<br>Number of RVs = [i].<br>Number of missiles = [j]. |
| RALLOC | Error skipping RV Allocation File's<br>[i]th record.<br>Error ignored. |
| RALLOC | End-of-file skipping RV Allocation File's<br>[i]th record. |
| RALLOC | Bad index descriptor for DGZ [i].<br>Index descriptor = [j].<br>Can't be < 1 or > 9999. |
| RALLOC | Bad DGZ type for DGZ [i].<br>DGZ type = [j].<br>Can't be < 0 or > 2. |
| RALLOC | Bad latitude for DGZ [i].<br>Latitude = [x].<br>Can't be < -90 or > + 90 degrees. |
| RALLOC | Bad longitude for DGZ [i].<br>Longitude = [x].<br>Can't be < -180 or > + 180 degrees. |
| RALLOC | Bad height-of-burst index for DGZ [i].<br>Index = [j].<br>Can't be < 1 or > [k]. |
| RALLOC | Bad number of RVs for DGZ [i].<br>Number of RVs = [j].<br>Can't be < 1 or > [k]. |
| RALLOC | Record [i] (DGZ [j]) had 1 error. |

**FIGURE 7-2 ALM WARNING MESSAGES (Continued)**

| MODULE | MESSAGE |
|---|---|
| RALLOC | Record [i] (DGZ [j]) had [i] errors. |
| RALLOC | No allocation records! |
| RALLOC | Number of Fratricide Constraint Files is less than number of heights-of-burst.<br>Number of files = [i]<br>Number of HOBs = [j] |
| RALLOC | Number of Fratricide Constraint Files is greater than number of heights-of-burst.<br>Number of files = [i]<br>Number of HOBs = [j] |
| RBOOST | Couldn't open Booster File!<br>File = "[s]". |
| RBOOST | Error reading Booster File.<br>File = "[s]".<br>Record number = [i] |
| RBOOST | End-of-file reading Booster File.<br>File = "[s]".<br>Record number = [i] |
| RBOOST | Inconsistent number of RVs per weapon!<br>Number of missiles in attack = [i]<br>Number of RVs allocated = [j]<br>Quotient = [k] RVs per missile But according to Booster File, there are [m] RVs per missile! |
| RCNTRL | Error reading name of RV Allocation File!<br>Control File's 2nd record is bad. |
| RCNTRL | End-of-file reading name of RV Allocation File!<br>Control File's 2nd record is bad. |
| RCNTRL | Name of RV Allocation File is blank!<br>Control File's 2nd record is bad. |
| RCNTRL | Error reading name of Launch Location File!<br>Control File's 3rd record is bad. |
| RCNTRL | End-of-file reading name of Launch Location File!<br>Control File's 3rd record is bad. |
| RCNTRL | Name of Launch Location File is blank!<br>Control File's 3rd record is bad. |
| RCNTRL | Missile name is blank!<br>Control File's 3rd record is bad. |
| RCNTRL | Error reading Booster File record number and name.<br>Control File's 4th record is bad. |

**FIGURE 7-2 ALM WARNING MESSAGES (Continued)**

| MODULE | MESSAGE |
|---|---|
| RCNTRL | End-of-file reading Booster File Data and name.<br>Control File's 4th record is bad. |
| RCNTRL | Name of Launch Location File is blank!<br>Control File's 3rd record is bad. |
| RCNTRL | Bad record number for booster file!<br>Record number = [i]. |
| RCNTRL | Booster File Name is blank! |
| RCNTRL | Error reading PBV File record number and name!<br>Control File's 5th record is bad. |
| RCNTRL | End-of-file reading PBV File record number and name!<br>Control File's 5th record is bad. |
| RCNTRL | Bad record number for PBV file!<br>Record number = [i]. |
| RCNTRL | PBV File Name is blank! |
| RCNTRL | Error reading RV File record number and name!<br>Control File's 6th record is bad. |
| RCNTRL | End-of-file reading RV File record number and name!<br>Control File's 6th record is bad. |
| RCNTRL | Bad record number for RV file!<br>Record number = [i]. |
| RCNTRL | RV File Name is blank! |
| RCNTRL | Error reading number of fratricide models and path!<br>Control File's 7th record is bad. |
| RCNTRL | End-of-file reading number of fratricide models and path!<br>Control File's 7th record is bad. |
| RCNTRL | Bad number of fratricide models!<br>Number of models = [i]. |
| RCNTRL | Path for fratricide models is blank! |
| RCNTRL | Error reading names of fratricide constraint files!<br>Control File's 8th record is bad. |
| RCNTRL | End-of-file reading names of fratricide constraint files!<br>Control File's 8th record is bad. |
| RCNTRL | 1st Constraint File name is blank. |
| RCNTRL | 2nd Constraint File name is blank. |
| RCNTRL | 3rd Constraint File name is blank. |

**FIGURE 7-2 ALM WARNING MESSAGES (Continued)**

| MODULE | MESSAGE |
|---|---|
| RCNTRL | 4th Constraint File name is blank. |
| RCNTRL | Error reading Attack File name!<br>Control File's 9th record is bad. |
| RCNTRL | End-of-file reading Attack File name!<br>Control File's 9th record is bad. |
| RCNTRL | Attack File name is blank. |
| RCNTRL | Error reading Sortie File name!<br>Control File's 10th record is bad. |
| RCNTRL | End-of-file reading Sortie File name!<br>Control File's 10th record is bad. |
| RCNTRL | Sortie File name is blank. |
| RCNTRL | Error reading run options!<br>Control File's 11th record is bad. |
| RCNTRL | End-of-file reading run options!<br>Control File's 11th record is bad. |
| RCNTRL | Fraction of fuel for range extension is bad!<br>Fraction = [x] |
| RCNTRL | You've allocated [x] of the fuel for range extension but the missile has no PBV!<br>Range(s) won't be extended. |
| RCNTRL | Reentry angle is bad!<br>Angle = [x] |
| RCNTRL | Error reading times between layers!<br>Control File's 12th record is bad. |
| RCNTRL | End-of-file reading times between layers!<br>Control File's 12th record is bad. |
| RCNTRL | 1st time between layers is bad. |
| RCNTRL | 2nd time between layers is bad. |
| RCNTRL | 3rd time between layers is bad. |
| RCNTRL | [i]th time between layers is bad. |
| RCNTRL | No times between layers on Control File's last record. At this point, however, the number of layers is unknown. If there's only one layer in the attack, there's no problem. |
| RCNTRL | Control File contained [i] errors. |
| RLAUNC | Couldn't open Launch Location File!<br>File = "[s]". |

**FIGURE 7-2 ALM WARNING MESSAGES (Continued)**

| MODULE | MESSAGE |
|---|---|
| RLAUNC | Error reading Launch Location File's 1st record. |
| RLAUNC | End-of-file reading Launch Location File's 1st record. |
| RLAUNC | Launch complex label is blank. |
| RLAUNC | Bad number of missile silos.<br>Number of missile silos = [i]<br>Can't be > [j]. |
| RLAUNC | Bad number of missile silos.<br>Number of missile silos = [i]<br>Can't be < [j]; i.e. number of missiles specified in Attack File. |
| RLAUNC | Error reading Launch Location File's 2nd record. |
| RLAUNC | End-of-file reading Launch Location File's 2nd record. |
| RLAUNC | Bad latitude for co-located silos.<br>Latitude = [x].<br>Can't be < -90 or > +90 degrees. |
| RLAUNC | Bad longitude for co-located silos.<br>Longitude = [x].<br>Can't be < -180 or > +180 degrees. |
| RLAUNC | Bad elevation for co-located silos.<br>Elevation (altitude) = [x].<br>Can't be < -10000 feet or > 30000 feet. |
| RLAUNC | Error reading Launch Location File's 3rd record. |
| RLAUNC | End-of-file reading Launch Location File's 3rd record. |
| RLAUNC | Error reading Launch Location File's [i]th record. |
| RLAUNC | End-of-file reading Launch Location File's [i]th record. |
| RLAUNC | Bad missile index for [i]th missile.<br>Missile index = [i].<br>Can't be > [j]. |
| RLAUNC | Bad latitude for [i]th missile.<br>Latitude = [x].<br>Can't be < -90 or > +90 degrees. |
| RLAUNC | Bad longitude for [i]th missile.<br>Longitude = [x].<br>Can't be < -180 or > +180 degrees. |
| RLAUNC | Bad elevation for [i]th missile.<br>Elevation (altitude) = [x].<br>Can't be < -10000 feet or > 30000 feet. |

**FIGURE 7-2 ALM WARNING MESSAGES (Continued)**

| MODULE | MESSAGE |
|---|---|
| RLAUNC | Bad flight letter for [i]th missile.<br>Flight letter = "[s]".<br>Must be A-Z. |
| RLAUNC | Record [i] (silo [j]) has 1 error. |
| RLAUNC | Record [i] (silo [j]) has [k] errors. |
| RLAUNC | Launch Location File had 1 error. |
| RLAUNC | Launch Location File had [i] errors. |
| RPBVFI | Couldn't open PBV File!<br>File = "[s]". |
| RPBVFI | Error reading PBV File!<br>File = "[s]".<br>Record number = [j] |
| RPBVFI | End-of-file reading PBV File!<br>File = "[s]".<br>Record number = [j] |
| RRVFIL | Couldn't open RV File!<br>File = "[s]". |
| RRVFIL | Error reading RV File!<br>File = "[s]".<br>Record number = [j] |
| RRVFIL | End-of-file reading RV File!<br>File = "[s]".<br>Record number = [j] |
| RSSAFE | Couldn't open Fratricide Constraint File!<br>Path = "[s]".<br>File = "[t]". |
| RSSAFE | Error reading Fratricide Constraint File's third record!<br>Path = "[s]".<br>File = "[t]". |
| RSSAFE | End-of-file reading Fratricide Constraint File's third record!<br>Path = "[s]".<br>File = "[t]". |
| RSSAFE | Bad yield in Fratricide Constraint File.<br>Yield = [x]<br>Path = "[s]".<br>File = "[t]". |

**FIGURE 7-2 ALM WARNING MESSAGES (Continued)**

| MODULE | MESSAGE |
|---|---|
| RSSAFE | Unequal scaled heights-of-burst in Fratricide Constraint File!<br>SHOB1 = [x]<br>SHOB2 = [y]<br>Path = "[s]".<br>File = "[t]". |
| RSSAFE | Error reading Fratricide Constraint File's 6th record!<br>Path = "[s]".<br>File = "[t]". |
| RSSAFE | End-of-file reading Fratricide Constraint File's 6th record!<br>Path = "[s]".<br>File = "[t]". |
| RSSAFE | Bad sure-safe time in Fratricide Constraint File.<br>Yield = [x]<br>Path = "[s]".<br>File = "[t]". |
| RSSAFE | Did not find sure-safe time for [i]th height-of-burst.<br>Height-of-burst = [x] feet. |
| WSORTI | Couldn't open Sortie File! |
| WSORTI | Error writing Sortie File's 1st record. |
| WSORTI | Error writing Sortie File record for Missile [i], RV [j]. |

**FIGURE 7-2 ALM WARNING MESSAGES (Concluded)**

| MODULE | MESSAGE |
|---|---|
| ALM | ALM aborted. |
| ABRTRN | ALM aborted! |
| RALLOC | ALM aborted. |
| RBOOST | ALM aborted. |
| RCNTRL | ALM aborted. |
| RLAUNC | ALM aborted. |
| RPBVFI | ALM aborted. |
| RRVFIL | ALM aborted. |
| WSORTI | ALM aborted. |

**FIGURE 7-3 ALM ERROR MESSAGES**

# SECTION 8
# SAMPLE PROBLEMS

The sample problem presented in this section demonstrates the development and execution of a laydown case in which the only function enabled is assignment. This example illustrates how the assignment function is used to generate a timed laydown of ballistic missile forces on a target set that fulfills an input RV allocation and that satisfies missile system operational constraints, and user-specified targeting requirements and objectives.

## 8.1 DESCRIPTION

The scenario implemented in this laydown case is defined in Figure 8-1. In an effort to keep this example as self contained as possible, the following input options were selected:

- Time Between Layers. A user-specified, uniform (constant) time between layers of 2.5 seconds was prescribed. Specification of this time obviates the need for one or more CAIN-produced, input Fratricide Constraints Files.
- RV Allocation. A user-specified, uniform 2-on-1 attack was employed. Specification of this RV allocation obviates the need for a SOADA-produced, input RV Allocation File.

As a result of these choices, the only input required to exercise this sample that cannot be specified during case definition is the characterization of the missile system to be employed. When developing a laydown case, the user must specify the weapon system to be employed, but the specification of the missile model corresponding to that system, must have been previously developed under the MPS "BUILD DATA SETS" option.

Figures 8-2 through 8-7 show the Man-Machine Interface (MMI) panels used to develop the missile data set for the "US-3RV" missile system employed in this example. The process begins by selecting the "BUILD DATA SETS" option on the MPS main menu.

## 8.2 INPUTS

Figures 8-8 through 8-13 show the MMI panels used to define the sample laydown case. The process actually begins by slecting the "DEFENSE TIMED LAYDOWN CASE" option on the MPS main menu shown earlier in Figure 8-2. Up to Figure 8-10, the panels are shown in the order they would appear when running the MPS. The order in which the remaining panels

| | |
|---|---|
| **Missile System** | |
| No. of Missiles | 6 |
| No. of RVs/Missile | 3 |
| Launch Location Centroid | 55°N, 80°E, 1000 ft elevation |
| Min. Time Between Launches | 0. seconds |
| Minimum Exoatmospheric Spacing | 90,000 ft. |
| **Target Array** | |
| No. of DGZs | 9 |
| Target Location Centroid | 48°N, -100°E, 3000 ft elevation |
| RV Allocation | Uniform, 2 RVs/DGZ |
| **Assignment Options** | |
| Time Between Layers | User-specified, Uniform, 2.5 sec |
| Earth Model | Rotating |
| Max. Fraction of PBV Fuel for Range Extension | .25 |
| Reentry Angle | 24° |
| Attack Timing Objective | Synchronize Arrival Times |
| RV Spacing | Space a Minimum Distance |
| Height of Burst | 2381.1 ft. |

**FIGURE 8-1 SCENARIO SPECIFICATION**

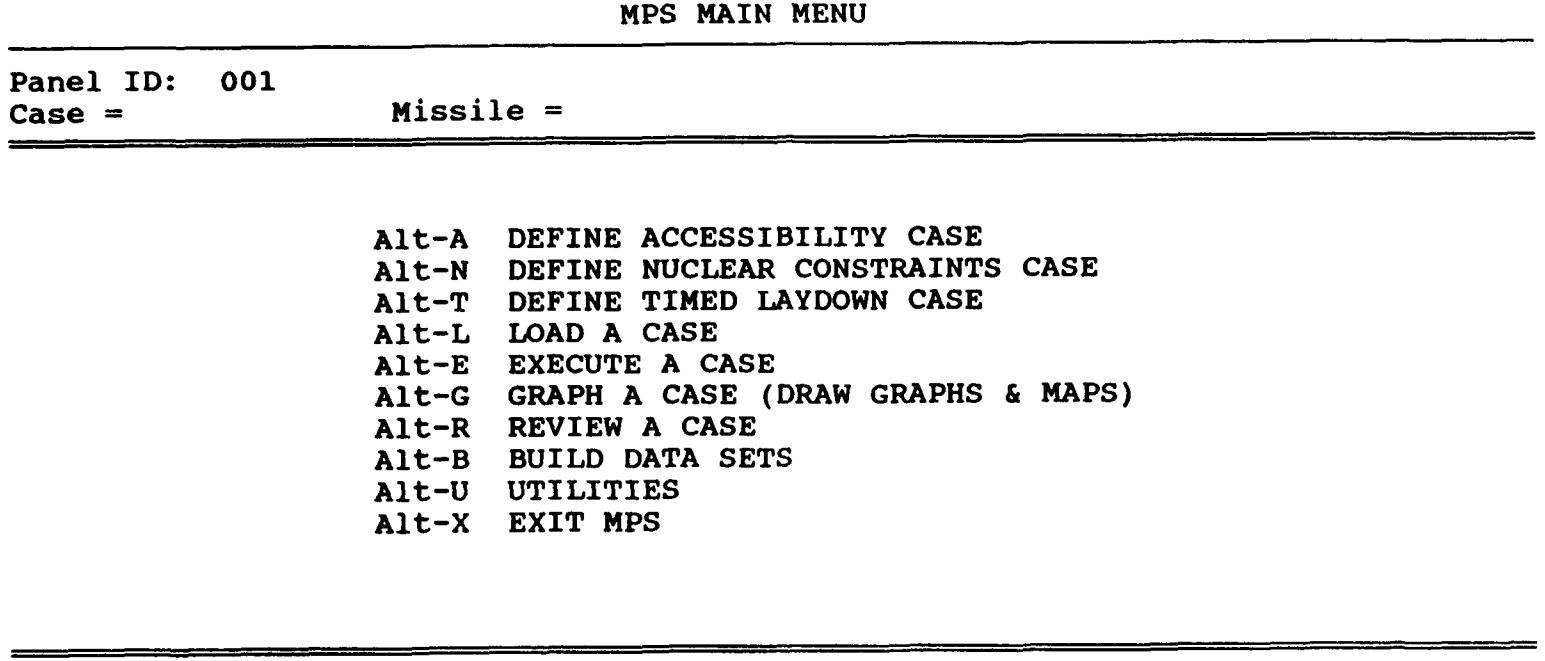



**FIGURE 8-2 MPS MAIN MENU**

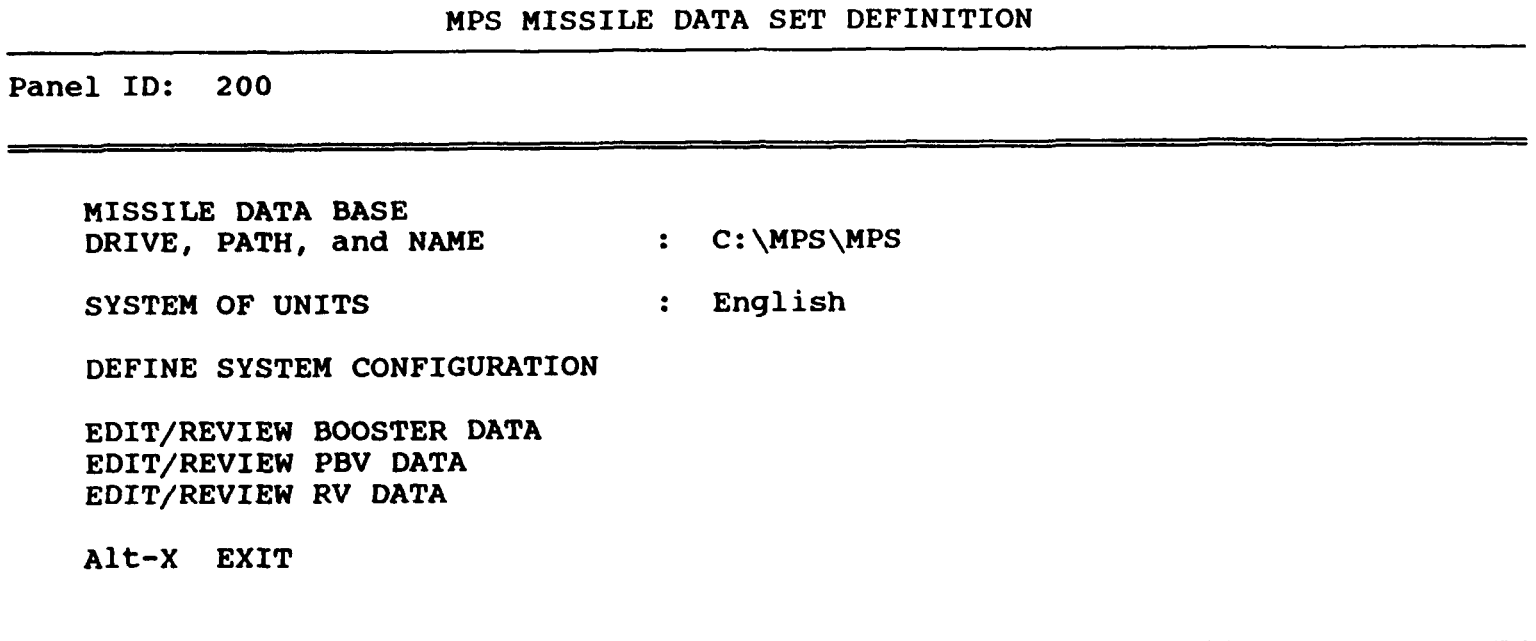



**FIGURE 8-3 MPS MISSILE DATA SET DEFINITION**

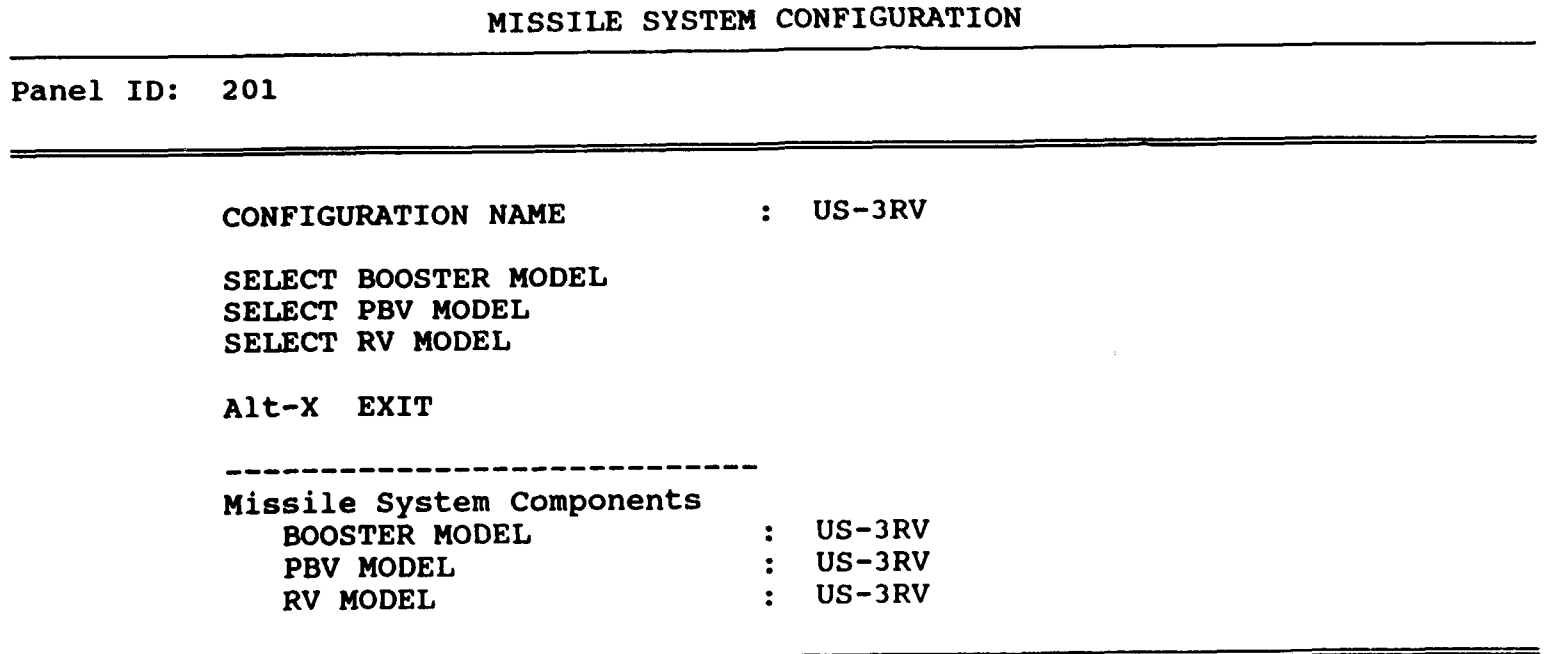



**FIGURE 8-4 MISSILE SYSTEM CONFIGURATION**

```
                              BOOSTER DEFINITION

Panel ID:  203

        BOOSTER NAME                          :  US-3RV
        NUMBER OF STAGES                      :        3
        ENERGY MANAGEMENT STAGE               :        3
        NUMBER OF DRAG vs. MACH ENTRIES       :        0
        NUMBER OF PBVs ON BOOSTER             :        1
        NUMBER OF RVs ON BOOSTER              :        3
        GEMS ?                                :   F
        LAUNCH RELIABILITY                    :     .800000
        MISSILE/PBV RELIABILITY               :     .960000
        MINIMUM TIME BETWEEN LAUNCHES         :     .000000      sec

        SPECIFY BOOSTER DATA
        SPECIFY BOOSTER DRAG vs. MACH DATA

        Alt-X  EXIT AND SAVE DATA

  Character Value:  Maximum number of characters =     20

  Esc-Abort  F1-Panel Help  F2-Field Help  F3-Key Table  F8-Main Menu  F10-GOTO
```

**FIGURE 8-5 BOOSTER DATA PANELS**

BOOSTER SPECIFICATION DATA

Panel ID: 204

Page Number = 1/ 3

| | | | |
|---|---|---|---|
| WEIGHT - STAGE 1 | : | 107000. | lb |
| WEIGHT - STAGE 2 | : | 61000.0 | lb |
| WEIGHT - STAGE 3 | : | 18011.0 | lb |
| WEIGHT - STAGE 4 | : | .000000 | lb |
| VACUUM SPECIFIC IMPULSE - STAGE 1 | : | 300.000 | sec |
| VACUUM SPECIFIC IMPULSE - STAGE 2 | : | 300.000 | sec |
| VACUUM SPECIFIC IMPULSE - STAGE 3 | : | 300.400 | sec |
| VACUUM SPECIFIC IMPULSE - STAGE 4 | : | .000000 | sec |
| VACUUM THRUST - STAGE 1 | : | 487000. | lb |
| VACUUM THRUST - STAGE 2 | : | 276000. | lb |
| VACUUM THRUST - STAGE 3 | : | 68335.0 | lb |
| VACUUM THRUST - STAGE 4 | : | .000000 | lb |

Real Value: Minimum = .000000 Maximum = .100000E+07

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F8-Main Menu F10-GOTO

**FIGURE 8-5 BOOSTER DATA PANELS (Continued)**

BOOSTER SPECIFICATION DATA

Panel ID: 204

Page Number = 2/ 3

| | | | |
|---|---|---|---|
| REFERENCE AREA - STAGE 1 | : | 40.2000 | $ft^2$ |
| REFERENCE AREA - STAGE 2 | : | 40.2000 | $ft^2$ |
| REFERENCE AREA - STAGE 3 | : | 40.2000 | $ft^2$ |
| REFERENCE AREA - STAGE 4 | : | .000000 | $ft^2$ |
| NOZZLE EXIT AREA - STAGE 1 | : | 18.2000 | $ft^2$ |
| NOZZLE EXIT AREA - STAGE 2 | : | 35.7000 | $ft^2$ |
| NOZZLE EXIT AREA - STAGE 3 | : | 26.3000 | $ft^2$ |
| NOZZLE EXIT AREA - STAGE 4 | : | .000000 | $ft^2$ |
| MAXIMUM BURNTIME - STAGE 1 | : | 56.6000 | sec |
| MAXIMUM BURNTIME - STAGE 2 | : | 58.2000 | sec |
| MAXIMUM BURNTIME - STAGE 3 | : | 69.7000 | sec |
| MAXIMUM BURNTIME - STAGE 4 | : | .000000 | sec |

Real Value: Minimum = .000000 Maximum = 1000.00

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F8-Main Menu F10-GOTO

**FIGURE 8-5 BOOSTER DATA PANELS (Continued)**

BOOSTER SPECIFICATION DATA

Panel ID: 204

Page Number = 3/ 3

| | | | |
|---|---|---|---|
| INTER-STAGE COAST DURATION - STAGE 1 | : | .000000 | sec |
| INTER-STAGE COAST DURATION - STAGE 2 | : | .000000 | sec |
| INTER-STAGE COAST DURATION - STAGE 3 | : | .000000 | sec |
| INTER-STAGE COAST DURATION - STAGE 4 | : | .000000 | sec |
| MIN. BURNTIME - ENERGY MANAGEMENT STAGE | : | .000000 | sec |
| VERTICAL LAUNCH DURATION | : | 2.00000 | sec |
| TIME OF FLIGHT TO INITIATE ATTITUDE HOLD | : | 56.6000 | sec |
| SHROUD JETTISON TIME | : | 56.6000 | sec |
| SHROUD WEIGHT | : | 745.000 | lb |
| LAUNCH VELOCITY GAIN | : | 112.000 | ft/sec |
| BOOST PHASE ACCELERATION LIMIT | : | 10.0000 | g |

Alt-X EXIT

Real Value: Minimum = .000000 Maximum = 100.000

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F8-Main Menu F10-GOTO

**FIGURE 8-5 BOOSTER DATA PANELS (Continued)**

BOOSTER DRAG COEFFICIENT vs. MACH NUMBER TABLE

Panel ID: 205

Number Selected = 0/ 10

| Entry | MACH NUMBER | DRAG COEFFICIENT |
|---|---|---|
| 1 | .0 | .00000 |
| 2 | .0 | .00000 |
| 3 | .0 | .00000 |
| 4 | .0 | .00000 |
| 5 | .0 | .00000 |
| 6 | .0 | .00000 |
| 7 | .0 | .00000 |
| 8 | .0 | .00000 |
| 9 | .0 | .00000 |
| 10 | .0 | .00000 |

Real Value: Minimum = .000000 Maximum = 100.000

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F10-GOTO RETURN-Exit

**FIGURE 8-5 BOOSTER DATA PANELS (Concluded)**

MISSILE DATA SET - PBV DEFINITION

Panel ID: 206

| | | |
|---|---|---|
| PBV NAME | : | US-3RV |
| PBV TYPE | : | Pusher |
| PBV MOUNTING OPTION | : | Nose Forward |
| MAIN ENGINE OPTION | : | On-Off Capability |
| NUMBER OF OBJECTS ON PBV | : | 3 |
| NUMBER OF OBJECTS ON TIER | : | 0 |
| RV DEPLOYMENT MODE | : | First-Off Last-In |

SPECIFY PBV DATA
SPECIFY OBJECT DATA

Alt-X EXIT AND SAVE DATA

Character Value: Maximum number of characters = 20

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F8-Main Menu F10-GOTO



**FIGURE 8-6 PBV DATA PANELS**

MISSILE DATA SET - PBV DATA

Panel ID: 207

Page Number = 1/ 2

| | | | |
|---|---|---|---|
| STRUCTURE (DRY) WEIGHT | : | 2435.00 | lb |
| USABLE FUEL WEIGHT | : | 1394.00 | lb |
| AXIAL ENGINE THRUST (Low if dual levels) | : | 165.600 | lb |
| SPECIFIC IMPULSE | : | 306.600 | sec |
| AVERAGE ROTATION DURATION | : | 4.80000 | sec |
| AVERAGE ROTATION FLOWRATE | : | .540000 | lb/sec |
| PRE-DEPLOYMENT BURN | : | 13.3000 | sec |
| BACK-AWAY BURN | : | 40.0000 | sec |
| DUAL LEVEL THRUST RATIO | : | 12.9000 | |
| HIGH THRUST TURN-ON CRITERION | : | .000000 | sec |
| TIER STRUCTURE WEIGHT | : | .000000 | lb |
| DUAL THRUST TRANSITION TIME | : | .000000 | sec |

Real Value: Minimum = .000000 Maximum = .100000E+07

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F8-Main Menu F10-GOTO

**FIGURE 8-6 PBV DATA PANELS (Continued)**

```
                    MISSILE DATA SET - PBV DATA

Panel ID:  207                                    Page Number =  2/ 2


   POST BOOSTER/PBV SEPARATION COAST        :      .000000      sec
   POST BOOSTER/PBV SEPARATION BURN         :      1.00000      sec
   STELLAR SENSOR OPERATING DURATION        :      .000000      sec

   Alt-X  EXIT


 Real Value:  Minimum =     .000000        Maximum =     1000.00

 Esc-Abort  F1-Panel Help  F2-Field Help  F3-Key Table  F8-Main Menu  F10-GOTO
```

**FIGURE 8-6 PBV DATA PANELS (Continued)**

MISSILE DATA SET - PBV OBJECT SPECIFICATION

Panel ID: 208

Page Number = 1/ 3
Number Selected = 3/ 25

| Object | OBJECT TYPE INDICATOR RV = 1 | MINIMUM INTER-OBJECT SPACING ft |
|---|---|---|
| 1 | 1 | 90000.0 |
| 2 | 1 | 90000.0 |
| 3 | 1 | 90000.0 |
| 4 | 0 | .000000 |
| 5 | 0 | .000000 |
| 6 | 0 | .000000 |
| 7 | 0 | .000000 |
| 8 | 0 | .000000 |
| 9 | 0 | .000000 |
| 10 | 0 | .000000 |
| 11 | 0 | .000000 |
| 12 | 0 | .000000 |

Integer Value: Minimum = 0 Maximum = 1

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F10-GOTO RETURN-Exit

**FIGURE 8-6 PBV DATA PANELS (Concluded)**

RV DEFINITION

Panel ID: 209

Page Number = 1/ 2

| | | | |
|---|---|---|---|
| RV NAME | : | US-3RV | |
| NUMBER OF SHOB VALUES | : | 5 | |
| NUMBER OF DRAG vs. MACH ENTRIES | : | 0 | |
| NUMBER OF MaRV α vs. LIFT ENTRIES | : | 0 | |
| BALLISTIC COEFFICIENT | : | 2000.10 | $lb/ft^2$ |
| WEIGHT | : | 1166.70 | lb |
| REFERENCE AREA | : | 7.88300 | $ft^2$ |
| YIELD | : | 321.000 | kT |
| CEP | : | 300.000 | ft |
| WARHEAD/RV RELIABILITY | : | .970000 | |
| NOSE RADIUS | : | 1.00000 | in |
| BASE RADIUS | : | 10.0000 | in |
| CONE HALF-ANGLE | : | 10.0000 | deg |
| SURFACE ROUGHNESS | : | .100000E-01 | in |

Character Value: Maximum number of characters = 20

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F8-Main Menu F10-GOTO

**FIGURE 8-7 RV DATA PANELS**

RV DEFINITION

Panel ID: 209

Page Number = 2/ 2

| | | | |
|---|---|---|---|
| SCALED HEIGHT OF BURST 1 | : | .000000 | ft/kT^1/3 |
| SCALED HEIGHT OF BURST 2 | : | 50.0000 | ft/kT^1/3 |
| SCALED HEIGHT OF BURST 3 | : | 100.000 | ft/kT^1/3 |
| SCALED HEIGHT OF BURST 4 | : | 300.000 | ft/kT^1/3 |
| SCALED HEIGHT OF BURST 5 | : | 1000.00 | ft/kT^1/3 |
| MaRV VISCOUS DRAG COEFFICIENT | : | .100000E-34 | |
| MaRV MAXIMUM LIFT-TO-DRAG RATIO | : | .100000E-34 | |
| MaRV MAXIMUM ATTACK ANGLE (α) | : | .100000E-34 | deg |
| MaRV MINIMUM ATTACK ANGLE (α) | : | .100000E-34 | deg |

SPECIFY RV DRAG vs. MACH DATA
SPECIFY MaRV α vs. LIFT DATA

Alt-X EXIT AND SAVE DATA

Press RETURN to select highlighted option

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F8-Main Menu F10-GOTO

**FIGURE 8-7 RV DATA PANELS (Continued)**

RV DRAG COEFFICIENT vs. MACH NUMBER TABLE

Panel ID: 210

Number Selected = 0/ 10

| Entry | MACH NUMBER | DRAG COEFFICIENT |
|---|---|---|
| 1 | .0 | .00000 |
| 2 | .0 | .00000 |
| 3 | .0 | .00000 |
| 4 | .0 | .00000 |
| 5 | .0 | .00000 |
| 6 | .0 | .00000 |
| 7 | .0 | .00000 |
| 8 | .0 | .00000 |
| 9 | .0 | .00000 |
| 10 | .0 | .00000 |

Real Value: Minimum = .000000 Maximum = 100.000

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F10-GOTO RETURN-Exit

**FIGURE 8-7 RV DATA PANELS (Continued)**

MaRV α vs. LIFT COEFFICIENT TABLE

Panel ID: 211

Number Selected = 0/ 10

| Entry | ANGLE OF ATTACK (α) deg | LIFT COEFFICIENT |
|---|---|---|
| 1 | .0000 | .00000 |
| 2 | .0000 | .00000 |
| 3 | .0000 | .00000 |
| 4 | .0000 | .00000 |
| 5 | .0000 | .00000 |
| 6 | .0000 | .00000 |
| 7 | .0000 | .00000 |
| 8 | .0000 | .00000 |
| 9 | .0000 | .00000 |
| 10 | .0000 | .00000 |

Real Value: Minimum = .000000 Maximum = 89.9999

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F10-GOTO RETURN-Exit

**FIGURE 8-7 RV DATA PANELS (Concluded)**

appear depends on the order in which the user addresses the following options in Figure 8-10: "SPECIFY RUN OPTIONS," "SPECIFY LAUNCH LOCATION DATA" and "SPECIFY RV ALLOCATION." Figures 8-11 through 8-15 reflect the panel order that results when the user works through these options from top to bottom in Figure 8-10.

## 8.3 EXECUTION RESULTS

Figure 8-16 is the ALM Status File, Figure 8-17 is the Attack File, and Figure 8-18 is the Sortie File produced by the execution of this timed laydown case. To interpret these files, the reader needs to consult the IOPTs provided in Figures 6-1 through 6-3. Alternatively, the user has the option of producing up to three special reports from the output Attack File (see Section 6.3). Figure 8-19 provides a description of the resultant timed laydown by missile.

Inspection of this figure reveals the following:

- The input RV allocation has been realized since every DGZ is targeted by 2 RVs.
- Three unique sorties have been formed from the 9 DGZs, and each sortie is executed by two missiles.
- Launch times of first-layer missiles (Missiles 1, 2 and 3) have been adjusted so that the downtime of the first RV to arrive from each of these missiles is identical.
- One missile is launched at the earliest possible launch time (t=0).
- Launch times of second-layer missiles (Missiles 4, 5 and 6) have been adjusted so that the in-line RV spacing is greater than or equal to 2.5 seconds at all DGZs. Note, the intra-DGZ species is identically 2.5 seconds for at least one DGZ targeted by each second-layer missile.

```
                        TIMED LAYDOWN CASE DEFINITION

Panel ID:  004
Case = SAMPLE1     Missile = US-3RV

    CASE NAME                  :  SAMPLE1
    CASE DESCRIPTOR            :  ALM Maintenance Manual sample problem
    SYSTEM OF UNITS            :  English (decimal deg.)
    SELECT MISSILE SYSTEM

    ALLOCATION                 :  Off
    ASSIGNMENT                 :  On
    FRATRICIDE ASSESSMENT      :  Off
    ATTRITION ASSESSMENT       :  Off
    DAMAGE ASSESSMENT          :  Off

    Alt-D  DATA ENTRY / EDIT
    Alt-X  EXIT AND SAVE CASE

  Character Value:  Maximum number of characters =      8

  Esc-Abort  F1-Panel Help  F2-Field Help  F3-Key Table  F8-Main Menu  F10-GOTO
```

**FIGURE 8-8 TIMED LAYDOWN CASE DEFINITION**

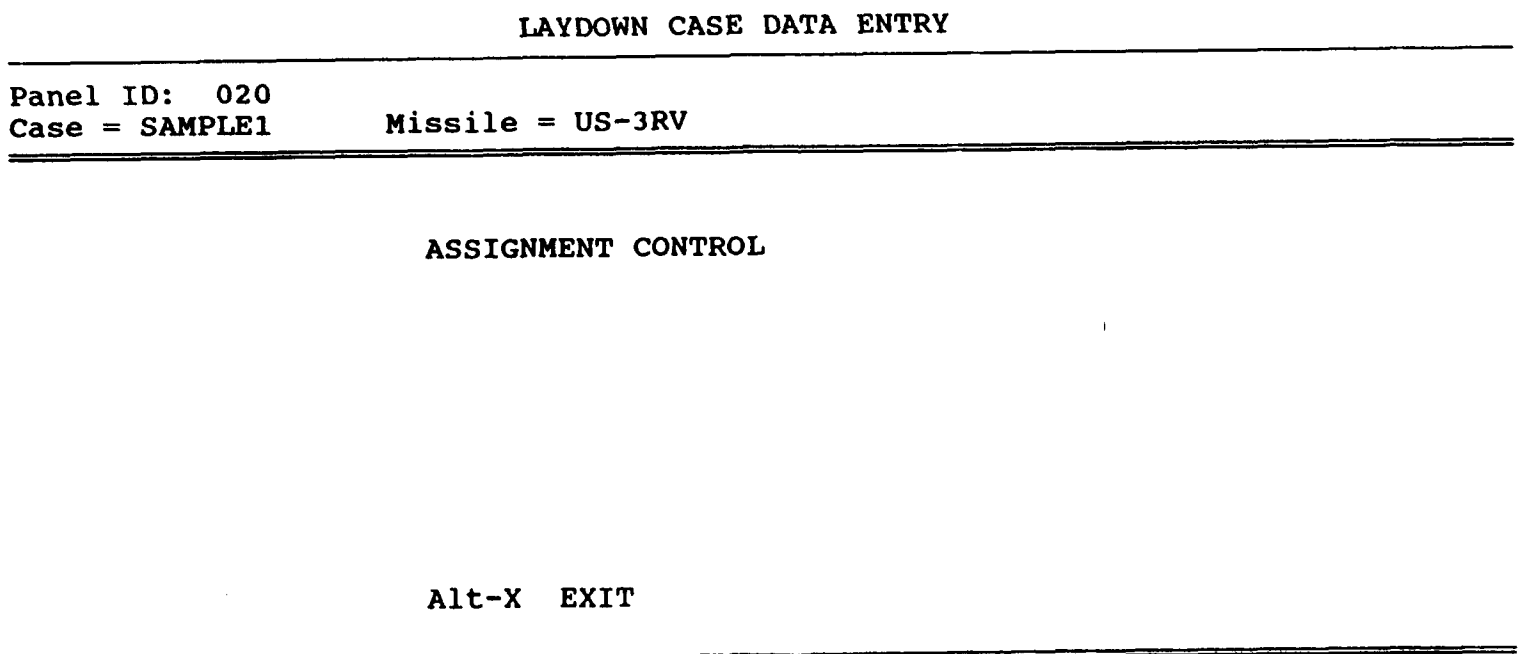



**FIGURE 8-9 LAYDOWN CASE DATA ENTRY**

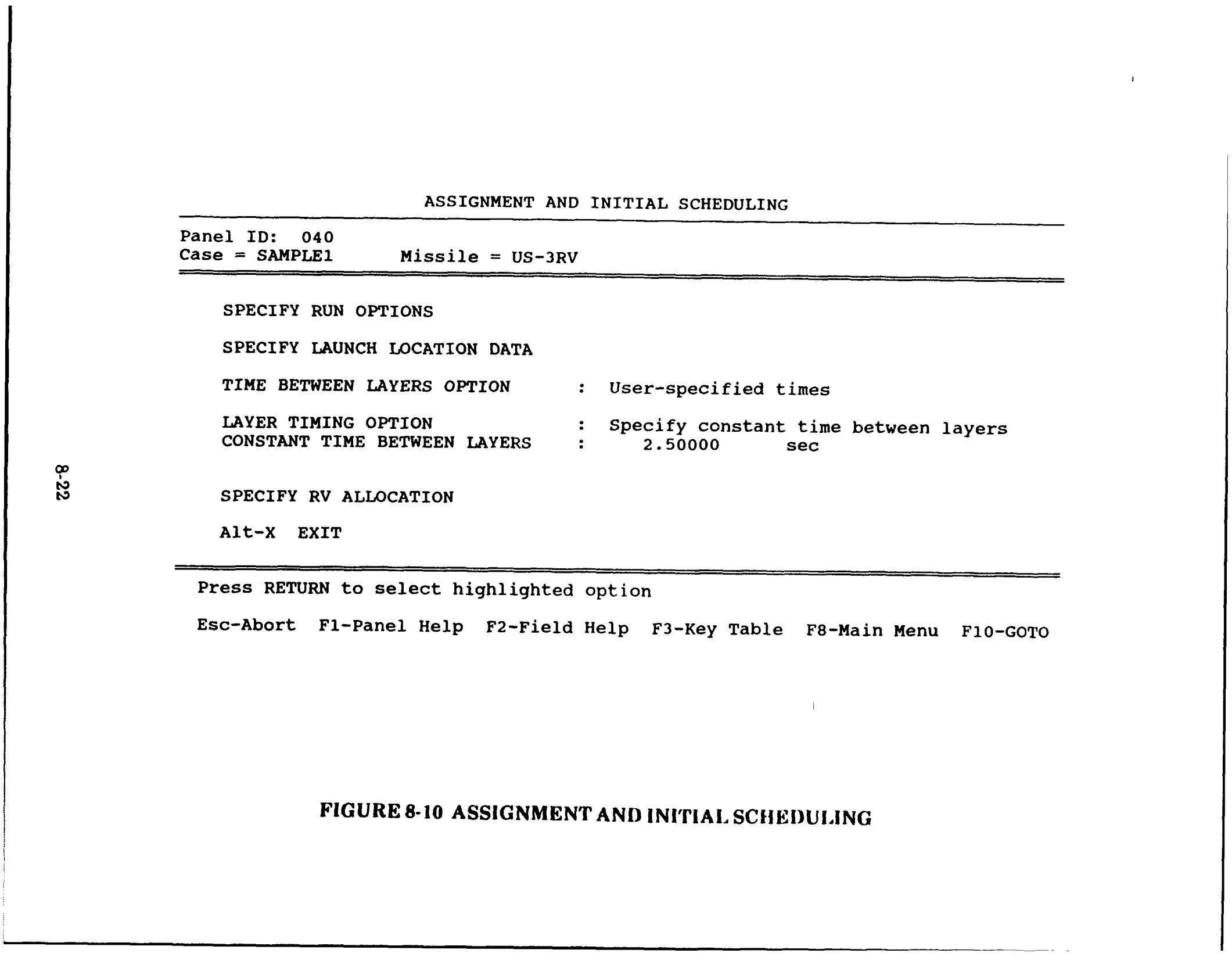



**FIGURE 8-10 ASSIGNMENT AND INITIAL SCHEDULING**

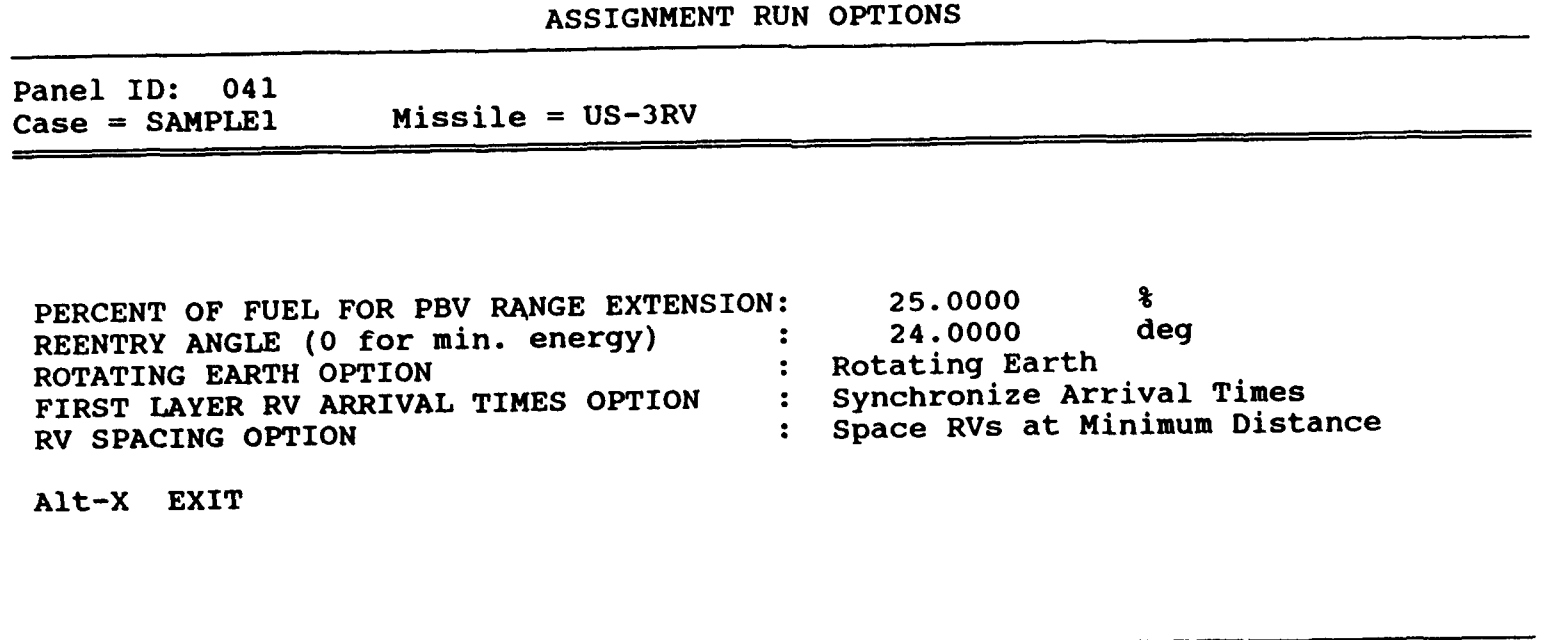



**FIGURE 8-11 ASSIGNMENT RUN OPTIONS**

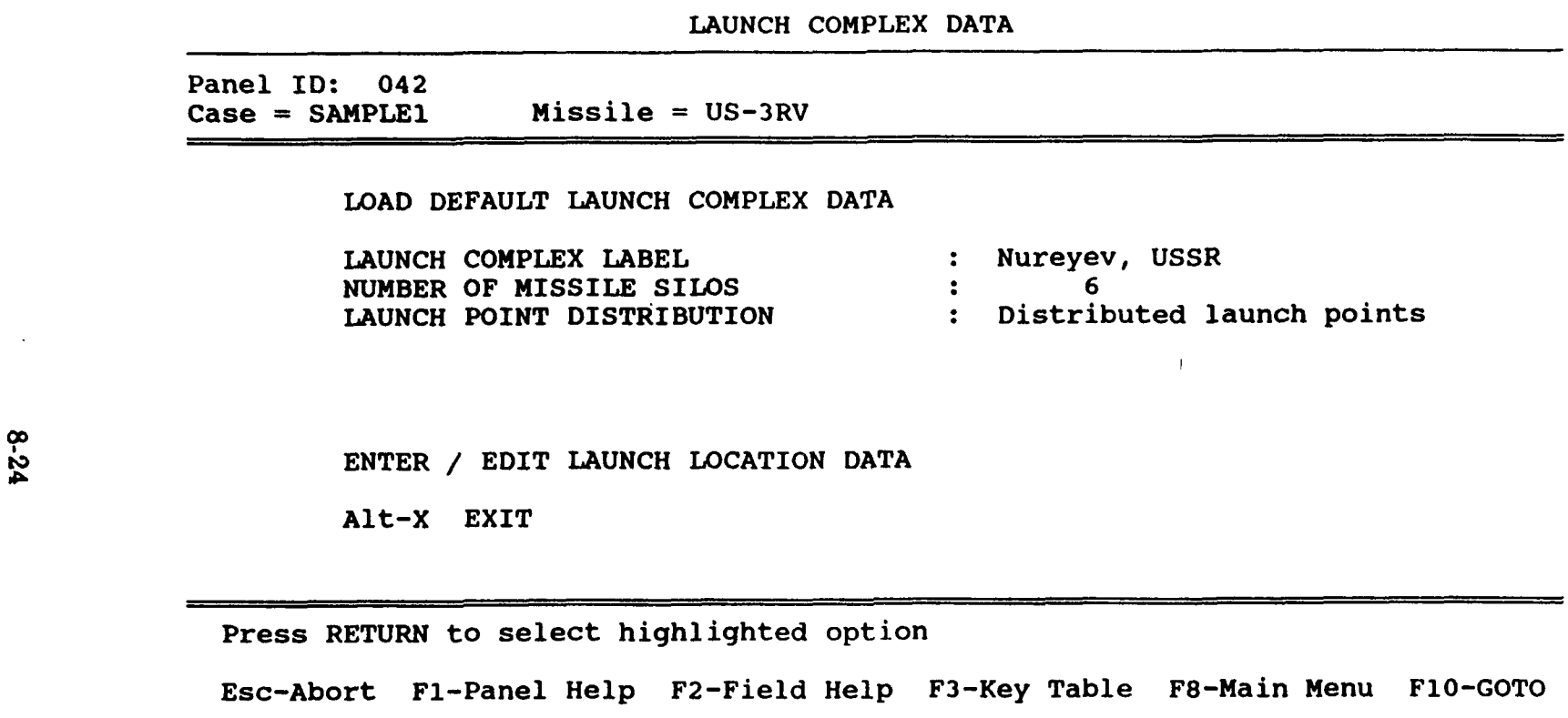



**FIGURE 8-12 LAUNCH COMPLEX DATA**

LAUNCH LOCATION DATA

Panel ID: 043
Case = SAMPLE1 Missile = US-3RV Number Selected = 6/ 10

| Missile | LATITUDE deg (+N) | LONGITUDE deg (+E) | ALTITUDE ft | MISSILE FLIGHT |
|---|---|---|---|---|
| 1 | 55.050 | 79.950 | 1000. | A |
| 2 | 55.050 | 80.000 | 1000. | A |
| 3 | 55.050 | 80.050 | 1000. | A |
| 4 | 54.950 | 79.950 | 1000. | A |
| 5 | 54.950 | 80.000 | 1000. | A |
| 6 | 54.950 | 80.050 | 1000. | A |
| 7 | .000 | .000 | 0. | A |
| 8 | .000 | .000 | 0. | A |
| 9 | .000 | .000 | 0. | A |
| 10 | .000 | .000 | 0. | A |

Real Value: Minimum = -90.0000 Maximum = 90.0000

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F10-GOTO RETURN-Exit

**FIGURE 8-13 LAUNCH LOCATION DATA**

```
                              RV ALLOCATION SPECIFICATION

Panel ID:  046
Case = SAMPLE1       Missile = US-3RV

    LOAD DEFAULT RV ALLOCATION DATA

    TARGET ISLAND LABEL                :  Northern, USA
    TARGET ISLAND ELEVATION            :     3000.00       ft

    HEIGHT OF BURST 1                  :     2381.10       ft
    HEIGHT OF BURST 2                  :     2381.10       ft
    NUMBER OF MISSILES IN ATTACK       :       6

    ENTER / EDIT DATA BY DGZ
    Alt-X  EXIT

    ---------------------------------
    NUMBER OF RVs ALLOCATED            :       18

  Press RETURN to select highlighted option

  Esc-Abort  F1-Panel Help  F2-Field Help  F3-Key Table  F8-Main Menu  F10-GOTO
```

**FIGURE 8-14 RV ALLOCATION SPECIFICATION**

RV ALLOCATION BY DGZ

Panel ID: 047 Page Number = 1/21
Case = SAMPLE1 Missile = US-3RV Number Selected = 9/ 250

| DGZ | LATITUDE deg (+N) | LONGITUDE deg (+E) | HOB INDEX | RVs ALLOCATED |
|---|---|---|---|---|
| 1 | 48.075 | -100.026 | 1 | 2 |
| 2 | 48.075 | -100.000 | 1 | 2 |
| 3 | 48.075 | -99.974 | 1 | 2 |
| 4 | 48.000 | -100.026 | 1 | 2 |
| 5 | 48.000 | -100.000 | 1 | 2 |
| 6 | 48.000 | -99.974 | 1 | 2 |
| 7 | 47.925 | -100.025 | 1 | 2 |
| 8 | 47.925 | -100.000 | 1 | 2 |
| 9 | 47.925 | -99.975 | 1 | 2 |
| 10 | .000 | .000 | 1 | 1 |
| 11 | .000 | .000 | 1 | 1 |
| 12 | .000 | .000 | 1 | 1 |

Real Value: Minimum = -90.0000 Maximum = 90.0000

Esc-Abort F1-Panel Help F2-Field Help F3-Key Table F10-GOTO RETURN-Exit

**FIGURE 8-15 RV ALLOCATION BY DGZ**

```
SAMPLE1    16:59:40  12/22/89
0  16:59:40  12/22/89  RCNTRL  ALM Maintenance Manual sample problem
0  16:59:40  12/22/89  RCNTRL  Since no fratricide models were specified,
4                               times between layers will be read from
4                               the Control File's last record.
0  16:59:40  12/22/89  RCNTRL  Read Control File with no errors.
0  16:59:41  12/22/89  RALLOC  Record 19 in the RV Allocation File
4                               is an end-of-file.
0  16:59:41  12/22/89  RALLOC  Read RV Allocation File with no errors.
0  16:59:41  12/22/89  RLAUNC  Read Launch Location File with no errors.
0  16:59:41  12/22/89  RBOOST  Read Booster File with no errors.
0  16:59:41  12/22/89  RPBVFI  Read PBV File with no errors.
0  16:59:41  12/22/89  RRVFIL  Read RV File with no errors.
0  16:59:54  12/22/89  ORDMSS  6 missiles (18 RVs) will be used.
0  16:59:54  12/22/89  ORDMSS  6 missiles (18 RVs) were specified
4                               in the RV Allocation File.
0  17:00:37  12/22/89  WATTCK  Wrote Attack File successfully.
0  17:00:38  12/22/89  WSORTI  Wrote Sortie File successfully.
2  17:00:38  12/22/89     ALM  ALM finished.
```

**FIGURE 8-16 ALM STATUS FILE**

```
Northern, USA                    3000.US-3RV                          2381.1  238
      .0       .7       .7       .0
  1809.5  1820.6    11.2
   3  55.050  80.050       .0
   2  55.050  80.000       .2
   1  55.050  79.950       .3
   6  54.950  80.050       .5
   5  54.950  80.000       .6
   4  54.950  79.950       .7
   1  1809.5     1 2  48.075-100.026 1  3  3  .0000
   2  1809.6     2 2  48.075-100.000 1  2  3  .0000  .0000
   3  1809.6     3 2  48.075 -99.974 1  1  3  .0000  .0000  .0000
   4  1812.2     1 2  48.075-100.026 1  6  3  .0000  .0000  .0000
   5  1812.2     2 2  48.075-100.000 1  5  3  .0000  .0000  .0000
   6  1812.2     3 2  48.075 -99.974 1  4  3  .0000  .0000  .0000
  .0000
   7  1814.0     4 2  48.000-100.026 1  3  2  .0000  .0000  .0000
  .0000  .0000
   8  1814.1     5 2  48.000-100.000 1  2  2  .0000  .0000  .0000
  .0000  .0000  .0000
   9  1814.1     6 2  48.000 -99.974 1  1  2  .0000  .0000  .0000
  .0000  .0000  .0000  .0000
  10  1816.7     4 2  48.000-100.026 1  6  2  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000
  11  1816.7     5 2  48.000-100.000 1  5  2  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000
  12  1816.7     6 2  48.000 -99.974 1  4  2  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000  .0000
  13  1818.1     7 2  47.925-100.025 1  3  1  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000
  14  1818.1     8 2  47.925-100.000 1  2  1  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000
  15  1818.1     9 2  47.925 -99.975 1  1  1  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000
  16  1820.6     9 2  47.925 -99.975 1  4  1  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000
  .0000
  17  1820.6     7 2  47.925-100.025 1  6  1  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000
  .0000  .0000
  18  1820.6     8 2  47.925-100.000 1  5  1  .0000  .0000  .0000
  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000  .0000
  .0000  .0000  .0000
```

**FIGURE 8-17 ATTACK FILE**

```
Northern, USA                Nureyev, USSR          US-3RV                1  3  3  6
   1    1      9 2  47.925 -99.975    1
   1    2      6 2  48.000 -99.974    1
   1    3      3 2  48.075 -99.974    1
   2    1      8 2  47.925-100.000    1
   2    2      5 2  48.000-100.000    1
   2    3      2 2  48.075-100.000    1
   3    1      7 2  47.925-100.025    1
   3    2      4 2  48.000-100.026    1
   3    3      1 2  48.075-100.026    1
   4    1      9 2  47.925 -99.975    2
   4    2      6 2  48.000 -99.974    2
   4    3      3 2  48.075 -99.974    2
   5    1      8 2  47.925-100.000    2
   5    2      5 2  48.000-100.000    2
   5    3      2 2  48.075-100.000    2
   6    1      7 2  47.925-100.025    2
   6    2      4 2  48.000-100.026    2
   6    3      1 2  48.075-100.026    2
```

**FIGURE 8-18 SORTIE FILE**

ATTACK DESCRIPTION BY MISSILE

Case = SAMPLE1 Missile System = US-3RV
Number of RVs = 18 Number of Missiles = 6

| MISSILE | LAUNCH TIME | RV | DGZ | DOWNTIME | LAYER |
|---|---|---|---|---|---|
| 3 | .0 | 3 | 1 | 1809.5 | 1 |
| 3 | .0 | 2 | 4 | 1814.0 | 1 |
| 3 | .0 | 1 | 7 | 1818.1 | 1 |
| 2 | .2 | 3 | 2 | 1809.6 | 1 |
| 2 | .2 | 2 | 5 | 1814.1 | 1 |
| 2 | .2 | 1 | 8 | 1818.1 | 1 |
| 1 | .3 | 3 | 3 | 1809.6 | 1 |
| 1 | .3 | 2 | 6 | 1814.1 | 1 |
| 1 | .3 | 1 | 9 | 1818.1 | 1 |
| 6 | .5 | 3 | 1 | 1812.2 | 2 |
| 6 | .5 | 2 | 4 | 1816.7 | 2 |
| 6 | .5 | 1 | 7 | 1820.6 | 2 |
| 5 | .6 | 3 | 2 | 1812.2 | 2 |
| 5 | .6 | 2 | 5 | 1816.7 | 2 |
| 5 | .6 | 1 | 8 | 1820.6 | 2 |
| 4 | .7 | 3 | 3 | 1812.2 | 2 |
| 4 | .7 | 2 | 6 | 1816.7 | 2 |
| 4 | .7 | 1 | 9 | 1820.6 | 2 |

**FIGURE 8-19 ATTACK DESCRIPTION BY MISSILE**

# SECTION 9
# COMPILING AND LINKING

## 9.1 COMPILING

The source code for the ALM CSC is contained in eight separate files, reflecting the CSC's overlay structure:

| | |
|---|---|
| ALMROOT.FOR | ALMOVR2B.FOR |
| ALMOVR1A.FOR | ALMOVR2C.FOR |
| ALMOVR1B.FOR | ALMOVR2D.FOR |
| ALMOVR2A.FOR | ALMOVR3.FOR |

The FORTRAN source file containing each ALM module is listed in Figure 9-1. The source file names are meant to indicate the overlay structure of the program; i.e. a root plus three overlays. Because the source files for overlays 1 and 2 grew too big to be added conveniently, they were broken into parts indicated by letters: e.g. 1A and 1B.

Each file is compiled (and the resulting object files are linked) using a MAKE description file (ALM.MAK). See Chapter 14 "Automating Program Development with MAKE" in Reference 1. The MAKE description file is shown in Figure 9-2. It invokes a second MAKE description file (X.CMP) to compile each FORTRAN source file. The one-statement file, X.CMP, is shown in Figure 9-3.

## 9.2 LINKING

The LINK statement at the end of ALM.MAK invokes a response file, shown in Figure 9-4. See Section 12.1.4 "Linking with a Response File" in Reference 3.

| MODULE | SOURCE FILE | MODULE | SOURCE FILE |
|---|---|---|---|
| ALM | ALMROOT.FOR | FXLYRS | ALMOVR2C.FOR |
| ABRTRN | ALMROOT.FOR | GETSRT | ALMOVR2C.FOR |
| ADDGZ1 | ALMOVR2A.FOR | GTBLY2 | ALMOVR2C.FOR |
| ADDGZ2 | ALMOVR2A.FOR | GTBLYR | ALMOVR2C.FOR |
| ADJST1 | ALMOVR2A.FOR | MOPUP | ALMOVR2C.FOR |
| ADJST4 | ALMOVR2A.FOR | MOPUPA | ALMOVR2C.FOR |
| ADJST5 | ALMOVR2A.FOR | MOPUPB | ALMOVR2C.FOR |
| ADLAYR | ALMOVR2A.FOR | MOPUPC | ALMOVR2C.FOR |
| ADXTRA | ALMOVR2A.FOR | MOVLFT | ALMOVR2C.FOR |
| ASR2T2 | ALMOVR2A.FOR | MULSRT | ALMOVR2C.FOR |
| ASR2TI | ALMOVR2A.FOR | ORDDGS | ALMOVR2C.FOR |
| CHKMLT | ALMOVR2A.FOR | ORDMSS | ALMOVR2C.FOR |
| CHKSRT | CHKSRT.FOR | PREPAR | ALMOVER3.FOR |
| CHKTMS | ALMOVR2A.FOR | RALLOC | ALMOVR1A.FOR |
| CHKWST | ALMOVR2A.FOR | RBOOST | ALMORV1A.FOR |
| CROSRT | ALMOVR2B.FOR | RCNTRL | ALMOVR1A.FOR |
| CSTOUT | ALMOVR2B.FOR | RPBVFI | ALMOVR1B.FOR |
| CTRVS | ALMOVR2B.FOR | RRVFIL | ALMOVR1B.FOR |
| CTUNIQ | ALMOVR2B.FOR | RSSAFE | ALMOVR1B.FOR |
| CTUNUS | ALMOVR2B.FOR | SPLSRT | ALMOVR2D.FOR |
| DPATTA | ALMROOT.FOR | SQUEE2 | ALMROOT.FOR |
| DPCNTR | ALMROOT.FOR | SQUEE3 | ALMROOT.FOR |
| DPLC | ALMROOT.FOR | SQUEEZ | ALMOVR2D.FOR |
| DPOPSH | ALMROOT.FOR | SRTSR2 | ALMOVR2D.FOR |
| DPSORT | ALMROOT.FOR | SRTSR3 | ALMOVR2D.FOR |
| DPTI | ALMROOT.FOR | SRTSRT | ALMOVR2D.FOR |
| DPWEAP | ALMROOT.FOR | STRSRT | ALMOVR2D.FOR |
| DWNSRT | ALMOVR2B.FOR | SWPSRT | ALMOVR2D.FOR |
| FMABAB | ALMOVR2B.FOR | SYNEZY | ALMOVR2D.FOR |
| FMABBA | ALMOVR2B.FOR | SYNTIM | ALMOVR2D.FOR |
| FNFRTH | ALMOVR2B.FOR | TRYFLY | ALMOVR1B.FOR |
| FNHIGH | ALMOVR2B.FOR | UNDO | ALMOVR2D.FOR |
| FNPREV | ALMOVR2B.FOR | UPRSRT | ALMOVR2D.FOR |
| FNVRTS | ALMOVR2B.FOR | WSORTI | ALMOVER3.FOR |

**FIGURE 9-1 FORTRAN SOURCE FILES**

```
#ALM.MAK

ALMROOT.OBJ: ALMROOT.FOR
   MAKE X=ALMROOT X.CMP

ALMOVR1A.OBJ: ALMOVR1A.FOR
   MAKE X=ALMOVR1A X.CMP

ALMOVR1B.OBJ: ALMOVR1B.FOR
   MAKE X=ALMOVR1B X.CMP

ALMOVR2A.OBJ: ALMOVR2A.FOR
   MAKE X=ALMOVR2A X.CMP

ALMOVR2B.OBJ: ALMOVR2B.FOR
   MAKE X=ALMOVR2B X.CMP

ALMOVR2C.OBJ: ALMOVR2C.FOR
   MAKE X=ALMOVR2C X.CMP

ALMOVR2D.OBJ: ALMOVR2D.FOR
   MAKE X=ALMOVR2D X.CMP

ALMOVER3.OBJ: ALMOVER3.FOR
   MAKE X=ALMOVER3 X.CMP

ALM.EXE: ALMROOT.OBJ ALMOVR1A.OBJ ALMOVR1B.OBJ ALMOVR2A.OBJ\
         ALMOVR2B.OBJ ALMOVR2C.OBJ ALMOVR2D.OBJ ALMOVER3.OBJ\
         \MPS\TARGET.LIB \MPS\MPS.LIB
  LINK /E /F /PAC /SE:140 @ALM.LNK
```

**FIGURE 9-2 MAKE DESCRIPTION FILE (ALM.MAK)**

```
#X.CMP

$(X).OBJ: $(X).FOR
   FL /4I2 /4Yd /c /Fs /FPi87 /G2 /Ox /Sl100 /Sp78 /St"$(X).FOR"  $(X).FOR
```

**FIGURE 9-3 MAKE DESCRIPTION FILE (X.CMP)**



```
ALMROOT+(ALMOVR1A+ALMOVR1B)+(ALMOVR2A+ALMOVR2B+ALMOVR2C+ALMOVR2D)+(ALMOVER3)
ALM.EXE
NUL
\TARGET\TARGET+\MPS\MPS
NUL
```

**FIGURE 9-4 LINKER RESPONSE FILE (ALM.LNK)**

# SECTION 10
# REFERENCES


1. Microsoft FORTRAN 4.1 Optimizing Compiler, Code View and Utilities Manual, Microsoft Corporation, 1987.

2. MPS Accessibility Computer Software Component - TARGET - Maintenance Manual (MM-05), K-89-59U(R), Kaman Sciences Corporation, 16 January 1990.

3. MPS Support Software - UTILITIES - Maintenance Manual (MM-01), K-89-55U (R), Kaman Sciences Corporation, 16 January 1990.

# APPENDIX A
# MODULE LIST

The module list, presented in this Appendix, identifies all modules used in the ALM CSC except for those TARGET CSD modules that are used by module CHKSRT to evaluate missile sortie feasibility. The reader is referred to Reference 2 for a description of the TARGET modules used by CHKSRT.

In this module list, modules are identified as either the main program (indicated by an M in the Module Type column), a subroutine (indicated by an S), or a function (indicated by an F). Subroutines and functions in the MPS Utilities Library, that are used by the ALM CSC, are indicated by an asterisk: see Reference 3 for detailed descriptions of these modules.

| MODULE | MODULE TYPE | DESCRIPTION |
|---|---|---|
| ABRTRN | S | Aborts the run (writes warning message, dumps all COMMON blocks, writes error message, and stops). |
| ADDGZ1 | S | Adds a DGZ from target island to the current sortie. |
| ADDGZ2 | S | Adds a DGZ from list of unique DGZs to the current sortie. |
| ADJST1 | S | Adjusts launch times and downtimes for all layers beyond the first. |
| ADJST4 | S | Adjusts a missile's launch time and its RVs' downtimes. |
| ADJST5 | S | Adjusts launch times and downtimes for all first-layer missiles. |
| ADLAYR | S | Adds RVs from subsequent layers to DGZs in a sortie, to use up a missile load. |
| ADXTRA | S | Adds extra RVs to most valuable DGZs in a sortie, to use up a missile load. |
| ALM | M | Assigns RVs to DGZs and schedules missile launches. |
| ASR2T2 | S | Assigns RVs to DGZs in the target island. |
| ASR2TI | S | Prepares to assign RVs, assigns RVs, adjusts launch times and downtimes, and fixes layer numbers. |
| ATAN2D* | F | Computes the angle, in degrees, whose tangent is the ratio of the two input arguments. |
| CDSTNC* | S | Computes the distance between two points in 3-dimensional space. |
| CHKMLT | S | Checks variations of a multi-layer sortie. |
| CHKSRT | S | Checks a sortie to see if a missile can successfully drop its RVs on the DGZs in the sortie. |
| CHKTMS | S | Checks downtimes for all RVs in the current sortie that go to the same DGZ. |
| CHKWST | S | Sees if current sortie wastes RVs. |
| CMEAN* | F | Computes the mean of an array of values. |
| CPCNTR* | S | Computes the centroid of a set of points in a plane. |
| CROSRT | S | Forms a crossrange sortie. |
| CSTOUT | S | Marks one of the DGZs in the current sortie as an outcast. |
| CTRVS | S | Counts the total number of RVs still to be assigned to DGZs (that aren't outcasts). |

**ALM MODULE LIST**

| MODULE | MODULE TYPE | DESCRIPTION |
|---|---|---|
| CTUNIQ | S | Counts unique DGZs in a sortie. |
| CTUNUS | S | Counts unused missiles in the launch complex. |
| DFTPCS* | S | Defines a tangent-plane coordinate system. |
| DPATTA | S | Dumps the contents of COMMON block /ATTACK/. |
| DPCNTR | S | Dumps the contents of COMMON blocks /CNTROL/ and /CSTRNG/. |
| DPLC | S | Dumps the contents of COMMON blocks /LC/ and /LSTRNG/. |
| DPOPSH | S | Dumps the contents of COMMON block /OPSHNS/. |
| DPSORT | S | Dumps the contents of COMMON block /SORTIE/. |
| DPTI | S | Dumps the contents of COMMON blocks /TI/ and /TSTRNG/. |
| DPWEAP | S | Dumps the contents of COMMON block /WEAPON/. |
| DWNSRT | S | Sorts a subset of the sortie lists so X-coordinates are in downrange order. |
| FBRVAL* | S | Finds two values in an array that bracket a third value. |
| FCVPOL* | S | Finds the convex polygon enclosing a set of points. |
| FEXT* | S | Finds the extremes in an array of real values. |
| FMABAB | S | Forms a sortie that moves uprange, returns to the beginning, and moves uprange again. |
| FMABBA | S | Forms a sortie that moves uprange, turns around, and then moves downrange. |
| FMAX* | S | Finds the maximum in a set of real values. |
| FMIN* | S | Finds the minimum in a set of real values. |
| FNFRTH | S | Finds the DGZ in the current sortie that's furthest from the sortie's mean Y-coordinate. |
| FNHIGH | S | Finds the highest layer number among all RVs attacking the target island. |
| FNORM* | F | Forms the norm (resultant/magnitude) of a 3-dimensional vector. |
| FNPREV | S | Finds the RV that hit a DGZ and was a member of the previous layer. |

**ALM MODULE LIST (Continued)**

| MODULE | MODULE TYPE | DESCRIPTION |
|---|---|---|
| FNVRTS | S | Finds the DGZs that are the vertices of a convex polygon enclosing the target island. |
| FXLYRS | S | Fixes layer numbers for the sequence of RVs attacking each DGZ. |
| GETSRT | S | Gets DGZs for a strict uprange sortie. |
| GTBLYR | S | Gets time between layers from list supplied in the Control File. |
| GTBLY2 | S | Gets time between layers from list of sure-safe times read from Fratricide Constraints files. |
| IFMIN* | S | Finds the minimum value in a list of integers. |
| INTRPL* | F | Linearly interpolates between (or extrapolates beyond) two elements of a one-dimensional real array. |
| ISWAP* | S | Swaps two INTEGER*2 values. |
| MOPUP | S | Assigns RVs on remaining missile(s) to DGZs. |
| MOPUPA | S | Assigns RVs to DGZs that didn't get any RVs at all. |
| MOPUPB | S | Assigns RVs to DGZs in the sortie that was "short-changed" the most. |
| MOPUPC | S | Assigns RVs to the most valuable sortie. |
| MOVLFT | S | Moves pointer to DGZ immediately to the left of the leftmost DGZ in the current sortie. |
| MULSRT | S | Forms a multi-layer sortie from an uprange sortie by reassigning one DGZ's RVs to other DGZs in the sortie. |
| ORDDGS | S | Orders DGZs; i.e. forms a list of pointers so DGZs can be accessed from right to left. |
| ORDER* | S | Forms a list of pointers so an array of real numbers can be accessed in ascending or descending order. |
| ORDMSS | S | Orders missiles; i.e. forms list of pointers that specifies the order in which missiles should be used. |
| ORDNAL* | F | Returns suffix ("st", "nd", "rd", or "th") corresponding to an integer. |
| PR1PNT* | S | Projects one point on the earth's surface onto a plane tangent to the earth. |

**ALM MODULE LIST (Continued)**

| MODULE | MODULE TYPE | DESCRIPTION |
|---|---|---|
| PRPNTS* | S | Projects a set of points on the earth's surface onto a plane tangent to the earth. |
| PREPAR | S | Prepares data for the Attack File. |
| RALLOC | S | Reads RV Allocation File. |
| RBOOST | S | Reads Booster Data File. |
| RCNTRL | S | Reads Control File. |
| RLAUNC | S | Reads Launch Location File. |
| RMBLKS* | S | Removes all blanks from a string. |
| RMLEAD* | S | Removes leading blanks from a string. |
| RPBVFI | S | Reads PBV Data File. |
| RRVFIL | S | Reads RV Data File. |
| RSSAFE | S | Reads sure-safe time from each Fratricide Constraints File. |
| SPLSRT | S | Splits sortie into two halves. |
| SQBLKS* | S | Squeezes multiple blanks in a string to single blanks. |
| SQUEEZ | S | Forms a list of unique DGZs in a sortie. |
| SQUEE2 | S | Squeezes multiple blanks from the message buffer and then write a Status File Message. |
| SQUEE3 | S | Squeezes multiple blanks from the message buffer and then displays message. |
| SRTSRT | S | Sorts a portion of each list of sortie data so X-coordinates are in descending order. |
| SRTSR2 | S | Sorts a section of the sortie lists so integers in one of the lists are in ascending order. |
| SRTSR3 | S | Sorts the sortie lists so DGZ values are in ascending order. |
| STRSRT | S | Stores successful sortie data in arrays corresponding to missile. |
| SWAP* | S | Swaps two real values. |
| SWPSRT | S | Swaps two sets of entries in the sortie lists. |
| SYNEZY | S | Synchronizes downtimes for RVs in the first layer, for the easy situation where time between launches is zero. |

**ALM MODULE LIST (Continued)**

| MODULE | MODULE TYPE | DESCRIPTION |
|---|---|---|
| SYNTIM | S | Synchronizes downtimes for RVs in the first layer. |
| TRULEN* | F | Finds the true length of a string (ignores trailing spaces and/or nulls). |
| TRYFLY | S | Tries to fly a missile from the silo closest to the target island, so it drops all its RVs on the DGZ furthest from the launch complex. |
| UNDO | S | Undoes a sortie that failed; i.e. resets launch time and number of RVs assigned so far to their original values. |
| UPSRT | S | Sorts a subset of the sortie lists so X-coordinates are in up-range order. |
| WATTCK* | S | Writes the Attack File. |
| WSORTI | S | Writes the Sortie File. |
| WRSFFR* | S | Writes the Status File's first record. |
| WRSFMS* | S | Writes a Status File message. |

**ALM MODULE LIST (Concluded)**

# APPENDIX B
# MODULE CROSS-REFERENCE

This appendix provides a module cross-reference. For each module in the ALM CSC, this cross reference identifies the modules that call it and the modules it calls. Modules preceded by an asterisk are in the MPS Utilities Library.

| MODULE | CALLED BY | CALLS TO |
|---|---|---|
| ABRTRN | ADDGZ1 ADDGZ2 ADJST1 ADLAYR ADXTRA FNPREV GETSRT GTBLYR MOPUP MOPUPC MOVLFT ORDMSS PREPAR RSSAFE | DPATTA DPCNTR DPLC DPOPSH DPSORT DPTI DPWEAP *WRSFMS |
| ADDGZ1 | ADLAYR ADXTRA GETSRT MOPUPA MOPUPB MOPUPC MULSRT | ABRTRN SQUEE2 *WRSFMS |
| ADDGZ2 | FMABAB FMABBA | ABRTRN SQUEE2 *WRSFMS |
| ADJST1 | ASR2TI | ABRTRN ADJST4 FNHIGH FNPREV GTBLY2 GTBLYR SQUEE2 *WRSFMS |
| ADJST4 | ADJST1 ADJST5 | |
| ADJST5 | SYNTIM | ADJST4 |
| ADLAYR | GETSRT | ABRTRN ADDGZ1 SQUEE2 *WRSFMS |
| ADXTRA | GETSRT | ABRTRN ADDGZ1 SQUEE2 SRTSR3 *WRSFMS |
| ALM | - | ASR2TI PREPAR RALLOC RBOOST RCNTRL RLAUNC RPBVFI RRVFIL RSSAFE TRYFLY *WATTCK *WRSFMS WSORTI |
| ASR2T2 | ASR2TI | CHKMLT CHKSRT CHKWST CROSRT CSTOUT CTRVS CTUNIQ DWNSRT GETSRT MOPUP MOVLFT MULSRT *ORDNAL SPLSRT SQUEE3 STRSRT UNDO UPRSRT |
| ASR2TI | ALM | ADJST1 ASR2T2 FNVRTS FXLYRS ORDDGS ORDMSS SYNTIM |
| CHKMLT | ASR2T2 MOPUPA | CHKSRT CHKTMS FMABAB FMABBA SRTSRT |
| CHKSRT | ASR2T2 CHKMLT MOPUPB MOPUPC TRYFLY | (Modules are in TARGET Library) |
| CHKTMS | CHKMLT | GTBLYR |
| CHKWST | ASR2T2 | - |

**ALM MODULE CROSS REFERENCE**

| MODULE | CALLED BY | CALLS TO |
|---|---|---|
| CROSRT | ASR2T2 | *FMAX SWPSRT |
| CSTOUT | ASR2T2 | *FNORM SQUEE2 |
| CTRVS | ASR2T2 GETSRT | - |
| CTUNIQ | ASR2T2 GETSRT MULSRT | - |
| CTUNUS | MOPUP | - |
| DWNSRT | ASR2T2 | *FMIN SWPSRT |
| FMABAB | CHKMLT | ADDGZ2 SQUEEZ |
| FMABBA | CHKMLT | ADDGZ2 SQUEEZ |
| FNFRTH | MULSRT | - |
| FNHIGH | ADJST1 | - |
| FNPREV | ADJST1 | ABRTRN SQUEE2 *WRSFMS |
| FNVRTS | ASR2T1 | *FCVPOL *PRPNTS |
| FXLYRS | ASR2T1 | *ORDER |
| GETSRT | ASR2T2 | ABRTRN ADDGZ1 ADLAYR ADXTRA CTRVS CTUNIQ SQUEE2 SRTSRT *WRSFMS |
| GTBLY2 | ADJST1 | - |
| GTBLYR | ADJST1 CHKTMS | ABRTRN SQUEE2 *WRSFMS |
| MOPUP | ASR2T2 | ABRTRN CTUNUS MOPUPA MOPUPB MOPUPC *WRSFMS |
| MOPUPA | MOPUP | ADDGZ1 CHKMLT SRTSR3 SRTSRT STRSRT UNDO |
| MOPUPB | MOPUP | ADDGZ1 CHKSRT STRSRT UNDO |
| MOPUPC | MOPUP | ABRTRN ADDGZ1 CHKSRT STRSRT UNDO *WRSFMS |
| MOVLFT | ASR2T2 | ABRTRN *FMIN SQUEE2 *WRSFMS |
| MULSRT | ASR2T2 | ADDGZ1 *CPCNTR CTUNIQ FNFRTH SQUEEZ |

**ALM MODULE CROSS REFERENCE (Continued)**

| MODULE | CALLED BY | CALLS TO |
|---|---|---|
| ORDDGS | ASR2TI | *ORDER *PRPNTS |
| ORDMSS | ASR2TI | ABRTRN *ISWAP *ORDER *PLURAL *PRPNTS *RMLEAD SQUEE2 *WRSFMS |
| PREPAR | ALM | ABRTRN *FMIN *ISWAP SQUEE2 *SWAP *WRSFMS |
| RALLOC | ALM | *CPCNTR *ORDNAL *PLURAL *RMLEAD *TRULEN *WRSFMS |
| RBOOST | ALM | SQUEE2 *TRULEN *WRSFMS |
| RCNTRL | ALM | *ORDNAL *RMBLKS *RMLEAD SQUEE2 *SUM *WRSFFR *WRSFMS |
| RLAUNC | ALM | *CMEAN *CPCNTR *ORDNAL *PLURAL *RMLEAD SQUEE2 *TRULEN *WRSFMS |
| RPBVFI | ALM | SQUEE2 *TRULEN *WRSFMS |
| RRVFIL | ALM | SQUEE2 *TRULEN *WRSFMS |
| RSSAFE | ALM | ABRTRN *ORDNAL SQUEE2 *TRULEN *WRSFMS |
| SPLSRT | ASR2T2 | *CDSTNC *FEXT SRTSRT SWPSRT |
| SQUEE2 | ADDGZ1 ADDGZ2 ADJST1 ADLAYR ADXTRA CSTOUT FNPREV GETSRT GTBLYR MOVLFT ORDMSS PREPAR RALLOC RBOOST RCNTRL RLAUNC RPBVFI RRVFIL RSSAFE TRYFLY WSORT1 | *SQBLKS *WRSFMS |
| SQUEE3 | ASR2T2 | *SQBLKS |
| SQUEEZ | FMABAB FMABBA MULSRT | SRTSRT |
| SRTSR2 | SRTSRT | *IFMIN SWPSRT |
| SRTSR3 | ADXTRA MOPUPA | SWPSRT |
| SRTSRT | CHKMLT GETSRT MOPUPA | SRTSR2 UPRSRT |
| STRSRT | ASR2T2 MOPUPA MOPUPB MOPUPC | - |

**ALM MODULE CROSS REFERENCE (Continued)**

| MODULE | CALLED BY | CALLS TO |
|---|---|---|
| SWPSRT | CROSRT DWNSRT SPLSRT SRTSR2 SRTSR3 UPRSRT | *ISWAP *SWAP |
| SYNEZY | SYNTIM | ADJST4 |
| SYNTIM | ASR2TI | ADJST5 SYNEZY |
| TRYFLY | ALM | CHKSRT *FMAX *PRPNTS SQUEE2 |
| UNDO | ASR2T2 MOPUPA MOPUPB MOPUPC | - |
| UPRSRT | ASR2T2 SRTSRT | *FMAX SWPSRT |
| WSORTI | ALM | SQUEE2 |

**ALM MODULE CROSS REFERENCE (Concluded)**

# APPENDIX C
# MODULE DESCRIPTIONS

The module descriptions contained in this appendix are confined to those modules that are unique to the ALM CSC. Modules (subroutines and functions) used in ALM that are part of the MPS Utilities are described in Reference 3. Modules shared by the ALM and TARGET CSCs are described in Reference 2.

A set of modules whose names have the form DPxxxx are NOT described in this appendix. Each module dumps the contents of a COMMON block when the ALM aborts (the string "xxxx" is usually the first four characters of the block's name). Because each module consists only of print and format statements they are not described in detail.

A special module, CHKSRT, is described in this document, even though the source code resides in the TARGET directory. CHKSRT is an interface between the ALM and modules shared by TARGET and ALM.

PROGRAM ALM

Purpose. To assign RVs to DGZs and to schedule missile launch times.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | | |
|---|---|---|
| ASR2TI | RRVFIL | |
| PREPAR | RSSAFE | |
| RALLOC | TRYFLY | |
| RBOOST | WATTCK | (MPS Library) |
| RCNTRL | WRSFMS | (MPS Library) |
| RLAUNC | WSORTI | |
| RPBVFI | | |

Calling Modules:

None

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| ATTAKK | CBATTACK.FOR | OPSHNS | CBOPSHNS.FOR |
| CNTROL | CBCNTROL.FOR | SORTEE | CBSORTIE.FOR |
| CSTRNG | CBCSTRNG.FOR | TI | CBTI.FOR |
| LC | CBLC.FOR | TSTRNG | CBTSTRNG.FOR |
| LSTRNG | CBLSTRNG.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE statement.

Description:

Program ALM is the main driver for the MPS assignment function. This program and its subroutines assign RVs to DGZs and schedule missile launch times to generate a timed laydown. The program calls subroutines to read the Control File, RV Allocation File, Launch Location File, Booster Data File, PBV Data File, and RV Data File. Then the program tries to fly the missile closest to the target island so that all its RVs fall on the DGZ farthest from the launch complex. If the trial flight fails, ALM stops. Otherwise, if the user has supplied the names of some Fratricide Constraints Files, the program reads the sure-safe time from each file to use as times between layers. Then the program calls the major subroutine ASR2TI to assign RVs to the target island and schedule launches. Next, ALM prepares data for the Attack File and writes the file. Finally, the program writes the Sortie File and the success message required for the Status File.

Source File:

ALMROOT.FOR

SUBROUTINE ABRTRN

Purpose: To abort the run.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | | |
|---|---|---|
| DPATTA | DPSORT | |
| DPCNTR | DPTI | |
| DPLC | DPWEAP | |
| DPOPSH | WRSFMS | (MPS Library) |

Calling Modules:

| | |
|---|---|
| ADDGZ1 | GTBLYR |
| ADDGZ2 | MOPUP |
| ADJST1 | MOPUPC |
| ADLAYR | MOVLFT |
| ADXTRA | ORDMSS |
| FNPREV | PREPAR |
| GETSRT | RSSAFE |

Common Blocks Used:

None

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ABRTRN writes a warning message to the Status File, indicating the contents of all COMMON blocks will be written, as information messages, in the Status File. ABRTRN then proceeds to do exactly that. Finally, the subroutine writes the error message required for the Status File and stops.

Source File:

ALMROOT.FOR

SUBROUTINE ADDGZ1 (IS, ID)

Purpose: To add a DGZ to the current sortie.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IS | INTEGER*2 | Sortie index; i.e., slot in which to store next set of sortie data. Ranges from 1 through number of RVs on a weapon. |
| ID | INTEGER*2 | Index of DGZ to add to sortie; ranges from 1 through number of DGZs in target island. |

Output Arguments:

None

Modules Called:

ABRTRN
SQUEE2
WRSFMS (MPS Library)

Calling Modules:

ADLAYR
ADXTRA
GETSRT
MOPUPA
MOPUPB
MOPUPC
MULSRT

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| OUTPUT | CBOUTPUT.FOR | TI | CBTI.FOR |
| SORTEE | CBSORTIE.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ADDGZ1 first checks the sortie index. If it's bad, ADDGZ1 writes a warning message and then calls ABRTRN to abort the run. The subroutine also checks the DGZ index and takes similar action if it's bad. Otherwise, ADDGZ1 stores the DGZ index in the list of DGZ indices for the sortie and increments the number of RVs assigned to the DGZ so far. If the maximum number of layers is exceeded, ADDGZ1 writes a warning message and calls ABRTRN to abort the run. If not, ADDGZ1 stores the number of RVs in the list of layer numbers for the sortie. Finally, ADDGZ1 stores the DGZ's latitude, longitude, X-coordinate, and Y-coordinate in the appropriate sortie lists.

Source File:

ALOVR2A.FOR

SUBROUTINE ADDGZ2 (IS, IU)

Purpose: To add a DGZ, from the list of unique DGZs, to the current sortie.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IS | INTEGER*2 | Sortie index; i.e., slot in which to store next set of sortie data. Ranges from 1 through number of RVs on a weapon. |
| IU | INTEGER*2 | Index for list of unique DGZs. Ranges from 1 through number of RVs per weapon. |

Output Arguments:

None

Modules Called:

ABRTRN
SQUEE2
WRSFMS (MPS Library)

Calling Modules:

FMABAB
FMABBA

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| OUTPUT | CBOUTPUT.FOR | TI | CBTI.FOR |
| SORTEE | CBSORTIE.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ADDGZ2 first checks the sortie index. If it's bad, ADDGZ2 writes a warning message and then calls ABRTRN to abort the run. The subroutine also checks the unique DGZ index and takes similar action if it's bad. Otherwise, ADDGZ2 gets values corresponding to the DGZ from lists of unique DGZ data and stores the values in corresponding lists of sortie data.

Source File:

ALMOVR2A.FOR

SUBROUTINE ADJST1

Purpose: To adjust launch times and downtimes for all layers beyond the first.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | | |
|---|---|---|
| ABRTRN | GTBLY2 | |
| ADJST4 | GTBLYR | |
| FNHIGH | SQUEE2 | |
| FNPREV | WRSFMS | (MPS Library) |

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | TI | CBTI.FOR |
| LC | CBLC.FOR | WEAPON | CBWEAPON.FOR |
| OUTPUT | CBOUTPUT.FOR | | |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ADJST1 first finds the highest layer number in the entire attack. Then ADJST1 executes a loop over all layers from the second through the highest.

At the beginning of each pass through the loop, ADJST1 forms the number of the previous layer. If no CAIN models (Fratricide Constraints Files) were read, ADJST1 gets the time between the previous and current layers from the list on the Control File's last record. If the time between launches is greater than this value, the time between launches is used instead.

ADJST1 executes an inner loop over all missiles used. At the beginning of each pass, ADJST1 checks to see if Fratricide Constraints Files were read. If so, the time between the previous and current layers is revised. Then, ADJST1 executes another inner loop, over the RVs on the missile. ADJST1 computes the difference in downtimes between each RV on the current missile, and the previous RV attacking the RV's DGZ. The smallest difference is saved.

If the smallest difference is greater than the time between layers, ADJST1 proceeds to the next missile. If not, ADJST1 adjusts the current missile's launch time. If the time between launches isn't zero, ADJST1 also adjusts the launch times of every missile launched after the current missile.

After examining all missiles used in each layer, ADJST1 returns.

Source File:

ALMOVR2A.FOR

SUBROUTINE ADJST4 (AJSMNT, IM, NRVS)

Purpose: To adjust launch times and downtimes for one missile.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| AJSMNT | REAL*4 | Delta-time to add to launch and downtimes (seconds). |
| IM | INTEGER*2 | Missile index. |
| NRVS | INTEGER*2 | Number of RVs on missile. |

Output Arguments:

None

Modules Called:

None

Calling Modules:

ADJST1
ADJST5

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ADJST4 first adds the adjustment to the missile's launch time. Then it adds the adjustment to each RV's downtime.

Source File:

ALMOVR2A.FOR

SUBROUTINE ADJST5 (NRVS, LATEST)

Purpose: To adjust launch times and downtimes for all first-layer missiles, when the time between launches is non-zero. The objective is to minimize the span of downtimes.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| NRVS | INTEGER*2 | Number of RVs on each missile. |
| LATEST | REAL*4 | Latest downtime (seconds) among all first-layer RVs. |

Output Arguments:

None

Modules Called:

ADJST4

Calling Modules:

SYNTIM

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ADJST5 minimizes the span of downtimes by matching missiles with longer times-of-flight with earlier launch times, and missiles with shorter times-of-flight with later launch times.

ADJST5 first examines each first-layer missile, storing the missile index in one list and the launch time in another. The list of launch times is sorted so launch times are in DESCENDING order. The list of missile indices is sorted so the downtimes of the first RV to arrive from each missile are in ASCENDING order. Then ADJST5 executes a loop that subtracts each missile's original launch time from, and adds the new sorted launch time to, the downtime of each RV on the missile. Finally, ADJST5 replaces each missile's original launch time with the new sorted launch time.

Source File:

ALMOVR2A.FOR

SUBROUTINE ADLAYR (IS, NRVS)

Purpose: To add RVs from subsequent layers to DGZs in a sortie, to use up a missile load.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IS | INTEGER*2 | Index for next entry in sortie data lists. Must be greater than 1. |
| NRVS | INTEGER*2 | Number of RVs on missile performing sortie. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| IS | INTEGER*2 | Index for next entry in sortie data lists. The input argument is incremented each time a DGZ is added to the sortie. |

Modules Called:

| | |
|---|---|
| ABRTRN | SQUEE2 |
| ADDGZ1 | WRSFMS |

Calling Modules:

GETSRT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| OUTPUT | CBOUTPUT.FOR |
| SORTEE | CBSORTIE.FOR |
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ADLAYR first checks the input argument IS. If it's less than or equal to 1, or greater than the maximum number of RVs on a missile, ADLAYR writes a warning message in the Status File. It also writes information messages containing the bad sortie index and maximum number of RVs per missile. Then ADLAYR calls ABRTRN to abort the run.

ADLAYR performs the same pair of checks on the input argument NRVS. If it's bad, ADLAYR writes a similar sequence of Status File messages and calls ABRTRN.

Otherwise, ADLAYR defines the start and stop indices for data already in the sortie lists. The start index is simply 1, and the stop index is IS-1. Then ADLAYR sets a second list index, a search index called JS, equal to the start index, and zeros a counter of DGZs added to the lists. ADLAYR loops through the DGZs already in the sortie, looking for one that needs another RV. If no such DGZ is found, ADLAYR returns.

But if such a DGZ exists, it's added to the sortie, the counter of DGZs added is incremented, and the sortie index is incremented. If the sortie is complete (i.e., if all RVs on the missile have been used), control returns to GETSRT. Otherwise, the search index, JS, is incremented. If all of the original DGZs in the sortie have been examined, and no DGZ was added, control returns to GETSRT. Otherwise, the search index is reset to the first original DGZ in the list and another pass is made. If there's still another original DGZ in the current pass to examine, it's examined.

Source File:

ALMOVR2A.FOR

SUBROUTINE ADXTRA (IS, NRVS)

Purpose: To add RVs to the most valuable DGZs in a sortie, to use up a missile load.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IS | INTEGER*2 | Index for next entry in sortie data lists. Must be greater than 1. |
| NRVS | INTEGER*2 | Number of RVs on missile performing sortie. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| IS | INTEGER*2 | Index for next entry in sortie data lists. The input argument is incremented each time a DGZ is added to the sortie. |

Modules Called:

| | |
|---|---|
| ABRTRN | SRTSR3 |
| ADDGZ1 | WRSFMS |
| SQUEE2 | |

Calling Modules:

GETSRT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| OUTPUT | CBOUTPUT.FOR |
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ADXTRA first checks the input argument IS. If it's less than or equal to 1,or greater than the maximum number of RVs on a missile, ADXTRA writes a warning message in the Status File. It also writes information messages containing the bad sortie index and maximum number of RVs per missile. Then ADXTRA calls ABRTRN to abort the run.

ADXTRA performs the same pair of checks on the input argument NRVS. If it's bad, ADXTRA writes a similar sequence of Status File messages and calls ABRTRN.

Otherwise, ADXTRA defines the start index for data already in the sortie lists. The start index is IS-1, because DGZs in the sortie will be selected in reverse order. Then ADXTRA sorts the sortie lists so DGZ values are in ascending order.

Next, ADXTRA sets a second list index, JS, equal to the start index. ADXTRA loops backwards through the DGZs already in the sortie, from

most to least valuable, adding each DGZ to the sortie. If the sortie is complete (i.e., if all RVs on the missile have been used), control returns to GETSRT. Otherwise, the second index, JS, is incremented. If all of the original DGZs in the sortie have been examined, JS is reset to the start of the sorted lists and another pass is made. If there's still another original DGZ in the current pass to examine, it's examined.

Source File:

ALMOVR2A.FOR

SUBROUTINE ASR2T2

Purpose: To form sorties for each missile that should be used.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | | | | | |
|---|---|---|---|---|---|
| CHKMLT | CROSRT | CTUNIQ | MOPUP | ORDNAL | STRSRT |
| CHKSRT | CSTOUT | DWNSRT | OMVLFT | SPLSRT | UNDO |
| CHKWST | CTRVS | GETSRT | MULSRT | SQUEE3 | UPRSRT |

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | SORTEE | CBSORTIE.FOR |
| OPSHNS | CBOPSHNS | TI | CBTI.FOR |
| OUTPUT | CBOUTPUT.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ASR2T2 begins by defining three variables: half the number of RVs on a missile; an INTEGER*2 representation of the number of DGZs in a sortie (equals the number of RVs on a missile); and an INTEGER*4 version of the same value (for subroutine CHKSRT). ASR2T2 also sets a pointer to the ordered list of DGZ indices to 1, and a pointer to the ordered list of missile indices to 1. Because ASR2T2 is long and complicated, the remainder of the description is keyed to statement numbers in the source code.

(10) ASR2T2 begins a loop over all the missiles to be used by getting a missile index from the ordered list, and displaying a message on the screen indicating the current sortie.

(15) ASR2T2 calls subroutine CTRVS to count the DGZs that aren't outcasts and which still need at least one more RV. If more than half a missile load is required, ASR2T2 goes to statement 17. If not, ASR2T2 examines the DGZ pointer. If it's 1, meaning the current DGZ is the rightmost DGZ, ASR2T2 goes to statement 80, to mop up. Otherwise, ASR2T2 resets the DGZ pointer to 1 and goes back to statement 15.

(17) ASR2T2 calls subroutine GETSRT to get an uprange sortie. The height of burst (HOB) for the first DGZ in the sortie is defined to be the HOB for all DGZs in the sortie. The launch time for the missile performing the sortie is computed by adding time between launches to the previous launch time. If the number of unique DGZs in the sortie is less than the number of DGZs in the sortie, the sortie is a multi-layer sortie; i.e. at least one DGZ in the sortie gets more than one RV. ASR2T2 goes to statement 30 to check the multi-layer sortie.

Otherwise, ASR2T2 calls CHKSRT to see if the missile can do the uprange (single-layer) sortie. If so, ASR2T2 goes to statement 60 to save the successful sortie data. If not, ASR2T2 displays a message indicating the uprange sortie failed and calls SPLSRT to form a "split sortie".

ASR2T2 calls CHKSRT again to see if the missile can do the split sortie. If so, ASR2T2 goes to statement 60 to save the successful sortie data. If not, ASR2T2 displays a message indicating the split sortie failed and calls CROSRT to form a crossrange sortie.

ASR2T2 calls CHKSRT again to see if the missile can do the crossrange sortie. If so, ASR2T2 goes to statement 60 to save the successful sortie data. If not, ASR2T2 displays a message indicating the crossrange sortie failed and calls DWNSRT to form a downrange sortie.

ASR2T2 calls CHKSRT again to see if the missile can do the downrange sortie. If so, ASR2T2 goes to statement 60 to save the successful sortie data. If not, ASR2T2 displays a message indicating the downrange sortie failed and calls UPRSRT to form an uprange sortie again.

(25) ASR2T2 calls MULSRT to form a multi-layer sortie; i.e., one of the DGZs in the current sortie is eliminated and its RVs reassigned to the other DGZs in the sortie. Therefore, at least one of the remaining DGZs gets more than one RV. The DGZ eliminated is the one furthest from the sortie's uprange/downrange axis.

(30) ASR2T2 calls CHKMLT to see if the missile can do the current multi-layer sortie. If so, ASR2T2 goes to statement 40 to see if the sortie is wasteful. If the missile can't do the sortie, ASR2T2 calls CTUNIQ to count the number of unique DGZs in the sortie. If there are more than two, ASR2T2 goes back to statement 25 to form a new multi- layer sortie by eliminating yet another DGZ. If there are only two DGZs in the sortie, ASR2T2 goes to statement 50 to make one of them an outcast.

(40) ASR2T2 calls CHKWST to see if the sortie just formed is wasteful; i.e., if more than half the RVs deployed are more than their respective

DGZs should get. If the sortie isn't wasteful, ASR2T2 goes to statement 60 to save the sortie data.

(50) ASR2T2 calls CSTOUT to cast out one of the DGZs in the current sortie. The DGZ furthest from the target island's centroid becomes the outcast. Then ASR2T2 calls UNDO to undo the sortie; i.e. to decrement the number of RVs assigned so far to the DGZs in the sortie and to reset the previous launch time to its former value. ASR2T2 goes back to statement 15 to recount the number of DGZs that need RVs.

(60) ASR2T2 calls STRSRT to store the sortie lists in arrays, and displays a message that a sortie has been formed. ASR2T2 increments the pointer to the list of missile indices. If all missiles have been used, ASR2T2 goes to statement 90 to return. Otherwise, ASR2T2 calls MOVLFT to move the DGZ pointer. It's moved to the DGZ immediately to the left of the leftmost DGZ in the current sortie. Then ASR2T2 goes to statement 10 to get the index of the next missile to use.

(80) ASR2T2 calls MOPUP to use any leftover missile(s), and displays a message indicating that fact.

(90) ASR2T2 returns control to ASR2TI.

Source File:

ALMOVR2A.FOR

SUBROUTINE ASR2TI

Purpose: To assign RVs to DGZs and schedule launches.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | | |
|---|---|---|
| ADJST1 | FXLYRS | SYNTIM |
| ASR2T2 | ORDDGS | |
| FNVRTS | ORDMSS | |

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| OPSHNS | CBOPSHNS.FOR |
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine ASR2TI begins by zeroing the number of RVs assigned so far to each DGZ in the target island. It also sets a flag associated with each DGZ to .FALSE., indicating no DGZ is an outcast (yet).

ASR2TI calls FNVRTS to find which DGZs are the vertices of a convex polygon enclosing the target island. It displays a message stating it has done so.

ASR2TI sets the downtime for each RV on each missile to an absurd value (-1.0).

ASR2TI calls ORDDGS to form an ordered list of DGZ indices. DGZs are ordered from right to left; i.e., so their Y-coordinates are in descending order. ASR2TI calls ORDMSS to form an ordered list of missile indices. Missiles are used in horizontal strips, from the front of the launch complex to the back. Within each strip, missiles are used from right to left.

ASR2TI calls ASR2T2 to form a sortie for each missile that should be used. If RV downtimes for the first layer should be "synchronized" ASR2TI calls SYNTIM to do just that. Note that downtimes can be truly synchronized only if the time between launches is zero; if it's non-zero, the best that can be done is to minimize the span of downtimes.

ASR2TI calls ADJST1 to adjust launch times and downtimes for subsequent layers. It calls FXLYRS to fix layer numbers within each string of RVs going to the same DGZ. It's possible for the downtime of an RV in a given layer to be earlier than the downtime of an RV in the previous layer! FXLYRS ensures that the layer numbers and downtimes for each

string of RVs are both ascending sequences. Then ASR2TI returns control to ALM.

Source File:

ALMOVR2A.FOR

SUBROUTINE CHKMLT (JM, NDPLYD, BTNEW, PNEW, TOFNEW)

Purpose: To check variations of a multi-layer walk.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| JM | INTEGER*2 | Missile number (within launch complex). |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| NDPLYD | INTEGER*2 | Number of RVs deployed. If NDPLYD equals number of RVs on missile, missile can do multilayer sortie. |
| BTNEW | REAL*4 | New burn-time (seconds) to use for next sortie. Defined only if sortie was successful. |
| PNEW | REAL*4 | New pitch (degrees). |
| TOFNEW | REAL*4 | New time-of-flight (seconds). |

Modules Called:

CHKSRT
CHKTMS
FMABAB
FMABBA
SRTSRT

Calling Modules:

ASR2T2
MOPUPA

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | TI | CBTI.FOR |
| OPSHNS | CBOPSHNS.FOR | WEAPON | CBWEAPON.FOR |
| SORTEE | CBSORTIE.FOR | | |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine CHKMLT begins by calling subroutine SRTSRT to form an uprange walk. Then CHKMLT prepares the input arguments needed to call subroutine CHKSRT and calls it. If the sortie fails, CHKMLT returns control to the calling module.

If the sortie is successful, however, CHKMLT calls CHKTMS to check the downtimes for each string of RVs that goes to the same DGZ. If the absolute value of the difference between each successive pair of downtimes is greater than or equal to the time between layers, the times are good, and control returns to the calling module.

If the times are bad, CHKMLT calls FMABAB to form a sortie that moves uprange, backs up to the beginning, and moves uprange again. Then CHKMLT calls CHKSRT to see if the missile can do the sortie. If so, control returns to the calling module.

If not, CHKMLT calls FMABBA to form a sortie that moves uprange, then turns around and moves downrange. CHKMLT calls CHKSRT again to see if the missile can do this sortie. If so, control returns to the calling module.

If not, CHKMLT forms the uprange sortie it formed at the beginning, sets the number of RVs deployed equal to the number of RVs on a missile, and returns.

Source File:

ALMOVR2A.FOR

SUBROUTINE CHKSRT (LEARTH, RXFUEL, LSPACE, SLAT, SLON, ELEVL, NPTS, APLAT, APLON, ELEVT, HOB1, IWALK, GAMMA1, TLNCH, BTIN, PIN, TOFIN, IWPN, WDATA, IWDATA, NDOWN, TDOWN, BTNEW, PNEW, TOFNEW)

Purpose: To assess a missile sortie's feasibility by simulating a missile's flight and determining if the sortie is within the missile's performance capability. This subroutine determines: (1) whether the missile can reach the first DGZ in the sortie, at the desired reentry angle, given the maximum booster burn time and the maximum allowable PBV propellant for range extension; and (2) whether the PBV can target all of its RVs, to their assigned DGZs, within the available PBV propellant load.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| LEARTH | LOGICAL*1 | Flag: if. TRUE., rotating earth should be used. |
| RXFUEL | REAL*4 | Maximum fraction of fuel to use for range extension. |
| LSPACE | LOGICAL*1 | Flag: if .TRUE., RVS should be spaced at least a certain minimum distance apart at an altitude specific to each type of missile. |
| SLAT | REAL*4 | Latitude from which missile is launched, in degrees ( + North/-South). |
| SLON | REAL*4 | Longitude from which missile is launched, in degrees ( + East/-West). |
| ELEVL | REAL*4 | Altitude from which missile is launched, in feet. |
| NPTS | INTEGER*4 | Number of DGZs in sortie. |

| | | |
|---|---|---|
| APLAT | REAL*4 | Array: latitude of each DGZ in the sortie, in degrees ( + North/-South). |
| APLON | REAL*4 | Array: longitude of each DGZ in the sortie, in degrees ( + East/-West). |
| ELEVT | REAL*4 | Target island's elevation (altitude) in feet above mean sea level. |
| HOB1 | REAL*4 | Height of burst for every DGZ in the sortie, in feet. First DGZ's height of burst is used for all DGZs. |
| IWALK | INTEGER*4 | Sortie type: 0 = One PBV; 1 = Two or more PBVs. |
| GAMMA1 | REAL*4 | Reentry angle for first RV, in degrees. |
| TLNCH | REAL*4 | Launch time in seconds. |
| BTIN | REAL*4 | Burntime in seconds. |
| PIN | REAL*4 | Booster pitch in degrees. |
| TOFIN | REAL*4 | Time-of-flight in seconds. |
| IWPN | INTEGER*4 | Weapon type indicator. |
| WDATA | REAL*4 | Array: contains real weapon data. |
| IWDATA | INTEGER*4 | Array: contains integer weapon data. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| NDOWN | INTEGER*2 | Number of RVs deployed. If NDOWN equals number of RVs on missile, missile can do sortie. |
| TDOWN | REAL*4 | Array: downtime for each RV in the sortie, in seconds. |
| BTNEW | REAL*4 | New burn time (seconds) to use for next sortie. Defined only if sortie was successful. |
| PNEW | REAL*4 | New pitch (degrees). |
| TOFNEW | REAL*4 | New time-of-flight (seconds). |

Modules Called:

All modules called by CHKSRT are in the TARGET CSC's library (TARGET.LIB).

Calling Modules:

| | |
|---|---|
| ASR2T2 | MOPUPC |
| CHKMLT | TRYFLY |
| MOPUPB | |

Common Blocks Used:

None

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

CHKSRT is used to interface between the modules of the TARGET CSC that are used by the ALM CSC and the calling module in the ALM. A subset of the TARGET subroutines is used by the ALM to CHecK SoRTie feasibility when generating attack laydowns. Since sortie feasibility assessment is an option of TARGET when it is executed autonomously, that software can be used by ALM for the same purpose.

The first time that CHKSRT is called during an ALM run, it begins by reassigning the missile data set (MDS) information for the booster, PBV and RV, contained in the arrays WDATA and IWDATA, to working

variables compatible with TARGET. Note that in order to keep the interface between ALM and TARGET as simple as possible, all MDS information is read by subroutines RBOOST, RPBVFI and RRVFIL, with the real variables being stored in WDATA and the integer variables in IWDATA. This reassignment is performed only for the first call to CHKSRT since the MDS information will remain the same (single weapon type used).

Processing continues with the targeting of the booster to the first DGZ in the sortie at either a prescribed reentry angle (passed in via GAMMA1) or at minimum energy (GAMMA1 equals zero). If the first DGZ is accessible by the booster or if it is accessible via range extension, then RVs are deployed to the remaining DGZs in the sortie. If the sortie is feasible (i.e., all RVs can be deployed successfully to their prescribed DGZs), CHKSRT saves the downtimes and sets the number of RVs successfully deployed (NDOWN) equal to the number of RVs on board the missile before exiting. However, if the sortie is infeasible, then one of the following three steps is taken: a) If range extension is required to reach the first DGZ and the sortie still cannot be completed, then NDOWN is set to -1; b) if the first DGZ is out of range even with range extension, then NDOWN is set to -2; and c) if the first DGZ is less than the minimum range capability of the booster, then NDOWN is set to -3.

Source File:

CHKSRT.FOR

(Please note that CHKSRT.FOR resides in the TARGET directory, and that CHKSRT.OBJ is stored in the library \TARGET\TARGET.LIB)

SUBROUTINE CHKTMS (GOOD)

Purpose: To check downtimes for RVs in the current sortie that go to the same DGZ. If the absolute value of the difference between successive downtimes at each DGZ is greater than or equal to the time between layers, downtimes are considered "good".

Input Arguments:

None

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| GOOD | LOGICAL*1 | Flag: if .TRUE., downtimes are good. |

Modules Called:

GTBLYR

Calling Modules:

CHKMLT

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTRO.FOR | SORTEE | CBSORTIE.FOR |
| OPSHNS | CBOPSHNS.FOR | TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine CHKTMS first sets the logical flag GOOD to .TRUE.; i.e., CHKTMS assumes the times will be good and will try to prove otherwise. Then it begins an outer loop that examines each RV except the last. CHKSRT next begins an inner loop that examines each successor to the RV currently selected in the outer loop. If the RVs go to different DGZs, no further checking is done; CHKSRT proceeds to the next successor, if there is one.

If the RVs do go to the same DGZ, CHKSRT gets the layer numbers associated with the RVs from a list. If the second RV's layer number is less than the first RV's, the two layer numbers are swapped. This step is necessary when last-off-first-in missile systems are used.

If Fratricide Constraints Files ("models") aren't being used, then the operator has supplied a list of times between layers on the Control File's last record. The required time between layers is computed by summing up values in that list, from the first RV's layer up to, but not including the second RV's layer. If Fratricide Constraints Files ARE being used, the required time between layers is computed: the difference between the first and second RV's layers is multiplied by the sure-safe time corresponding to the first RV's height of burst.

CHKTMS computes the absolute value of the difference between the two RVs' downtimes. If that value is less than the required time between layers, the flag GOOD is set to .FALSE. and control returns to

CHKMLT. Otherwise, CHKTMS proceeds to the next RV in the inner or outer loop. If all times are good, control returns to CHKMLT with GOOD still set to .TRUE.

Source File:

ALMOVR2A.FOR

SUBROUTINE CHKWST (WSTFUL)

Purpose: To see if the current sortie is "wasteful". A sortie is wasteful if more than half the RVs are excess RVs. An excess RV is one assigned to a DGZ that's already received its quota of RVs.

Input Arguments:

None

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| WSTFUL | LOGICAL*1 | Flag: if.TRUE., the sortie is wasteful. |

Modules Called:

None

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine CHKWST first zeros a counter of excess RVs. Then it examines the layer number of each RV in the sortie. If a layer number is greater than the number of RVs the associated DGZ was supposed to get, the counter of excess RVs is incremented. Then if the counter is greater than half the number of RVs on the missile, the flag WSTFUL is set to .TRUE.: if not, it's set to FALSE. Then CHKWST returns control to ASR2T2.

Source File:

ALMOVR2A.FOR

SUBROUTINE CROSRT

Purpose: To form a crossrange sortie.

Input Arguments:

None

Output Arguments:

None

Modules Called:

FMAX
SWPSRT

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

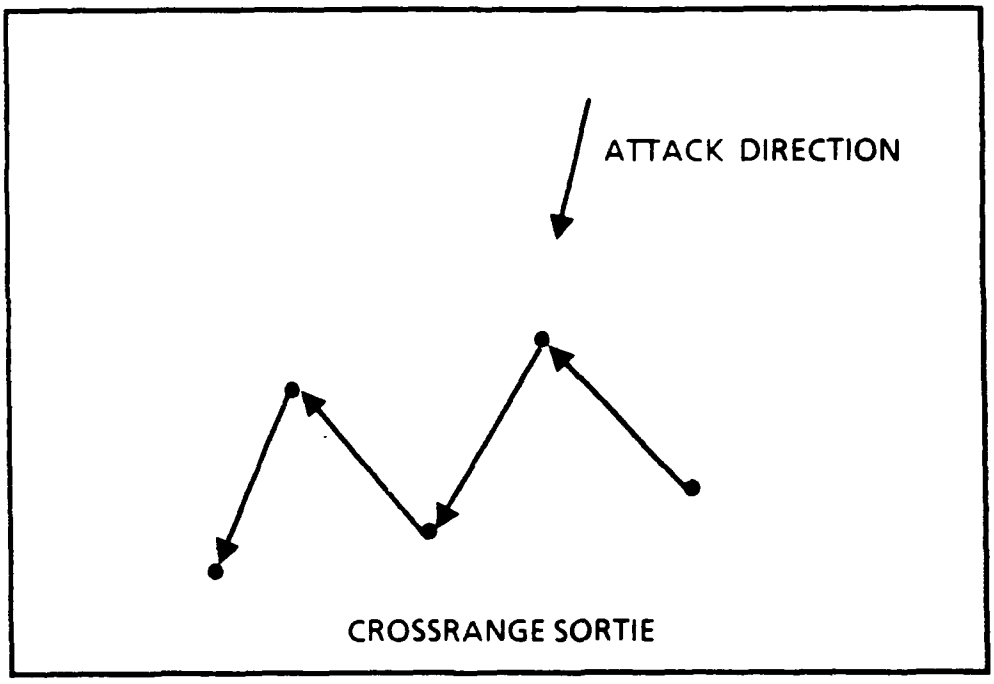
Description:

Subroutine CROSRT simply sorts the lists of sortie data so the Y-coordinates are in descending (right-to-left) order. See Figure 5-13 and the example below. The result is a missile sortie that targets DGZs in the crossrange direction from right to left.

Source File:

ALMOVR2B.FOR

Example:



SUBROUTINE CSTOUT (NDGSRT, DGZSRT)

Purpose: To mark one of the DGZs in the current sortie as an outcast; i.e., to indicate the DGZ should be ignored from now on as permanently unreachable.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| NDGSRT | INTEGER*2 | Number of DGZs (and RVs) in current sortie. |
| DGZSRT | INTEGER*2 | Array containing indices for each DGZ in the sortie. |

Output Arguments:

None

Modules Called:

FNORM
SQUEE2

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| OUTPUT | CBOUTPUT.FOR |
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine CSTOUT looks for the DGZ in the current sortie that's furthest from the target island's centroid. Since the centroid is the origin of the tangent-plane coordinate system, computing distances is easy. CSTOUT also looks for DGZs in the current sortie that are vertices of the convex polygon enclosing the target island, and looks for the furthest one of them too. If CSTOUT finds one or more DGZs that are vertices, the furthest one of them is cast out. If none of the DGZs in the sortie is a vertex, then the ordinary DGZ that's furthest from the centroid is cast out.

Source File:

ALMOVR2B.FOR

SUBROUTINE CTRVS (IDSTRT, NEEDED)

Purpose: To count the total number of RVs that still must be assigned to a subset of the target island. The subset consists of all DGZs to the left of, and including, the starting DGZ.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IDSTRT | INTEGER*2 | Pointer to list of sorted DGZ indices. Indicates where to start. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| NEEDED | INTEGER*2 | Number of RVs needed. |

Modules Called:

None

Calling Modules:

ASR2T2
GETSRT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine CTRVS zeros the counter of RVs needed and executes a loop that examines DGZs. Outcasts are ignored. For each DGZ, CTRVS subtracts the number of RVs assigned so far from the number of RVs allocated to the DGZ in the input RV Allocation File. If the difference is greater than zero, it's added to the counter of RVs needed. Then CTRVS returns control to ASR2T2 or GETSRT.

Source File:

ALMOVR2B.FOR

SUBROUTINE CTUNIQ (NDGZS, DGZSRT, NUNEEK)

Purpose: To count the number of unique DGZs in a sortie.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| NDGZS | INTEGER*2 | The number of DGZs in the sortie. |
| DGZSRT | INTEGER*2 | Array: contains the index of each DGZ in the sortie. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| NUNEEK | INTEGER*2 | Number of unique DGZs. |

Modules Called:

None

Calling Modules:

ASR2T2
GETSRT
MULSRT

Common Blocks Used:

None

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine CTUNIQ initially defines the number of unique DGZs in the sortie to be one. Then CTUNIQ executes a loop that ranges from the second through the last DGZ in the sortie. During each pass, CTUNIQ sees if the current DGZ appears earlier in the sortie. If so, the DGZ isn't unique and CTUNIQ proceeds to the next DGZ. But if the DGZ is unique, the number of unique DGZs in incremented. Then CTUNIQ returns control to the calling module.

Source File:

ALMOVR2B.FOR

SUBROUTINE CTUNUS (NUNUSD)

Purpose: To count the number of unused missiles in the launch complex.

Input Arguments:

None

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| NUNUSD | INTEGER*2 | Number of unused missiles. |

Modules Called:

None

Calling Modules:

MOPUP

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine CTUNUS initially defines the number of unused missiles to be zero. Then CTUNUS executes a loop that ranges from one through the number of missiles it's supposed to use. During each pass, CTUNUS examines a flag associated with each missile that indicates whether or not the missile has been used yet. If the flag is. FALSE., meaning the missile hasn't been used yet, the number of unused missiles is incremented. Then CTUNUS returns control to the calling module.

Source File:

ALMOVR2B.FOR

SUBROUTINE DWNSRT (FIRST, LAST)

Purpose: To form a downrange sortie from a subset of the current sortie.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| FIRST | INTEGER*2 | Index for first DGZ in sortie lists. |
| LAST | INTEGER*2 | Index for last DGZ in sortie lists. |

Output Arguments:

None

Modules Called:

FMIN
SWPSRT

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

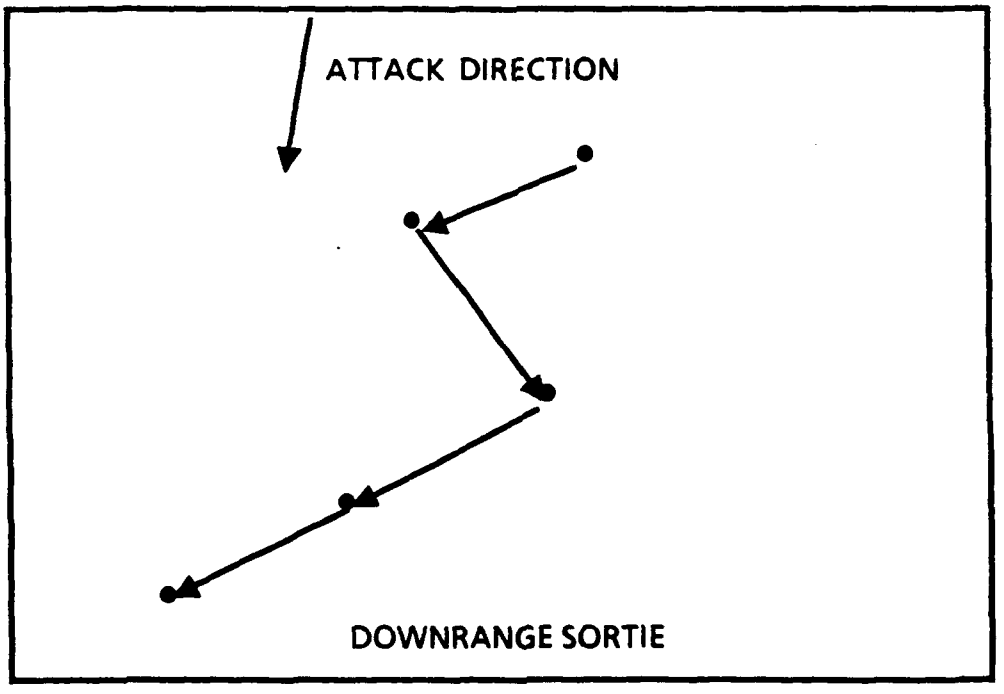
Description:

Subroutine DWNSRT simply sorts the specified sublists of sortie data so the X-coordinates are in ascending (downrange) order. See Figure 5-13 and the figure below. The result is a missile sortie that targets DGZs in the downrange direction.

Source File:

ALMOVR2B.FOR

Example:



SUBROUTINE FMABAB

Purpose: To form a sortie that moves uprange, returns to the beginning, and moves uprange again.

Input Arguments:

None

Output Arguments:

None

Modules Called:

ADDGZ2
SQUEEZ

Calling Modules:

CHKMLT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

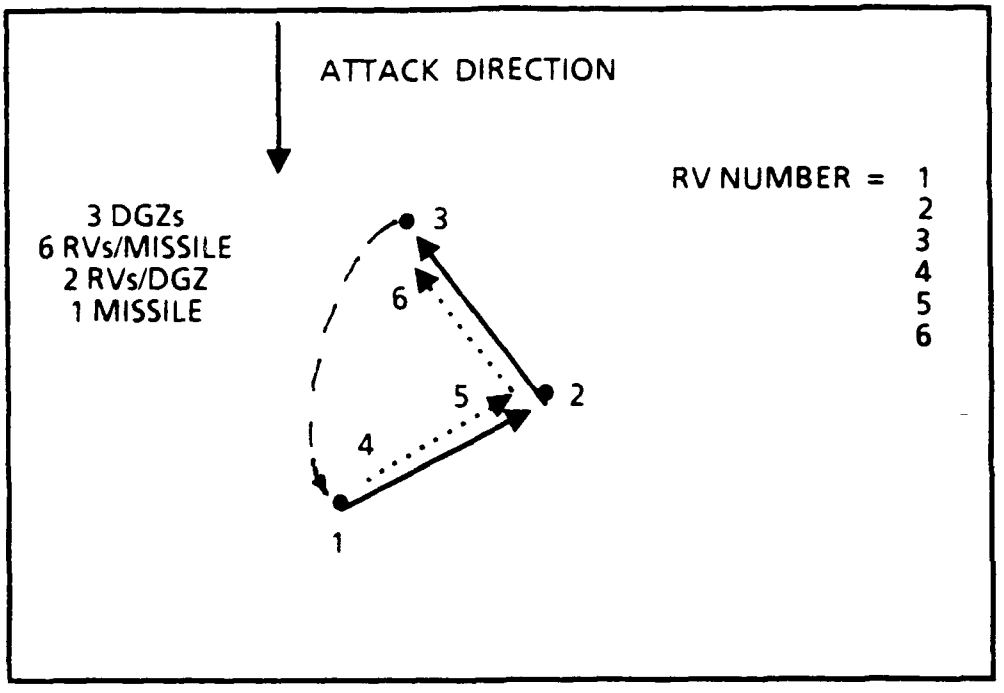
Subroutine FMABAB calls subroutine SQUEEZ to form a list of unique DGZ numbers, a list of the number of times each DGZ appears in the sortie, and a list of the starting layer number for each DGZ. Then FMABAB prepares the index used to form a new sortie: the index is set to one.

FMABAB begins a loop that ranges over the list of unique DGZs. During each pass, FMABAB sees if the number of times the DGZ appears in the sortie is greater than zero. If so, the DGZ is added to the sortie by calling subroutine ADDGZ2 and the sortie index is incremented. The number of times the DGZ appears in the input sortie is decremented. If the sortie is completed during a pass through the loop, control is returned to CHKMLT. Otherwise, FMABAB proceeds to the next unique DGZ in the list. If all unique DGZs have been examined and the sortie is still incomplete, FMABAB goes back to the beginning of the loop and starts over.

Source File:

ALMOVR2B.FOR

Example:


ATTACK DIRECTION
3 DGZs
6 RVs/MISSILE
2 RVs/DGZ
1 MISSILE
RV NUMBER =
1
2
3
4
5
6
1
2
3
4
5
6

SUBROUTINE FMABBA

Purpose: To form a sortie that moves uprange, turns around, and then moves downrange.

Input Arguments:

None

Output Arguments:

None

Modules Called:

ADDGZ2
SQUEEZ

Calling Modules:

CHKMLT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTIE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine FMABBA calls subroutine SQUEEZ to form a list of unique DGZ numbers, a list of the number of times each DGZ appears in the sortie, and a list of the starting layer number for each DGZ. Then FMABBA prepares the index used to form a new sortie: the index is set to one.

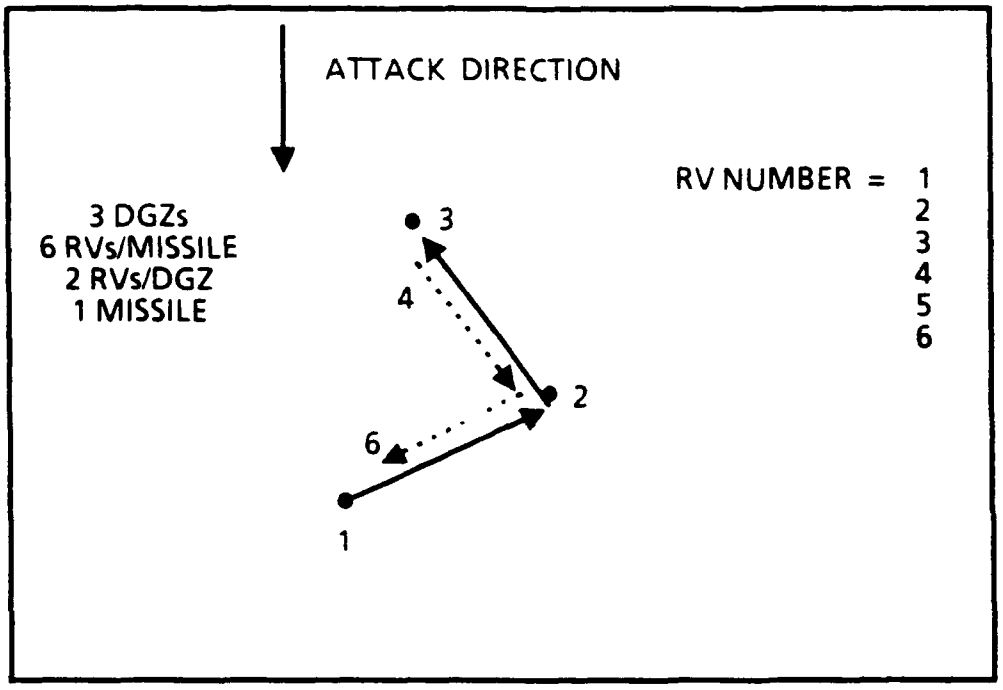
FMABBA prepares to move uprange by setting the flag UPRNGE to .TRUE., the starting DGZ index to 1, the ending DGZ index to the number of unique DGZs, and the loop increment to + 1.

FMABBA begins a loop that ranges (forward or backward) over the list of unique DGZs. During each pass, FMABBA sees if the number of times the DGZ appears in the sortie is greater than zero. If so, the DGZ is added to the sortie by calling subroutine ADDGZ2, and the sortie index is incremented. The number of times the DGZ appears in the input sortie is decremented. If the sortie is completed during a pass through the loop, control is returned to CHKMLT. Otherwise, FMABBA proceeds to the next unique DGZ in the list. If all unique DGZs have been examined and the sortie is still incomplete, FMABBA reverses the direction in which it scans the list of unique DGZs, goes back to the beginning of the loop, and starts over.

Source File:

ALMOVR2B.FOR

Example:


ATTACK DIRECTION
3 DGZs
6 RVs/MISSILE
2 RVs/DGZ
1 MISSILE
RV NUMBER = 1
2
3
4
5
6
3
4
2
6
1

SUBROUTINE FNFRTH (YMEAN, ISFUR)

Purpose: To find the DGZ in the current sortie that's furthest from the sortie's vertical midline.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| YMEAN | REAL*4 | Mean of the sortie's leftmost and rightmost DGZs. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| ISFUR | INTEGER*2 | Index of furthest point. |

Modules Called:

None

Calling Modules:

MULSRT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine FNFRTH sets the furthest distance found so far to an absurdly small value. Then FNFRTH executes a loop that computes the absolute value of the difference between each DGZ's Y-coordinate and the mean Y-coordinate. If the value is greater than the current maximum, the maximum is updated, and the DGZ's index is stored in ISFUR. After all DGZs have been checked, control returns to MULSRT.

Source File:

ALMOVR2B.FOR

SUBROUTINE FNHIGH (HIGEST)

Purpose: To find the highest layer number among all RVs attacking the target island.

Input Arguments:

None

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| HIGEST | INTEGER*2 | Highest layer number. |

Modules Called:

None

Calling Modules:

ADJST1

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine FNHIGH sets the highest layer found so far to an absurdly small value. Then FNHIGH executes an outer loop that examines each missile used and an inner loop that examines the layer number for each RV on the missile. If the layer number is higher than the highest layer number found so far, the highest layer number is updated. After all RVs have been checked, control returns to ADJST1.

Source File:

ALMOVR2B.FOR

SUBROUTINE FNPREV (IDGZ, IPL, JR, JM)

Purpose: To find the RV in the previous layer that hit a specified DGZ, and return its missile number and order in the missile.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IDGZ | INTEGER*2 | DGZ's index. |
| IPL | INTEGER*2 | Previous layer number. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| JR | INTEGER*2 | RV index; i.e., order within missile. |
| JM | INTEGER*2 | Missile index. |

Modules Called:

ABRTRN
SQUEE2
WRSFMS

Calling Modules:

ADJST1

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| OUTPUT | CBOUTPUT.FOR |
| WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine FNPREV executes an outer loop that examines each missile used and an inner loop that examines each RV on the missile. If the DGZ attacked by the RV is the one specified by IDGZ, and if the RV's layer number equals IPL, then control returns to ADJST1. Otherwise the search continues. If the RV can't be found, this is considered a fatal error. FNPREV writes a warning message in the status file, and two information messages indicating the DGZ and previous layer number. Then it calls ABRTRN.

Source File:

ALMOVR2B.FOR

SUBROUTINE FNVRTS

Purpose: To determine which DGZs are the vertices of a convex polygon enclosing the target island.

Input Arguments:

None

Output Arguments:

None

Modules Called:

FCVPOL
PRPNTS

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine FNVRTS executes a loop that sets a logical flag associated with each DGZ to .FALSE.; these flags indicate none of the DGZs is a vertex. FNVRTS checks the number of DGZs in the target island. If there's only one, its flag is set to .TRUE., indicating it's the (only) vertex, and control returns to ASR2TI. Otherwise, FNVRTS calls PRPNTS to project the DGZs onto a plane tangent to the earth at the target island's centroid. Then FNVRTS calls FCVPOL to form a list of DGZ indices. Each DGZ in this list is a vertex of the convex polygon enclosing the set of DGZs. Finally, FNVRTS executes a loop that examines each DGZ index in the list and sets the corresponding DGZ's flag to .TRUE., indicating the DGZ is a vertex.

Source File:

ALMOVR2B.FOR

SUBROUTINE FXLYRS

Purpose: To fix the layer numbers for each string of RVs attacking the same DGZ. Layer numbers were assigned to RVs with no regard whatsoever for flight times, so it's possible for one RV assigned to a DGZ to have an EARLIER downtime than an RV in a previous layer. FXLYRS ensures that in each string of RVs, the downtime for layer N is later than the downtime for layer N-1. (N = 2, 3, 4, ...).

Input Arguments:

None

Output Arguments:

None

Modules Called:

ORDER

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| TI | CBTI.FOR |
| WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine FXLYRS executes a loop that examines each DGZ in turn. If the number of RVs assigned so far (i.e., number of layers) is greater than 1, FXLYRS prepares three lists. It does so by executing a pair of nested loops: the outer loop examines each missile used and the inner examines each RV on the missile. For each RV attacking the current DGZ, FXLYRS saves the RV index (order within missile), missile index, and downtime in the lists. When done, the number of entries in each list equals the number of layers attacking the DGZ.

Then FXLYRS calls ORDER to form a list of pointers so RV data can be accessed in ascending order of downtime. FXLYRS executes a loop over layer numbers that puts the RV indices, missile indices, and downtimes back in their original arrays, but this time the layer numbers and downtimes are in ascending order.

Source File:

ALMOVR2C.FOR

SUBROUTINE GETSRT (ID, NRVS)

Purpose: To get DGZs for a strict uprange sortie or simple multi-layer sortie.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| ID | INTEGER*2 | Pointer to ordered list of DGZ indices. |
| NRVS | INTEGER*2 | Number of RVs on missile that will perform sortie. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| ID | INTEGER*2 | Pointer to ordered list of DGZ indices. Input argument may be changed. |

Modules Called:

| | | |
|---|---|---|
| ABRTRN | ADXTRA | SQUEE2 |
| ADDGZ1 | CTRVS | SRTSRT |
| ADLAYR | CTUNIQ | WRSFMS |

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | SORTEE | CBSORTIE.FOR |
| OUTPUT | CBOUTPUT.FOR | TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Numbers in parentheses are statement numbers in the source code.

(10) Subroutine GETSRT calls CTRVS to count the number of RVs still needed by DGZs that aren't outcasts. Counting starts with the DGZ pointed to by ID and sweeps left. If the number of RVs needed is greater than half the number of RVs on a missile, GETSRT goes to statement 30 to prepare a sortie index. If not, GETSRT checks to see if counting RVs started with the rightmost DGZ (ID = 1). If so, GETSRT goes to statement 30 to prepare a sortie index. If not, GETSRT resets ID to 1, and goes back to statement 10, to count RVs across the entire target island.

(30) GETSRT prepares an index used to store sortie data in several lists.

(35) GETSRT examines the current DGZ. If it's an outcast or it has already gotten its full quota of RVs, GETSRT goes to statement 50 to increment the DGZ pointer. If not, GETSRT calls ADDGZ1 to add the DGZ to the sortie, and increments the sortie index.

(50) GETSRT increments the DGZ index (i.e., moves left). If the sortie is complete, GETSRT goes to statement 60 to check the number of entries in the sortie lists. If not, GETSRT checks the DGZ pointer to see if it's past the target island's left edge. If not, GETSRT goes back to statement 35 to examine the new DGZ. Otherwise, GETSRT resets the DGZ pointer to the rightmost DGZ and calls ADLAYR to add RVs from

subsequent layers, to use up the missile load. If that's not possible, GETSRT call ADXTRA to add extra RVs to the most valuable DGZs in the sortie. Then GETSRT goes to statement 65 to sort the lists of sortie data.

(60) If the number of DGZs in the sortie doesn't equal the number of RVs on the missile, something is terribly wrong. GETSRT calls WRSFMS and SQUEE2 repeatedly to write a warning message in the Status File, and calls ABRTRN to abort the run.

(65) GETSRT calls SRTSRT to sort the lists of sortie data, so DGZs are in uprange order. GETSRT calls CTUNIQ to count the number of unique DGZs in the sortie and returns control to ASR2T2.

Source File:

ALMOVR2C.FOR

SUBROUTINE GTBLY2 (IM, TBLYRS)

Purpose: To get the time between layers from a list of sure-safe times read from the Fratricide Constraints Files.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IM | INTEGER*2 | Missile index. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| TBLYRS | INTEGER*2 | Time between current and next layer, in seconds. |

Modules Called:

None

Calling Modules:

ADJST1

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | TI | CBTI.FOR |
| OPSHNS | CBOPSHNS.FOR | WEAPON | CBWEAPON.FOR |
| OUTPUT | CBOUTPUT.FOR | | |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

GTBLY2 executes a simple loop that examines each DGZ in the sortie performed by the specified missile. GTBLY2 uses the height of burst index associated with each DGZ to get the largest sure-safe time. This maximum is returned as the time between layers.

Source File:

ALMOVR2C.FOR

SUBROUTINE GTBLYR (LAYER, TBLYRS)

Purpose: To get the time between the current layer and the next from a list supplied in the Control File.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| LAYER | INTEGER*2 | Current layer number. |

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| TBLYRS | INTEGER*2 | Time between current and next layer, in seconds. |

Modules Called:

ABRTRN
SQUEE2
WRSFMS

Calling Modules:

ADJST1
CHKTMS

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| CNTROL | CBCNTROL.FOR |
| OPSHNS | CBOPSHNS.FOR |
| OUTPUT | CBOUTPUT.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

If the layer number is bad, GTBLYR calls WRSFMS and SQUEE2 to write a warning message in the Status File. Then GTBLYR calls ABRTRN to abort the run. Otherwise, GTBLYR uses the current layer number as an index to get the desired time between layers from the list, and then returns control to ADJST1 or CHKTMS.

Source File:

ALMOVR2C.FOR

SUBROUTINE MOPUP

Purpose: To assign RVs on any remaining missile(s) to DGZs in the target island.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ABRTRN | MOPUPB |
| CTUNUS | MOPUPC |
| MOPUPA | WRSFMS |

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

MOPUP calls CTUNUS to count the unused missiles in the launch complex. If all missiles, have been used, MOPUP returns control to ASR2T2. If not, MOPUP calls MOPUPA to try to assign RVs to DGZs that didn't get any RVs at all.

MOPUP calls CTUNUS again to count the unused missiles in the launch complex. If all missiles, have been used, MOPUP returns control to ASR2T2.

If not, MOPUP checks to see if any missiles have been used at all, since the next two MOPUP techniques require at least one previous sortie. If no missiles have been used at all, MOPUP calls WRSFMS to write a warning message in the Status File, and then calls ABRTRN to abort the run.

Otherwise, MOPUP calls MOPUPB to try to assign RVs to sorties with the greatest number of short-changed DGZs; i.e., DGZs that got fewer RVs than they were supposed to get (as prescribed in the input RV Allocation File).

MOPUP calls CTUNUS again to count the unused missiles in the launch complex. If all missiles, have been used, MOPUP returns control to ASR2T2. Otherwise, MOPUP calls MOPUPC to assign RVs on unused missiles to the most valuable sorties, and then returns control to ASR2T2.

Source File:

ALMOVR2C.FOR

SUBROUTINE MOPUPA

Purpose: To assign RVs on an unused missile to DGZs in the target island. MOPUPA picks DGZs that didn't get any RVs at all.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ADDGZ1 | SRTSRT |
| CHKMLT | STRSRT |
| SRTSR3 | UNDO |

Calling Modules:

MOPUP

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDED |
|---|---|---|---|
| LC | CBLC.FOR | TI | CBTI.FOR |
| SORTEE | CBSORTIE.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

MOPUPA tries to form a sortie from DGZs that didn't get any RVs at all (outcasts are ignored as usual). If there aren't any such DGZs, MOPUPA returns control to MOPUP.

MOPUPA sorts the data in the sortie lists so DGZ values are in ascending order. Then it cycles through the DGZs (backwards), adding them to the sortie, until all RVs on the unused missile have been used. MOPUPA sorts the sortie lists to form an uprange sortie. The first unused missile is picked and its launch time computed. MOPUPA calls CHKMLT to see if the missile can perform the sortie. If so, MOPUPA calls STRSRT to store the sortie data in arrays. If not, MOPUPA calls UNDO to undo the sortie just found. Then MOPUPA returns control to MOPUP.

Source File:

ALMOVR2C.FOR

SUBROUTINE MOPUPB

Purpose: To assign RVs on any remaining missile(s) to DGZs in the target island. MOPUPB repeats the sortie with the greatest number of short-changed DGZs; i.e., DGZs that got fewer RVs than they were supposed to get (as prescribed in the input RV Allocation Fiel).

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ADDGZ1 | STRSRT |
| CHKSRT | UNDO |

Calling Modules:

MOPUP

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDED |
|---|---|---|---|
| LC | CBLC.FOR | TI | CBTI.FOR |
| OPSHNS | CBOPSHNS.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

MOPUPB executes a loop over all missiles that were supposed to be used, looking for one that hasn't been used yet. If it finds one, it begins another loop over all missiles used, looking for one whose sortie has the greatest number of short-changed DGZs. If it can't find one, MOPUPB returns control to MOPUP. Otherwise, MOPUPB duplicates the sortie and calls CHKSRT to see if the current unused missile can perform the sortie. If so, MOPUPB calls STRSRT to store the sortie data in arrays: if not MOPUPB calls UNDO to undo the sortie just formed. Then MOPUPB makes another pass through the outer loop, looking for the next unused missile.

Source File:

ALMOVR2C.FOR

SUBROUTINE MOPUPC

Purpose: To assign RVs on any remaining missile(s) to DGZs in the target island. MOPUPC repeats the most valuable sortie; i.e., the sortie with the most valuable DGZs.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ABRTRN | STRSRT |
| ADDGZ1 | UNDO |
| CHKSRT | WRSFMS |

Calling Modules:

MOPUP

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | TI | CBTI.FOR |
| OPSHNS | CBOPSHNS.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

MOPUPC first sets a logical flag associated with each missile to .FALSE., indicating the missile's sortie hasn't been repeated. MOPUPC also defines the current pass through the sortie to be the first.

MOPUPC executes a loop over all missiles, looking for one that hasn't been used yet. If it finds one, MOPUPC looks for the most valuable sortie that hasn't already been repeated during the current pass. If MOPUPC can't find a sortie to repeat during the first pass, something is very wrong: MOPUPC calls WRSFMS to write a warning message and calls ABRTRN to abort the run. Otherwise, MOPUPC duplicates the sortie and calls CHKSRT to see if the unused missile can perform the sortie. If so, MOPUPC calls STRSRT to store the sortie data in arrays: if not, MOPUPC calls UNDO to undo the sortie. After trying to duplicate successively less valuable sorties for any unused missiles, MOPUPC returns control to MOPUP.

Source File:

ALMOVR2C.FOR

SUBROUTINE MOVLFT (ID)

Purpose: To move the DGZ pointer immediately to the left of the leftmost DGZ in the current sortie.

Input Arguments:

None

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| ID | INTEGER*2 | DGZ pointer. |

Modules Called:

| | |
|---|---|
| ABRTRN | SQUEE2 |
| FMIN | WRSFMS |

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDED |
|---|---|---|---|
| LC | CBLC.FOR | SORTEE | CBSORTIE.FOR |
| OUTPUT | CBOUTPUT.FOR | TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

MOVLFT calls FMIN to find the DGZ in the current sortie with the smallest (i.e., leftmost) Y-coordinate. Then MOVLFT searches the list of DGZs, in descending order of their Y-coordinates, to find a match. If no match is found, something is very wrong: MOVLFT calls WRSFMS to write a warning message in the Status File, and calls ABRTRN to abort the run. Otherwise, MOVLFT increments the DGZ pointer to move left one more DGZ. If this wraps around the left edge of the target island, the pointer is reset to the right edge. Then MOVLFT returns control to ASR2T2.

Source File:

ALMOVR2C.FOR

SUBROUTINE MULSRT

Purpose: To form a multi-layer sortie from an uprange sortie by reassigning one DGZ's RVs to other DGZs in the sortie.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ADDGZ1 | FNFRTH |
| CPCNTR | SQUEEZ |
| CTUNIQ | |

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

MULSRT calls CPCNTR to compute the sortie's mean Y-coordinate, which establishes the sortie's vertical midline. Then MULSRT calls FNFRTH to find the DGZ in the sortie that's furthest from the vertical midline.

MULSRT counts the number of times the furthest DGZ appears in the sortie; that's the number of RVs that have to be reassigned. Then MULSRT goes through the lists of sortie data, deleting all values associated with the furthest DGZ. MULSRT decrements the number of RVs assigned so far to the furthest DGZ.

MULSRT calls CTUNIQ to count the unique DGZs in the modified sortie and sets a logical flag associated with each unique DGZ to .FALSE., to indicate the DGZ hasn't been reassigned an RV from the deleted DGZ. Then MULSRT executes a loop that reassigns RVS to the DGZs remaining in the sortie. At first, if there are DGZs that still need RVs, RVs are reassigned to the LEAST valuable DGZs. Later, if all DGZs have received their full quota, RVs are reassigned to the MOST valuable DGZs.

Source File:

ALMOVR2C.FOR

SUBROUTINE ORDDGS

Purpose: To order DGZs; i.e., to form a list of pointers so DGZs can be accessed from right to left.

Input Arguments:

None

Output Arguments:

None

Modules Called:

ORDER
PRPNTS

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

ORDDGS called PRPNTS to project the DGZs onto a plane tangent to the earth at the target island's centroid. ORDDGS calls ORDER to form the list of pointers required: by ordering DGZs with their Y-coordinates in descending order, the DGZs can be accessed from right to left. See Figure 5-13.

Source File:

ALMOVR2C.FOR

SUBROUTINE ORDMSS

Purpose: To order missiles; i.e., to form a list of pointers that controls the order in which they're used. A horizontal strip of missiles is used for each layer, and strips are used from front to back. Within a strip, missiles are used from right to left. See Figure 5-13.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ABRTRN | PRPNTS |
| ISWAP | RMLEAD |
| ORDER | SQUEE2 |
| PLURAL | WRSFMS |

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | TI | CBTI.FOR |
| OUTPUT | CBOUTPUT.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Numbers in parentheses refer to statement numbers in the source code.

ORDMSS calls PRPNTS to project the missile silos onto a plane tangent to the earth at the launch complex's centroid. See Figure 5-13. ORDMSS tinkers with signs so Y-coordinates decrease from right to left just like DGZ Y-coordinates. X-coordinates decrease from front to back. ORDMSS calls ORDER to form a list of pointers, so silos can be accessed from front to back. ORDMSS zeros the number of RVs assigned to each DGZ so far and sets the index for the first missile in the first layer to one.

(20) ORDMSS counts the number of DGZs that need at least one more RV. The result is the number of RVs needed for the current layer. If it's less than the number of RVs on a missile, ORDMSS goes to statement 80 to define the number of missiles and RVs to use in the current layer.

Otherwise, ORDMSS divides the number of RVs in the current layer by the number of RVs on a missile, yielding the number of full missile loads (i.e., missiles) in the current layer. ORDMSS computes the index for the last missile in the current layer. If that index is greater than the number of missiles in the launch complex, something is wrong. ORDMSS calls WRSFMS and SQUEE2 to write a warning message in the Status File, and then calls ABRTRN to abort the run.

Otherwise, ORDMSS sorts the pointers for the current layer so missiles will be used from right to left. Then ORDMSS computes the number of RVs in the current layer (number of missiles in current layer times number of RVs per missile). ORDMSS executes a loop over all the DGZs in the target island, sweeping from right to left, and assigning an RV to each DGZ that still needs one. If all RVs in the current layer are used up before examining all DGZs, ORDMSS goes to statement 60 immediately, rather than continuing to statement 60 after examining all DGZs.

(60) ORDMSS computes the index for the first missile in the next layer (immediately follows the last missile in the current layer), and goes back to statement 20 to count DGZs again.

(80) If none of the DGZs need an RV, the total number of missiles used is simply one less than the index for the first missile of the current (empty) layer. Otherwise, the total number is the index of the first missile in the current layer. The total number of RVs used is the product of the total number of missiles used and the number of RVs per missile. ORDMSS writes an information message in the Status File stating how many missiles and RVS will be used. Finally, ORDMSS resets the number of RVs assigned to each DGZ so far to zero and returns control to ASR2TI.

Source File:

ALMOVR2C.FOR

SUBROUTINE PREPAR

Purpose: To prepare data for the Attack File.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ABRTRN | SQUEE2 |
| FMIN | SWAP |
| ISWAP | WRSFMS |

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| ATTACK | CBATTACK.FOR | TI | CBTI.FOR |
| LC | CBLC.FOR | WEAPON | CBWEAPON.FOR |
| OUTPUT | CBOUTPUT.FOR | | |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

PREPAR copies data for each missile used to a set of one-dimensional arrays, so data can be sorted in ascending order of launch time. If no missiles were used, PREPAR calls WRSFMS and SQUEE2 to write a warning message in the Status File, and then calls ABRTRN to abort the run. Otherwise, PREPAR calls FMIN, ISWAP, and SWAP to sort the missile data. PREPAR gets the first and last launch times from the ordered list of launch times and computes the launch duration.

PREPAR copies data for each RV on each missile used to a set of one-dimensional arrays, so data can be sorted in ascending order of downtime. The probability of attrition for each RV is set to zero. If arrays are somehow empty, PREPAR calls WRSFMS and SQUEE2 to write a warning message in the Status File, and then calls ABRTRN to abort the run. Otherwise, PREPARE calls FMIN, ISWAP, and SWAP to sort the RV data. PREPAR gets the first and last downtimes from the ordered list of downtimes and computes the laydown duration.

Finally, PREPAR zeros every entry in the array containing probabilities of fratricide and returns control to ALM.

Source File:

ALMOVER3.FOR

SUBROUTINE RALLOC

Purpose: To read the RV Allocation File.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| CPCNTR | RMLEAD |
| ORDNAL | TRULEN |
| PLURAL | WRSFMS |

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | TI | CBTI.FOR |
| CSTRNG | CBCSTRNG.FOR | TSTRNG | CBTSTRNG.FOR |
| OUTPUT | CBOUTPUT.FOR | | |

File Input/Output:

| File | Unit No. | Description |
|---|---|---|
| "case".RVA | 4 | RV Allocation File |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

RALLOC reads the RV allocation data and checks for errors. If an error is serious, RALLOC writes a warning message in the Status File and either calls ABRTRN to abort the run, or stops immediately. If an error isn't serious, RALLOC writes a warning message and continues.

After reading the entire file successfully, RALLOC computes the normalized value of each DGZ in the target island, and the number of heights of burst used in the attack.

Source File:

ALMOVR1A.FOR

SUBROUTINE RBOOST

Purpose: To read the Booster Data File record corresponding to the missile system to be used in the laydown.

Input Arguments:

None

Output Arguments:

None

Modules Called:

SQUEE2
TRULEN
WRSFMS

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | OUTPUT | CBOUTPUT.FOR |
| CSTRNG | CBCSTRNG.FOR | TI | CBTI.FOR |
| LC | CBLC.FOR | TSTRNG | CBTSTRNG.FOR |
| OPSHNS | CBOPSHNS.FOR | | |

File Input/Output:

| File | Unit No. | Description |
|---|---|---|
| MPS.BST | 4 | Booster Data File |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

RBOOST reads a record in the Booster Data File and checks for errors. If an error occurs, RBOOST writes a warning message in the Status File and stops immediately.

After reading the record successfully, RBOOST defines the previous launch time to be 0.0 minus the time between launches. RBOOST also divides the number of RVs in the attack by the number of missiles in the attack and compares the quotient with the number of RVs per weapon. If the two values aren't equal, RBOOST calls WRSFMS and SQUEE2 to write a warning message, and then stops.

If the values agree, RBOOST computes the number of RVs at the launch complex and returns control to ALM.

Source File:

ALMOVR1A.FOR

SUBROUTINE RCNTRL

Purpose: To read the Control File.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ORDNAL | SUM |
| RMBLKS | WRSFFR |
| RMLEAD | WRSFMS |
| SQUEE2 | |

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | OPSHNS | CBOPSHNS.FOR |
| CSTRNG | CBCSTRNG.FOR | OUTPUT | CBOUTPUT.FOR |

File Input/Output:

| File | Unit No | Description |
|---|---|---|
| "case".CF4 | 5 | Control File. EXEC/MMI is expected to redirect the standard input (logical unit 5 or *) so input comes from the Control File rather than the keyboard. |
| - | 6 | If errors occur before the Status File is opened, error messages will be displayed on the screen. |
| "case".SF4 | 10 | Status File. RCNTRL opens the Status File by calling WRSFFR, specifying the Status File should be connected to logical unit 10. |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

RCNTRL reads records in the Control File and checks for errors. If errors occur before the Status File is opened, RCNTRL displays a message on the screen and stops immediately. If the case name and description are read successfully from the Control File's first record, the Status File is opened by calling WRSFFR. Thereafter, if errors occur, RCNTRL writes warning and error messages in the Status File and stops immediately.

If RCNTRL reads the entire file with no errors, it writes an information message in the Status File and returns control to ALM. If one or more errors occurred, RCNTRL writes a warning message in the Status File and stops.

Source File:

ALMOVR1A.FOR

SUBROUTINE RLAUNC

Purpose: To read the Launch Location File.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| CMEAN | RMLEAD |
| COCNTR | SQUEE2 |
| ORDNAL | TRULEN |
| PLURAL | WRSFMS |

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | OPSHNS | CBOPSHNS.FOR |
| CSTRNG | CBCSTRNG.FOR | OUTPUT | CBOUTPUT.FOR |
| LC | CBLC.FOR | TI | CBTI.FOR |
| LSTRNG | CBLSTRNG.FOR | | |

File Input/Output:

| File | Unit No. | Description |
|---|---|---|
| "case".LLF | 4 | Launch Location File. |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

RLAUNC reads records in the Launch Location File and checks for errors. If RLAUNC reads the entire file with no errors, it computes the launch complex's centroid and the mean silo elevation (altitude). Then RLAUNC writes an information message in the Status File and returns control to ALM. If one or more errors occur, RLAUNC writes a warning message in the Status File and stops.

Source File:

ALMOVR1B.FOR

SUBROUTINE RPBVFI

Purpose: To read the PBV Data File record corresponding to the missile system to be used in the laydown.

Input Arguments:

None

Output Arguments:

None

Modules Called:

SQUEE2
TRULEN
WRSFMS

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | OUTPUT | CBOUTPUT.FOR |
| CSTRNG | CBCSTRNG.FOR | WEAPON | CBWEAPON.FOR |
| OPSHNS | CBOPSHNS.FOR | | |

File Input/Output:

| File | Unit No | Description |
|---|---|---|
| MPS.PBV | 4 | PBV Data File |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

RPBVFI reads a record in the PBV Data File and checks for errors. If an error occurs, RPBVFI writes a warning message in the Status File and stops immediately.

After reading the record successfully, RPBVFI defines the number of RVs on each PBV. RPBVFI also defines the number of the RV on each missile that's the first to arrive (currently it's either the first or the last). RPBVFI calls WRSFMS to write an information message in the Status File and returns control to ALM.

Source File:

ALMOVR1B.FOR

SUBROUTINE RSSAFE

Purpose: To read the sure-safe time from each Fratricide Constraints File.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| ABRTRN | TRULEN |
| ORDNAL | WRSFMS |
| SQUEE2 | |

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | OUTPUT | CBOUTPUT.FOR |
| CSTRNG | CBCSTRNG.FOR | TI | CBTI.FOR |
| OPSHNS | CBOPSHNS.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

| File | Unit No. | Description |
|---|---|---|
| xxx.FCF | 4 | Fratricide Constraints File. The string "xxx" represents the file's name. Up to two such files can be read by ALM. |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

RSSAFE first sets the sure-safe time associated with each height of burst (HOB) to an absurd value (-1.0). Then RSSAFE begins a loop over each Fratricide Constraints File. If errors or premature ends-of-file occur, RSSAFE writes one or more messages in the Status File and calls ABRTRN to abort the run. Otherwise, RSSAFE reads the HOB and sure-safe time from each file. RSSAFE searches the list of HOBs read from the RV Allocation File for an entry that matches (within 1%) the HOB read from the current Fratricide Constraints File. If such a match is found, the sure-safe time is stored in a corresponding list.

After reading all the Fratricide Constraints Files, RSSAFE checks each sure-safe time. If one of them is -1.0 (the absurd value originally stored in the list), RSSAFE writes a warning message and calls ABRTRN to abort the run. Otherwise, RSSAFE writes an information message and returns control to ALM.

Source File:

ALMOVR1B.FOR

SUBROUTINE RRVFIL

Purpose: To read the RV Data File record corresponding to the missile system to be used in the laydown.

Input Arguments:

None

Output Arguments:

None

Modules Called:

SQUEE2
TRULEN
WRSFMS

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | OUTPUT | CBOUTPUT.FOR |
| CSTRNG | CBCSTRNG.FOR | WEAPON | CBWEAPON.FOR |
| OPSHNS | CBOPSHNS.FOR | | |

File Input/Output:

| File | Unit No. | Description |
|---|---|---|
| MPS.RV | 4 | RV Data File |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

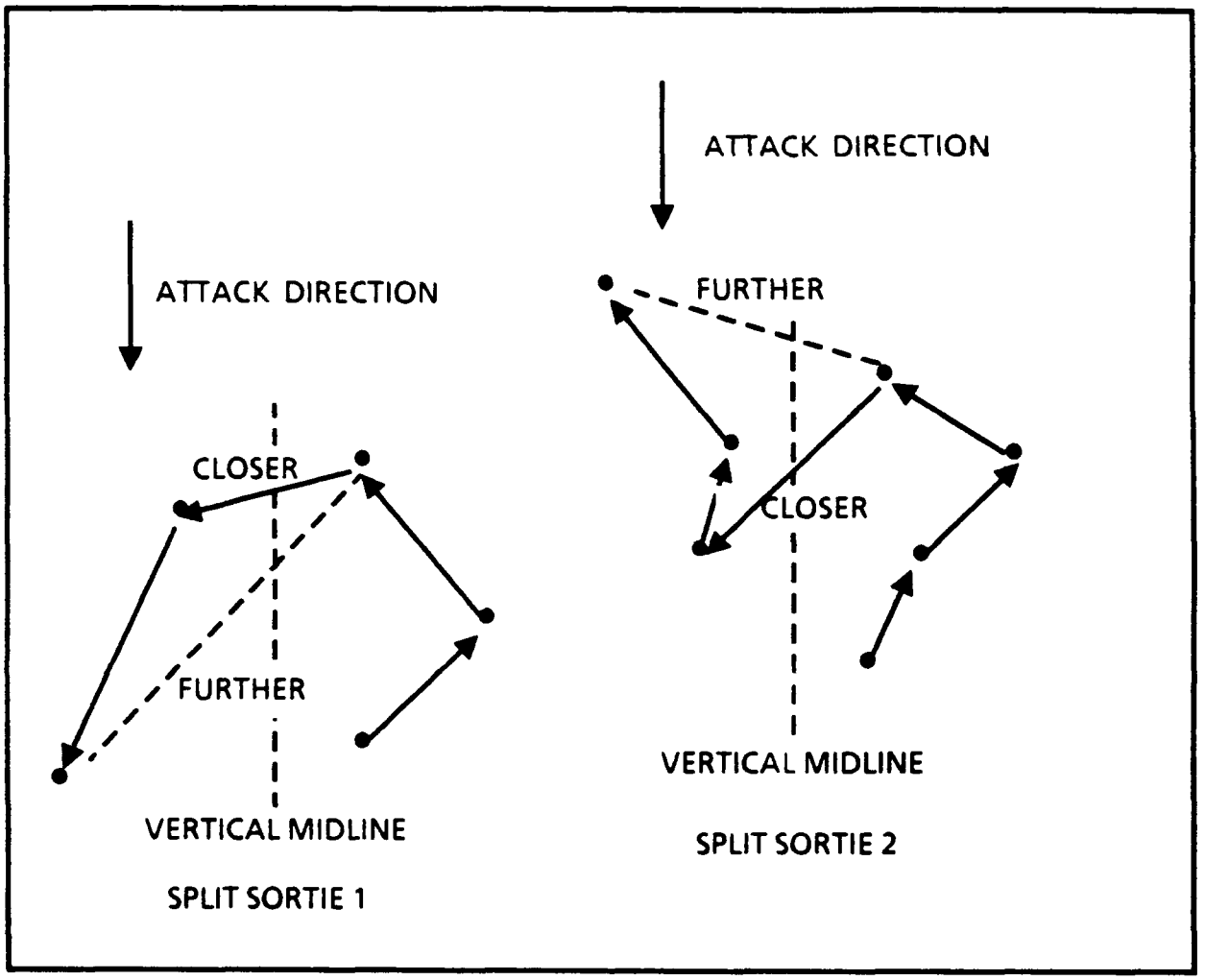
RRVFIL reads a record in the RV Data File and checks for errors. If an error occurs, RRVFIL writes a warning message in the Status File and stops immediately.

After reading the record successfully, RRVFIL calls WRSFMS to write an information message in the Status File. Then RRVFIL gets the yield from the array of real weapon data and returns control to ALM.

Source File:

ALMOVR1B.FOR

Example:


ATTACK DIRECTION
CLOSER
FURTHER
VERTICAL MIDLINE
SPLIT SORTIE 1
ATTACK DIRECTION
FURTHER
CLOSER
VERTICAL MIDLINE
SPLIT SORTIE 2

SUBROUTINE SPLSRT

Purpose: To split an uprange sortie into halves. The missile will move uprange in the half with the greater number of DGZs, then move to the most uprange or most downrange DGZ in the other half (depending on which is closer), and then move either downrange or uprange.

Input Arguments:

None

Output Arguments:

None

Modules Called:

| | |
|---|---|
| CDSTNC | SRTSRT |
| FEXT | SWPSRT |

Calling Modules:

ASR2T2

Common Blocks Used:

| | |
|---|---|
| Block | $INCLUDE |
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SPLSRT calls FEXT to find the extreme Y-coordinates in the current sortie. SPLSRT computes the mean of the extremes to establish a vertical midline that splits the sortie.

SPLSRT executes a loop over the DGZs in the sortie, counting the DGZs in the left half, and setting flags corresponding to each DGZ. If a DGZ is in the left half, the flag is .TRUE.; if it's in the right half, the flag is .FALSE.. SPLSRT rearranges the contents of the sortie data lists. The first part of the lists describes the DGZs in the denser half of the sortie and the second part of the lists describes DGZs in the sparser half.

SPLSRT calls SRTSRT to sort the data in each part of the lists so DGZs are in uprange order. Next, SPLSRT computes the distance from the DGZ at the end of part one to the DGZ at the beginning of part two, and to the DGZ at the end of part two. If the DGZ at the end of part two is closer, SPLSRT reverses the order of the data in part two, so that half of the sortie is a downrange sortie. Then SPLSRT returns control to ASR2T2.

Source File:

ALMOVR2D.FOR

SUBROUTINE SQUEE2 (MSGTVP, SUBROU)

Purpose: To squeeze multiple blanks from the message buffer and then write a Status File message.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| MSGTYP | INTEGER*2 | Message type indicator: 0 = Information, 1 = Warning, 2 = Success, 3 = Error, 4 = Continuation |
| SUBROU | CHARACTER*(*) | Name of the subroutine generating the message |

Output Arguments:

None

Modules Called:

SQBLKS
WRSFMS

Calling Modules:

| | | | |
|---|---|---|---|
| ADDGZ1 | FNPREV | RALLOC | RSSAFE |
| ADDGZ2 | GETSRT | RBOOST | TRYFLY |
| ADJST1 | GTBLYR | RCNTRL | WSORTI |
| ADLAYR | MOVLFT | RLAUNCH | |
| ADXTRA | ORDMSS | RPBVFI | |
| CSTOUT | PREPAR | RRVFIL | |

Common Blocks Used:

| | |
|---|---|
| Block | $INCLUDE |
| OUTPUT | CBOUTPUT.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SQUEE2 calls SQBLKS to squeeze multiple blanks from the message; i.e. each sequence of two or more blanks is reduced to a single blank. If the result is at least one character long, SQUEE2 calls WRSFMS to write the message in the Status File.

Source File:

ALMROOT.FOR

SUBROUTINE SQUEE3

Purpose: To squeeze multiple blanks from the message buffer and then display the message on the screen.

Input Arguments:

None

Output Arguments:

None

Modules Called:

SQBLKS

Calling Modules:

ASR2T2

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| OUTPUT | CBOUTPUT.FOR |

File Input/Output:

| File | Unit No. | Description |
|---|---|---|
| CON | * | The screen (CONsole) can be accessed using an asterisk for the logical unit number. |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SQUEE3 calls SQBLKS to squeeze multiple blanks from the message; i.e., each sequence of two or more blanks is reduced to a single blank. If the result is at least one character long, SQUEE3 displays the message on the screen.

Source File:

ALMROOT.FOR

SUBROUTINE SQUEEZ

Purpose: To form lists of unique DGZs, starting layer numbers, and the number of times each DGZ appears in the current sortie; i.e., to squeeze out duplicate DGZs from a list of DGZs.

Input Arguments:

None

Output Arguments:

None

Modules Called:

SRTSRT

Calling Modules:

FMABAB
FMABBA
MULSRT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SQUEEZ calls SRTSRT to sort the lists of sortie data, so DGZs are in uprange order. Then SQUEEZ sets indices for storing data in the squeezed lists, and for examining data in the original lists, to one.

(10) SQUEEZ stores the current DGZ number, in the original lists, in the list of unique DGZs. SQUEEZ stores the DGZ's layer number as the DGZ's starting layer. SQUEEZ prepares to examine subsequent DGZs in the original list, and defines the number of times the current DGZ appears to be one.

(15) SQUEEZ checks to see if all subsequent DGZs in the original list have been checked. If so, SQUEEZ goes to statement 20 to prepare to find another unique DGZ. If not, SQUEEZ compares the two DGZs to see if they're identical. If so, the number of times the DGZ appears in the sortie is incremented, the pointer to the next DGZ is incremented, and SQUEEZ goes back to statement 15 to check the new DGZ.

(20) SQUEEZ prepares to store data for the next unique DGZ in the lists, and increments the first index used to examine DGZs in the original sortie. If there's another DGZ to examine, SQUEEZ goes back to statement 10.

Otherwise, SQUEEZ defines the number of unique DGZs and returns control to the calling module.

Source File:

ALMOVR2D.FOR

SUBROUTINE SRTSR2 (ISTART, ISTOP, LIST)

Purpose: To sort a section of the sortie lists, so integers in one of the lists are in ascending order.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| ISTART | INTEGER*2 | Starting index for lists. |
| ISTOP | INTEGER*2 | Stopping index for lists. |
| LIST | INTEGER*2 | Array: list of integer sortie data. |

Output Arguments:

None

Modules Called:

IFMIN
SWPSRT

Calling Modules:

SRTSRT

Common Blocks Used:

None

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SRTSR2 executes a loop over the sets of data specified by the starting and stopping indices. During each pass, SRTSR2 calls IFMIN to find the smallest integer in the subset of the specified integer list. To sort the data, SRTSR2 calls SWPSRT whenever two sets of sortie data should be swapped.

Source File:

ALMOVR2D.FOR

SUBROUTINE SRTSR3 (LAST)

Purpose: To sort the sortie lists so DGZ values are in ascending order.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| LAST | INTEGER*2 | Index for last DGZ in current sortie. |

Output Arguments:

None

Modules Called:

SWPSRT

Calling Modules:

ADXTRA
MOPUPA

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |
| TI | CBTI.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SRTSR3 executes a loop over the number of DGZs specified by LAST. During each pass, SRTSR3 finds the DGZ with the highest value in the current subset of the sortie data. To sort the data, SRTSR3 calls SWPSRT whenever two sets of sortie data should be swapped.

Source File:

ALMOVR2D.FOR

SUBROUTINE SRTSRT (ISTART, ISTOP)

Purpose: To sort a section of the sortie lists, so X-coordinates are in descending order; i.e., so DGZs are in uprange order. If the same DGZ appears more than once, sort layer numbers so they're in ascending order.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| ISTART | INTEGER*2 | Starting index for lists. |
| ISTOP | INTEGER*2 | Stopping index for lists. |

Output Arguments:

None

Modules Called:

SRTSR2
UPRSRT

Calling Modules:

CHKMLT
GETSRT
MOPUPA

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SRTSRT calls UPRSRT to sort the specified subset of the sortie data, so X-coordinates are in descending order. Then SRTSRT ensures that identical DGZs are contiguous in the lists. (Two different DGZs could both appear in the lists more than once, and could have the same X-coordinate. Data for the two DGZs might alternate in the lists.)

SRTSRT then calls SRTSR2 for each subset of data corresponding to a single DGZ, to ensure layer numbers for the DGZ are in ascending order.

DGZs are in uprange order.

Source File:

ALMOVR2D.FOR

SUBROUTINE STRSRT (JM, BTNEW, PNEW, TOFNEW)

Purpose: To store successful sortie data in arrays corresponding to a specified missile; to indicate missile has been used; and to update burntime, pitch, and time-of-flight.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| JM | INTEGER*2 | Missile index. |
| BTNEW | REAL*4 | New burntime (seconds). |
| PNEW | REAL*4 | New booster pitch (degrees). |
| TOFNEW | REAL*4 | New time-of-flight (seconds). |

Output Arguments:

None

Modules Called:

None

Calling Modules:

| | |
|---|---|
| ASR2T2 | MOPUPB |
| MOPUPA | MOPUPC |

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine STRSRT executes a loop over all DGZs in the sortie, storing the DGZ index,the downtime, and the layer number in array elements corresponding to the RVs on the specified missile. STRSRT sets a logical flag associated with the missile to .TRUE., indicating the missile has been used. STRSRT copies the new flight parameters to variables specifying the previous flight parameters. Then control returns to the calling module.

Source File:

ALMOVR2D.FOR

SUBROUTINE SWPSRT (IA, IB)

Purpose: To swap two sets of data in the sortie lists.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| IA | INTEGER*2 | Index for first set to swap. |
| IB | INTEGER*2 | Index for second set to swap. |

Output Arguments:

None

Modules Called:

ISWAP
SWAP

Calling Modules:

| | |
|---|---|
| CROSRT | SRTSR2 |
| DWNSRT | SRTSR3 |
| SPLSRT | UPRSRT |

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SWPSRT calls ISWAP to swap integer values and SWAP to swap real values.

Source File:

ALMOVR2D.FOR

SUBROUTINE SYNEZY (NRVS, LATEST)

Purpose: To synchronize downtimes for RVs in the first layer that are the first to arrive from their respective missiles. Because the time between launches is known to be zero, the downtimes can be truly synchronized.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| NRVS | INTEGER*2 | The number of RVs on each missile. |
| LATEST | REAL*4 | The latest downtime among all the RVs in the first layer that are the first to arrive from their respective missile. |

Output Arguments:

None

Modules Called:

ADJST4

Calling Modules:

SYNTIM

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

Subroutine SYNEZY executes a loop over all missiles used, looking for missiles whose first RV to arrive is in the first layer. For each such missile, SYNEZY subtracts the downtime of the first RV to arrive from the latest downtime supplied as an input argument. If the difference is greater than zero, SYNEZY calls ADJST4 to adjust the launch time and the downtime for every RV on the missile (times are adjusted by adding the difference just computed to the launch time and to every downtime).

Source File:

ALMOVR2D.FOR

SUBROUTINE SYNTIM

Purpose: To "synchronize" downtimes for RVs in the first layer that are the first to arrive from their respective missiles. If the time between launches is greater than zero, true synchronization is impossible. The most one can hope for is to minimize the span of downtimes.

Input Arguments:

None

Output Arguments:

None

Modules Called:

ADJST5
SYNEZY

Calling Modules:

ASR2TI

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| LC | CBLC.FOR |
| TI | CBTI.FOR |
| WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

If the time between launches is zero, subroutine SYNTIM executes a loop over all missiles used, looking for missiles whose first RV to arrive is in the first layer. SYNTIM finds the latest downtime among those RVs. Then SYNTIM calls SYNEZY to synchronize the downtimes for those RVs.

If the time between launches is greater than zero, SYNTIM calls ADJST5 to minimize the span of downtimes for first-layer RVs.

Source File:

ALMOVR2D.FOR

SUBROUTINE TRYFLY (POSSBL)

Purpose: To fly a missile from the silo closest to the target island, and to determine if it can drop all its RVs on the DGZ furthest from the launch complex.

Input Arguments:

None

Output Arguments:

| Variable | Type | Description |
|---|---|---|
| POSSBL | LOGICAL*1 | Flag: If .TRUE., it's possible to fly the missile and deploy RVs successfully. |

Modules Called:

| | |
|---|---|
| CHKSRT | PRPNTS |
| FMAX | SQUEE2 |

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | SORTEE | CBSORTIE.FOR |
| OPSHNS | CBOPSHNS.FOR | TI | CBTI.FOR |
| OUTPUT | CBOUTPUT.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

TRYFLY calls PRPNTS to project all DGZs onto a plane tangent to the earth at the target island's centroid. The negative X-axis points up-range to the launch complex and the positive X-axis, downrange. TRYFLY calls FMAX to find the DGZ furthest from the launch complex; i.e., the DGZ with the maximum X-coordinate.

TRYFLY calls PRPNTS again to project all missile silos onto the same tangent plane. TRYFLY calls FMAX again to find the silo with the maximum X-coordinate; i.e., the silo closest to the target island. TRYFLY forms a sortie consisting of the DGZ furthest from the launch complex, repeated until there are as many DGZs in the sortie as there are RVs on the missile.

TRYFLY calls CHKSRT to see if the missile can do the sortie. If so, TRYFLY returns control to ALM with POSSBL set to .TRUE.; if not, TRYFLY returns with POSSBL set to .FALSE..

Source File:

ALMOVR1B.FOR

SUBROUTINE UNDO

Purpose: To undo a failed sortie; i.e., to restore values to what they were before the sortie was formed.

Input Arguments:

None

Output Arguments:

None

Modules Called:

None

Calling Modules:

| | |
|---|---|
| ASR2T2 | MOPUPB |
| MOPUPA | MOPUPC |

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| LC | CBLC.FOR | TI | CBTI.FOR |
| SORTEE | CBSORTIE.FOR | WEAPON | CBWEAPON.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

UNDO restores the former value of "previous launch time" by subtracting time between launches from its current value. UNDO executes a loop over the DGZs in the current sortie, decrementing the number of RVs assigned so far to each DGZ. Then UNDO returns control to the calling module.

Source File:

ALMOVR2D.FOR

SUBROUTINE UPRSRT (FIRST, LAST)

Purpose: To sort data in a subset of the sortie lists, so DGZs are in up-range order.

Input Arguments:

| Variable | Type | Description |
|---|---|---|
| FIRST | INTEGER*2 | Index for first entry in lists. |
| LAST | INTEGER*2 | Index for last entry in lists. |

Output Arguments:

None

Modules Called:

FMAX
SWPSRT

Calling Modules:

ASR2T2
SRTSRT

Common Blocks Used:

| Block | $INCLUDE |
|---|---|
| SORTEE | CBSORTIE.FOR |

File Input/Output:

None

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

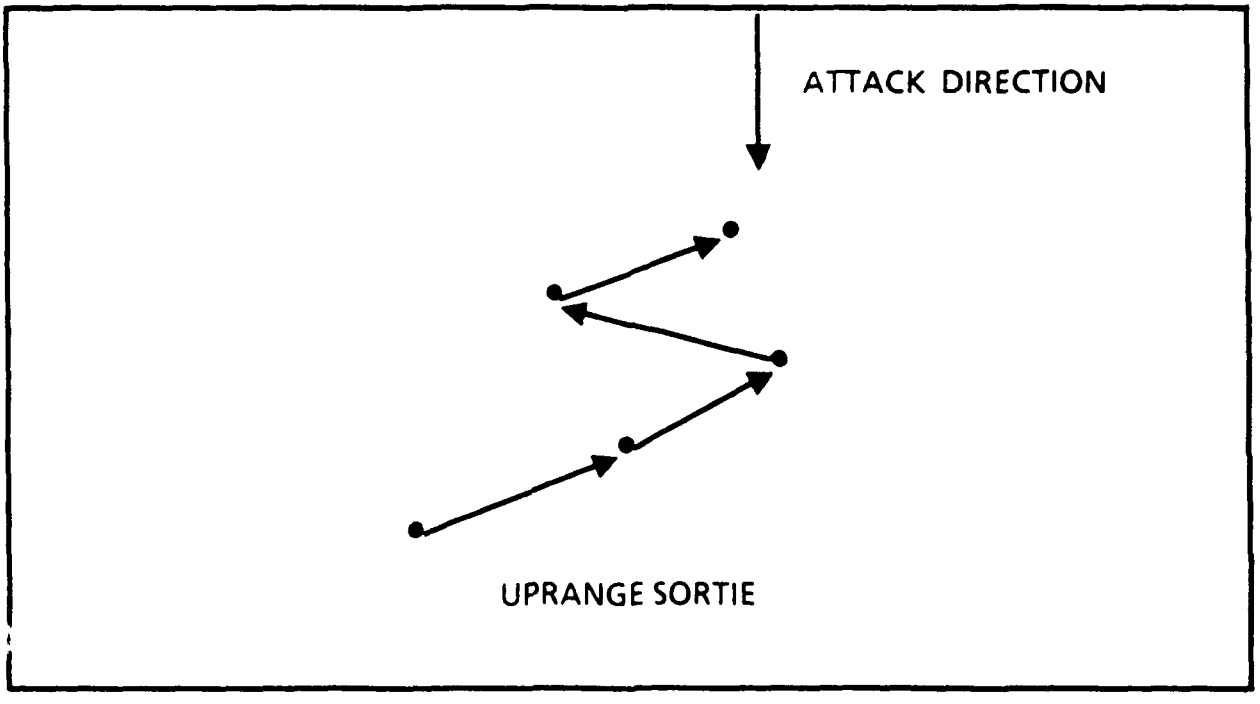
Description:

UPRSRT executes a loop over the range of entries specified by FIRST and LAST, performing a bubble sort on X-coordinates. UPRSRT calls FMAX to find the maximum X-coordinate, and calls SWPSRT to swap sets of sortie data. The result is shown in the example below.

Source File:

ALMOVR2D.FOR

Example:



SUBROUTINE WSORTI

Purpose: To write the Sortie File.

Input Arguments:

None

Output Arguments:

None

Modules Called:

SQUEE2

Calling Modules:

ALM

Common Blocks Used:

| Block | $INCLUDE | Block | $INCLUDE |
|---|---|---|---|
| CNTROL | CBCNTROL.FOR | OUTPUT | CBOUTPUT.FOR |
| CSTRNG | CBCSTRNG.FOR | TI | CBTI.FOR |
| LC | CBLC.FOR | TSTRNG | CBTSTRNG.FOR |
| LSTRNG | CBLSTRNG.FOR | WEAPON | CBWEAPON.FOR |
| OPSHNS | CBOPSHNS.FOR | | |

File Input/Output:

| File | Unit No. | Description |
|---|---|---|
| "case".SF | 4 | Sortie File |

FORTRAN 77 Exceptions:

1. Uses "*" typing for variables.
2. Uses $INCLUDE, $PAGE, and $SUBTITLE metacommands.

Description:

WSORTI opens the Sortie File, writes records according to the IOPT in Section 4, and then closes the file. If errors occur, WSORTI writes a warning message in the Status File and stops.

Source File:

ALMOVER3.FOR

# APPENDIX D
# COMMON BLOCK LIST

This appendix lists all common blocks that are unique to the ALM CSC and provides a brief description of the type of data contained in each. The user is referred to Reference 2 for a description of the common blocks used exclusively in the TARGET CSC modules that are employed by ALM to assess sortie feasibility.

Note that Microsoft FORTRAN forbids mixing character variables with other kinds of variables in the same common block. Consequently, several pairs of common blocks are used. Both members of a pair might contain data read from the same input file, for example, but one contains character data only, while the other contains integer, real, and logical data.

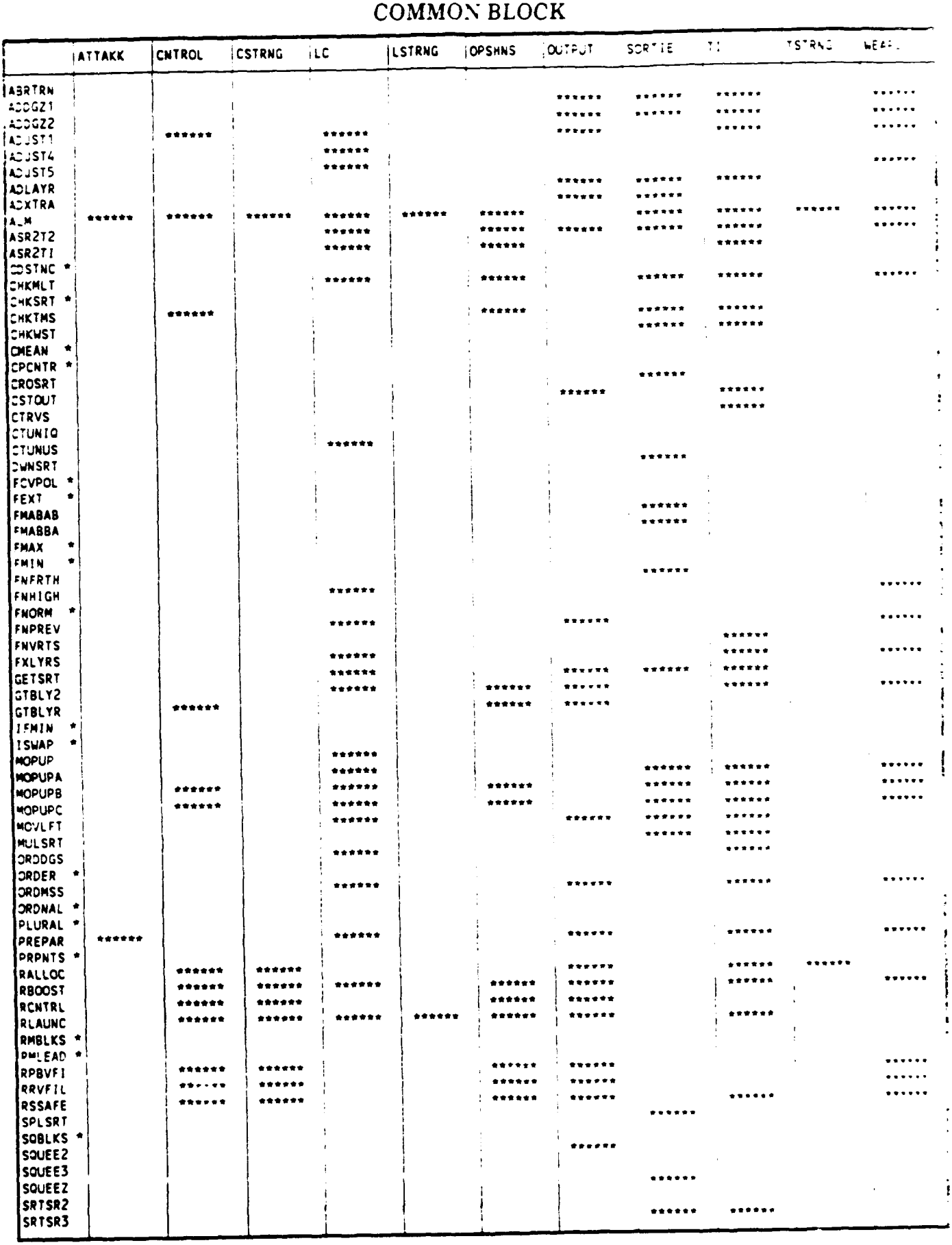
ALM COMMON BLOCK LIST

| | |
|---|---|
| ATTAKK | Variables that describe the attack; e.g. the DGZ attacked by each RV. NOTE: The block's name is deliberately spelled ATTAKK because subroutines shared by the ALM and TARGET CSCs use another block spelled ATTACK. |
| CNTROL | Integers read from the Control File and auxiliary values associated with them; e.g. number of Fratricide Constraints Files to use, and the length of each input file's name. |
| CSTRNG | Character strings read from the Control File; e.g. the Attack File's name. |
| LC | Launch complex data; e.g. silo latitudes and longitudes. |
| LSTRNG | A character string containing the launch complex's name. |
| OPSHNS | Options read from the Control File; the flag indicating whether or not to minimize the span of RV downtimes. |
| OUTPUT | A buffer used to write Status File messages, and an integer specifying the message length. Note that a character variable is combined with an integer variable in this common block. Since the character variable comes last, the combination is acceptable. |
| SORTEE | Variables describing the current sortie; e.g. the downtimes for each RV in the sortie. The block's name is deliberately spelled SORTEE because the TARGET library contains a subroutine named SORTIE. |
| TI | Variables describing the target island; e.g. latitudes and longitudes for each DGZ. |
| TSTRNG | A character string containing the target island's label. |

| | |
|---|---|
| WEAPON | Variables describing the weapon (missile) used to attack the target island; e.g. the yield in kilotons. |

# APPENDIX E
# COMMON BLOCK CROSS REFERENCE

This Appendix provides a common block cross reference that shows the common blocks used by each module, and the modules that use each common block. COMMON block names are used as column headings across the top of the figure: module names run down the left side. Module names marked with an asterisk are in the MPS Utilities Library; see Reference 3.




COMMON BLOCK CROSS REFERENCE

COMMON BLOCK

| MODULE | ATTAKK | CNTROL | CSTRNG | LC | LSTRNG | OPSHNS | OUTPUT | SORTIE | TI | TSTRNG | WEAPON |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SRTSRT | | | | | | | | ****** | | | |
| STRSRT | | | | ****** | | | | ****** | | | |
| SUM * | | | | | | | | | | | |
| SWAP * | | | | | | | | | | | |
| SWPSRT | | | | | | | | ****** | | | |
| SYNEZY | | | | ****** | | | | | | | ****** |
| SYNTIM | | | | ****** | | | | | ****** | | ****** |
| TRULEN * | | | | | | | | | | | |
| TRYFLY | | | | ****** | | ****** | ****** | ****** | ****** | | ****** |
| UNDO | | | | ****** | | | | ****** | ****** | | ****** |
| UPRSRT | | | | | | | | ****** | | | |
| WATTCK * | | | | | | | | | | | |
| WRSFFR * | | | | | | | | | | | |
| WRSFMS * | | | | | | | | | | | |
| WSORTI | | ****** | ****** | ****** | ****** | ****** | ****** | | ****** | ****** | ****** |

## COMMON BLOCK CROSS REFERENCE

# APPENDIX F
# COMMON BLOCK DEFINITIONS

Appendix F identifies and describes all common blocks used in the ALM CSC. Every common variable is identified and defined. Additionally, parameters used in dimensioning common arrays are cross referenced to the document where they are defined.

## ATTAKK COMMON

COMMON block /ATTAKK/ is defined in the FORTRAN source fi.. CBATTACK.FOR. The block's name is deliberately misspelled because the TARGET library contains a block named ATTACK. The file is inserted in any module requiring attack data using the $INCLUDE metacommand. The file's contents are as follows:

```
INTEGER*2 DGZ4R2, DGZTY2, HOBIN2, MSL4RV, MSLIND, ORDINM

REAL*4   DGZLA2, DGZLO2, DWNTI2, EARDT, EARLT, LATDT,
1        LATLT, LDURAT, LYDRAT, MISLA2, MISLO2, PATTRI,
2        PFRATR, TLNCH2

COMMON /ATTAKK/DGZ4R2(MRVST), DGZLA2(MRVST), DGZLO2(MRVST),
1                DGZTY2(MRVST), DWNTI2(MRVST), EARDT, EARLT,
2                HOBIN2(MRVST), LATDT, LATLT, LDURAT, LYDRAT,
3                MISLA2(MMSLS), MISLO2(MMSLS), MSL4RV(MRVST),
4                MSLIND(MMSLS), ORDINM(MRVST), PATTRI(MRVST),
5                PFRATR(MPROBS), TLNCH2(MMSLS)
```

| Variable | Type | Definition |
|---|---|---|
| DGZ4R2(I) | INTEGER*2 | Array: DGZ index for I-th RV to arrive at target island |
| DGZLA2(I) | REAL*4 | Array: latitude for I-th RV's DGZ (degrees North) |
| DGZLO2(I) | REAL*4 | Array: longitude for I-th RV's DGZ (degrees East) |
| DGZTY2(I) | INTEGER*2 | Array: type of DGZ attacked by I-th RV |
| DWNTI2(I) | REAL*4 | Array: downtime for I-th RV (seconds) |
| EARDT | REAL*4 | Earliest downtime (seconds) |
| EARLT | REAL*4 | Earliest launch time (seconds) |

| | | |
|---|---|---|
| HOBIN2(I) | INTEGER*2 | Array: height of burst index for I-th RV |
| LATDT | REAL*4 | Latest downtime (seconds) |
| LATLT | REAL*4 | Latest launch time (seconds) |
| LDURAT | REAL*4 | Launch time duration (seconds) |
| LYDRAT | REAL*4 | Laydown duration (seconds) |
| MISLA2(I) | REAL*4 | Array: I-th missile silo's latitude (degrees North) |
| MISLO2(I) | REAL*4 | Array: I-th missile silo's longitude (degrees East) |
| MSL4RV(I) | INTEGER*2 | Array: I-th RV's missile |
| MSLIND(I) | INTEGER*2 | Array: I-th missile silo's index |
| ORDINM(I) | INTEGER*2 | Array: I-th RV's order in missile |
| PATTRI(I) | REAL*4 | Array: probability of attrition for I-th RV |
| PFRATR(K) | REAL*4 | Array: probability of fratricide for J-th RV (J = 2, 3, 4,...); i.e. probability that I-th RV (I = 1, 2, 3,...,J-1) destroys J-th RV. $K = J*(J-3)/2 + I + 1$ |
| TLNCH2(I) | REAL*4 | Array: I-th missile's launch time (seconds) |

| Parameter | Where Defined | Where Documented |
|---|---|---|
| MMSLS | MAXIMA.FIF | Reference 1 |
| MPROBS | MAXIMA.FIF | Reference 1 |
| MRVST | MAXIMA.FIF | Reference 1 |

# CNTROL COMMON

COMMON block /CNTROL/ is defined in the FORTRAN source file CBCNTROL.FOR. The file is inserted in any module requiring control data using the $INCLUDE metacommand. The file's contents are as follows:

```
INTEGER*2 LATKFI, LBOOFI, LCSDES, LCSNAM, LFCPAT
INTEGER*2 LLNLFI, LMSLNA, LPBVFI, LRVAFI, LRVFIL, LSORFI
INTEGER*2 NFCFLS, NMODLS, RNBOOF, RNPBVF, RNRVFI

COMMON /CNTROL/ LATKFI, LBOOFI, LCSDES, LCSNAM, LFCPAT,
1                LLNLFI, LMSLNA, LPBVFI, LRVAFI, LRVFIL, LSORFI,
2                NFCFLS, RNBOOF, RNPBVF, RNRVFI

EQUIVALENCE (NFCFLS, NMODLS)
```

| Variable | Type | Definition |
|---|---|---|
| LATKFI | INTEGER*2 | Length of attack file's name (includes path) |
| LBOOFI | INTEGER*2 | Length of booster file's name (includes path) |
| LCSDES | INTEGER*2 | Length of case descriptor |
| LCSNAM | INTEGER*2 | Length of case name |
| LFCPAT | INTEGER*2 | Length of path for fratricide constraint files |
| LLNLFI | INTEGER*2 | Length of launch location file's name |
| LMSLNA | INTEGER*2 | Length of missile name |
| LPBVFI | INTEGER*2 | Length of PBV file's name |
| LRVFIL | INTEGER*2 | Length of RV file's name |
| LRVAFI | INTEGER*2 | Length of RV allocation file's name |
| LSORFI | INTEGER*2 | Length of sortie file's name |
| NFCFLS | INTEGER*2 | Number of fratricide constraints files to use |

| | | |
|---|---|---|
| RNBOOF | INTEGER*2 | Record number for booster file |
| RNPBVF | INTEGER*2 | Record number for PBV file |
| RNRVFI | INTEGER*2 | Record number for RV file |

## CSTRNG COMMON

COMMON block /CSTRNG/ is defined in the FORTRAN source file CBCSTRNG.FOR. The file is inserted in any module requiring control data (in the form of character strings) using the $INCLUDE metacommand. The file's contents are as follows:

```
CHARACTER*8     CSNAME
CHARACTER*12    FCNAME
CHARACTER*20    MSLNAM
CHARACTER*40    CSDESC
CHARACTER*60    ATKFIL, BOOFIL, FCPATH, LNLFIL, PBVFIL, RVFILE
CHARACTER*60    RVAFIL, SORFIL

COMMON/CSTRNG/ ATKFIL, BOOFIL, CSDESC, CSNAME, FCNAME(MHOBS)
1               FCPATH, LNLFIL, MSLNAM, PBVFIL, RVFILE,
2               RVAFIL, SORFIL
```

| Variable | Type | Definition |
|---|---|---|
| ATKFIL | CHARACTER*60 | Attack file's name (includes path) |
| BOOFIL | CHARACTER*60 | Booster file's name (includes path) |
| CSDESC | CHARACTER*40 | Case description |
| CSNAME | CHARACTER*8 | Case name |
| FCNAME | CHARACTER*12 | Name of I-th fratricide constraints file |
| FCPATH | CHARACTER*60 | Path for fratricide constraints files |
| LNLFIL | CHARACTER*60 | Launch location file's name (includes path) |
| MSLNAM | CHARACTER*20 | Missile name |
| PBVFIL | CHARACTER*60 | PBV file's name (includes path) |
| RVAFIL | CHARACTER*60 | RV allocation file's name (includes path) |

| | | |
|---|---|---|
| RVFILE | CHARACTER*60 | RV file's name (includes path) |
| SORFIL | CHARACTER*60 | Sortie file's name (includes path) |

| Parameter | Where Defined | Where Documented |
|---|---|---|
| MHOBS | MAXIMA.FIF | Reference 3 |

## LC COMMON

COMMON block /LC/ is defined in the FORTRAN source file CBLC.FOR. The file is inserted in any module requiring launch complex data using the $INCLUDE metacommand. The file's contents are as follows:

```
INTEGER*2    DGZ4RV, LAYER, LLCLAB, NMSLLC, NMUSED, NRUSED
INTEGER*2    NRVSLC, NSILOS, ORMSLS
LOGICAL*1    COLOCA, USDMSL
REAL*4       BRNTIM, DWNTIM, LCELEV, LCLAT, LCLON
REAL*4       MISALT, MISLAT, MISLON, PITCH, PLTIME, TLNCH
REAL*4       TOFLC, XSILO, YSILO

COMMON /LC/ BRNTIM, COLOCA, DGZ4RV(MRVSM,MMSLS)
1           DWNTIM(MRVSM,MMSLS), LAYER(MRVSM,MMSLS),
2           LCELEV, LCLAT, LCLON, LLCLAB,
3           MISALT(MMSLS), MISLAT(MMSLS),
4           MISLON(MMSLS), NMSLLC, NMUSED, NRUSED, NRVSLC,
5           ORMSLS(MMSLS), PITCH, PLTIME, TLNCH(MMSLS),
6           TOFLC, USDMSL(MMSLS), XSILO(MMSLS), YSILO(MMSLS)

EQUIVALENCE (NMSLLC, NSILOS)
```

| Variable | Type | Definition |
|---|---|---|
| BRNTIM | REAL*4 | Initial (default) burntime in seconds |
| COLOCA | LOGICAL*1 | Flag: if .TRUE., missiles are co-located; i.e. have identical coordinates |
| DGZ4RV(I,J) | INTEGER*2 | Array: DGZ for I-th RV on J-th missile |
| DWNTIM(I,J) | REAL*4 | Array: downtime (seconds) for I-th RV on J-th missile |
| LAYER(I,J) | INTEGER*2 | Array: layer number for I-th RV on J-th missile |

| | | |
|---|---|---|
| LCELEV | REAL*4 | Launch complex's elevation (altitude) in feet |
| LCLAT | REAL*4 | Launch complex's latitude (in degrees North) |
| LCLON | REAL*4 | Launch complex's longitude (in degrees East) |
| LLCLAB | INTEGER*2 | Length of launch complex label |
| MISALT(I) | REAL*4 | Array: I-th missile's altitude (in feet) |
| MISLAT(I) | REAL*4 | Array: I-th missile's latitude (in degrees North) |
| MISLON(I) | REAL*4 | Array: I-th missile's longitude (in degrees East) |
| NMSLLC | INTEGER*2 | Number of missiles (silos) at launch complex. Read from Launch Location File. |
| NMUSED | INTEGER*2 | Number of missiles used. Computed during run: NMUSED < = NMSLLC. |
| NRUSED | INTEGER*2 | Number of RVs used: product of number of missiles used (NMUSED) and number of RVs per weapon (NRVSWP, in COMMON block /WEAPON/). |
| NRVSLC | INTEGER*2 | Number of RVs at launch complex: product of number of missiles at launch complex (NMSLLC) and number of RVs per weapon (NRVSWP, in COMMON block /WEAPON/). |
| NSILOS | INTEGER*2 | Number of silos (missiles) at launch complex; synonym for NMSLLC |
| ORMSLS(I) | INTEGER*2 | Array: Index (subscript, pointer) of I-th missile to use |
| PITCH | REAL*4 | Initial (default) booster pitch angle (degrees) |
| PLTIME | REAL*4 | Previous launch time (seconds) |
| TLNCH(I) | REAL*4 | Array: launch time for I-th missile (seconds) |
| TOFLC | REAL*4 | Initial (default) time-of-flight (seconds) |

| | | |
|---|---|---|
| USDMSL(I) | LOGICAL*1 | Array: flag: if .TRUE., I-th missile was used |
| XSILO(I) | REAL*4 | Array: tangent-plane X-coordinate for I-th silo |
| YSILO(I) | REAL*4 | Array: tangent-plane Y-coordinate for I-th silo |

| Parameter | Where Defined | Where Documented |
|---|---|---|
| MMSLS | MAXIMA.FIF | Reference 3 |
| MRVSM | MAXIMA.FIF | Reference 3 |

## LSTRNG COMMON

COMMON block /LSTRNG/ is defined in the FORTRAN source file CBLSTRNG.FOR. The file is inserted in any module requiring the launch complex's label using the $iNCLUDE metacommand. The file's contents are as follows:

```
CHARACTER*24  LCLABL
COMMON/LSTRNG/LCLABL
```

| Variable | Type | Definition |
|---|---|---|
| LCLABL | CHARACTER*24 | Launch complex's label |

## OPSHNS COMMON

COMMON block /OPSHNS/ is defined in the FORTRAN source file CBOPSHNS.FOR. The file is inserted in any module requiring Control File options using the $INCLUDE metacommand. The file's contents are as follows:

```
LOGICAL*1 SYNCRO
LOGICAL*4 ROTRTH, SPACED
REAL*4   FCFHOB, FRF4RX, GAMMA, SHRSAF, TBLAYR

COMMON /OPSHNS/ FCFHOB(MHOBS), FRF4RX, GAMMA, ROTRTH,
1                SHRSAF(MHOBS), SPACED, SYNCRO,
2                TBLAYR(MLAYRS-1)
```

| Variable | Type | Definition |
|---|---|---|
| FCFHOB(I) | REAL*4 | Array: Height of burst read from I-th fratricide constraints file; in kilofeet |
| FRF4RX | REAL*4 | Maximum fraction of fuel to use for range extension |
| GAMMA | REAL*4 | Reentry angle (degrees) for all RVs; if zero, use minimum energy reentry angle |
| ROTRTH | LOGICAL*4 | Flag: if .TRUE., use rotating earth |
| SHRSAF(I) | REAL*4 | Array: sure-safe time read from I-th fratricide constraints file; in seconds |
| SPACED | LOGICAL*4 | Flag: if .TRUE., RVs should be spaced at least a minimum distance apart at the spacing reference altitude |
| SYNCRO | LOGICAL*1 | Flag: if .TRUE., downtimes should be synchronized (or the span of downtimes should be minimized) |
| TBLAYR(I) | REAL*4 | Array: time between I-th and I + 1-th layers (seconds) |

| Parameter | Where Defined | Where Documented |
|---|---|---|
| NHOBS | MAXIMA.FIF | Reference 3 |
| MLAYRS | MAXIMA.FIF | Reference 3 |

## OUTPUT COMMON

COMMON block /OUTPUT/ is defined in the FORTRAN source file CBOUTPUT.FOR. The file is inserted in any module requiring an output message buffer using the $INCLUDE metacommand. The file's contents are as follows:

```
CHARACTER*48 MESSAG
INTEGER*2    LENGTH

COMMON/OUTPUT/ LENGTH, MESSAG
```

| Variable | Type | Definition |
|---|---|---|
| LENGTH | INTEGER*2 | Length of message (or other string) |
| MESSAG | CHARACTER*48 | Buffer for status file messages |

## SORTEE COMMON

COMMON block /SORTEE/ is defined in the FORTRAN source file CBSORTIE.FOR. The block's name is deliberately misspelled because the TARGET library contains a subroutine named SORTIE. The file is inserted in any module requiring sortie data using the $INCLUDE metacommand. The file's contents are as follows:

```
INTEGER*2 DGZSRT, LYRSRT, NGZSRT, NTIMES, NUNEEK, STRLYR, UNEEK
REAL*4    DT4SRT, LATSRT, LONSRT, XSRT, YSRT

COMMON /SORTEE/ DGZSRT(MRVSM), DT4SRT(MRVSM),
1                LATSRT(MRVSM), LONSRT(MRVSM), LYRSRT(MRVSM),
2                NGZSRT, NTIMES(MRVSM) ;UNEEK, STRLYR(MRVSM),
3                UNEEK(MRVSM), XSRT(MRVSM), YSRT(MRVSM)
```

| Variable | Type | Definition |
|---|---|---|
| DGZSRT(I) | INTEGER*2 | Array: DGZ for I-th RV in the sortie |
| DT4SRT(I) | REAL*4 | Array: downtime for I-th RV in the sortie |
| LATSRT(I) | REAL*4 | Array: latitude for I-th DGZ in the sortie |
| LONSRT(I) | REAL*4 | Array: longitude for I-th DGZ in the sortie |
| LYRSRT(I) | INTEGER*2 | Array: layer number for I-th RV in the sortie |
| NGZSRT | INTEGER*2 | Number of DGZs (RVs) in the sortie |
| NTIMES(I) | INTEGER*2 | Array: number of times the I-th unique DGZ appears in the sortie |
| NUNEEK | INTEGER*2 | Number of unique DGZs in the sortie |
| STRLYR(I) | INTEGER*2 | Array: starting layer number for I-th unique DGZ in the sortie |
| UNEEK(I) | INTEGER*2 | Array: I-th unique DGZ number in the sortie |

| | | |
|---|---|---|
| XSRT(I) | REAL*4 | Array: X-coordinate for I-th DGZ in the sortie |
| YSRT(I) | REAL*4 | Array: Y-coordinate for I-th DGZ in the sortie |

| Parameter | Where Defined | Where Documented |
|---|---|---|
| MRVSM | MAXIMA.FIF | Reference 3 |

## TI COMMON

COMMON block /TI/ is defined in the FORTRAN source file CBTI.FOR. The file is inserted in any module requiring target island data using the $INCLUDE metacommand. The file's contents are as follows:

```
INTEGER*2 DGZDES, DGZNUM, DGZTYP, HIMAX, HIMIN, HOBIND
INTEGER*2 LTILAB, NGZSTI, NHOBS, NMATTI, NRATTI, NRSOFR
INTEGER*2 NRVDGZ, ORDGZS, ORDRVS

LOGICAL*1  OUTCST, VERTEX

REAL*4     DGZLAT, DGZLON, HOB
REAL*4     TIELEV, TILAT, TILON
REAL*4     VALUE, XDGZ, YDGZ

COMMON /TI/ DGZLAT(MDGZS), DGZLON(MDGZS), DGZNUM(MDGZS),
1            DGZTYP(MDGZS), HIMAX, HIMIN, HOB(1:MHOBS),
2            HOBIND(MDGZS), LTILAB, NGZSTI, NHOBS, NMATTI,
3            NRATTI, NRSOFR(MDGZS), NRVDGZ(MDGZS),
4            ORDGZS(MDGZS), OUTCST(MDGZS), TIELEV, TILAT, TILON,
5            VALUE(MDGZS), VERTEX(MDGZS),
6            XDGZ(MDGZS), YDGZ(MDGZS)

DIMENSION DGZDES(MDGZS)
EQUIVALENCE (DGZDES, DGZNUM)
```

| Variable | Type | Definition |
|---|---|---|
| DGZLAT(I) | REAL*4 | Array: latitude of I-th DGZ (degrees North) |
| DGZLON(I) | REAL*4 | Array: longitude of I-th DGZ (degrees East) |

| | | |
|---|---|---|
| DGZNUM(I) | INTEGER*2 | Array: four-digit integer identifying I-th DGZ |
| HIMAX | INTEGER*2 | Array: maximum height of burst index found in RV Allocation file |
| HIMIN | INTEGER*2 | Minimum height of burst index found in RV Allocation file |
| NGZSTI | INTEGER*2 | Number of DGZs in target island |
| NHOBS | INTEGER*2 | Number of heights of burst in the attack. Equals number of unique height of burst indices found in RV Allocation File. |
| NMATTI | INTEGER*2 | Number of missiles attacking target island. Read from RV Allocation File. Should be less than or equal to number of missiles in launch complex (NMSLLC, in COMMON block /LC/. Number of missiles actually used (NMUSED, in COMMON block /LC/) may be less than NMATTI. |
| NRATTI | INTEGER*2 | Number of RVs attacking target island. Read from RV Allocation File. Should be product of NMATTI and number of RVs per weapon (NRVSWP, in COMMON block /WEAPON/). |
| NRSOFR(I) | INTEGER*2 | Array: number of RVs assigned, so far, to I-th DGZ |
| NRVDGZ(I) | INTEGER*2 | Array: number of RVs allocated to I-th DGZ (read from RV Allocation File) |
| ORDGZS(I) | INTEGER*2 | Array: index (subscript, pointer) to I-th DGZ, when sweeping from right to left |
| OUTCST(I) | LOGICAL*1 | Array: flag: if .TRUE., I-th DGZ is an "outcast"; i.e. unreachable |
| TIELEV | REAL*4 | Centroid's elevation (altitude) in feet |
| TILAT | REAL*4 | Centroid's latitude (degrees North) |
| TILON | REAL*4 | Centroid's longitude (degrees East) |

| | | |
|---|---|---|
| VERTEX(I) | LOGICAL*1 | Array: flag: if .TRUE., I-th DGZ is one vertex of a convex polygon enclosing all DGZs |
| XDGZ(I) | REAL*4 | Array: tangent-plane X-coordinate of I-th DGZ |
| YDGZ(I) | REAL*4 | Array: tangent-plane Y-coordinate of I-th DGZ |

| Parameter | Where Defined | Where Documented |
|---|---|---|
| MDGZS | MAXIMA.FIF | Reference 3 |

## WEAPON COMMON

COMMON block /WEAPON/ is defined in the FORTRAN source file CBWEAPON.FOR. The file is inserted in any module requiring weapon data using the $INCLUDE metacommand. The file's contents are as follows:

```
INTEGER*2       FIBV, FIPV, FIRV, FRBV, FRPV, FRRV, FRVIN
INTEGER*2       LIBV, LIPV, LIRV, LRBV, LRPV, LRRV
INTEGER*2       MIWVS, MRWVS, NPBVSM, NRVSMS, NRVSPB
INTEGER*2       NRVSWP, NWPUSD

INTEGER*4       IWPDAT

REAL*4          TMBLNC, WPNDAT, YIELD

PARAMETER       (FIBV = 1)
PARAMETER       (FIRV = FIBV + MIBVS)
PARAMETER       (FIPV = FIRV + MIRVVS - 1)

PARAMETER       LIBV = FIRV - 1)
PARAMETER       (LIRV = FIPV - 1)
PARAMETER       (LIPV = FIPV + MIPVS - 1)

PARAMETER       (FRBV = 1)
PARAMETER       (FRRV = FRBV + MRBVS - 3)
PARAMETER       (FRPV = FRRV + MRRVVS - (2 + MSHOBS + 4))

PARAMETER       (LRBV = FRRV - 1)
PARAMETER       (LRRV = FRPV - 1)
PARAMETER       (LRPV = FRPV + MRPVS - 1)

PARAMETER       (MIWVS = LIPV)
PARAMETER       (MRWVS = LRPV)
```

```
COMMON /WEAPON/FRVIN, IWPDAT(MIWVS), NPBVSM, NRVSPB, NRVSWP,
1                     NWPUSD, TMBLNC, WPNDAT(MRWVS), YIELD

EQUIVALENCE (NRVSMS, NRVSWP)
```

| Variable | Type | Definition |
|---|---|---|
| BRLEN | INTEGER*2 | Booster record length [20-character booster name plus 4 times number of 4-byte (integer) values plus 4 times number of 4-byte (real) values] |
| FIBV | INTEGER*2 | Index for first integer booster value in IWPDAT |
| FIPV | INTEGER*2 | Index for first integer PBV value in IWPDAT |
| FIRV | INTEGER*2 | Index for first integer RV value in IWPDAT |
| FRBV | REAL*4 | Index for first real booster value in WPNDAT |
| FRPV | REAL*4 | Index for first real PBV value in WPNDAT |
| FRRV | REAL*4 | Index for first real RV value in WPNDAT |
| FRVIN | INTEGER*2 | First RV in (into a target island) among the RVs on any given weapon. It's either the first or the last RV on a missile; it can't be any RV in between. |
| IWPDAT | INTEGER*4 | Array: contains integer weapon data passed to subroutine CHKSRT |
| LIBV | INTEGER*2 | Index for last integer booster value in IWPDAT |
| LIPV | INTEGER*2 | Index for last integer PBV value in IWPDAT |
| LIRV | INTEGER*2 | Index for last integer RV value in IWPDAT |
| LRBV | REAL*4 | Index for last real booster value in WPNDAT |
| LRPV | REAL*4 | Index for last real PBV value in WPNDAT |
| LRRV | REAL*4 | Index for last real RV value in WPNDAT |
| MIWVS | INTEGER*2 | Maximum number of integer weapon values in IWPDAT |
| MRWVS | INTEGER*2 | Maximum number of real weapon values in WPNDAT |

| | | |
|---|---|---|
| NPBVSM | INTEGER*2 | Number of PBVs per missile |
| NRVSMS | INTEGER*2 | Number of RVs per missile ( = NRVSWP) |
| NRVSPB | INTEGER*2 | Number of RVs per PBV |
| NRVSWP | INTEGER*2 | Number of RVs per weapon ( = NRVSMS) |
| NWPUSD | INTEGER*2 | Number of weapons used |
| PRLEN | INTEGER*2 | PBV record length [20-character PBV name plus 4 times number of 4-byte (integer) values plus 4 times number of 4-byte (real) values] |
| RRLEN | INTEGER*2 | RV record length [20-character RV name plus 4 times number of 4-byte (integer) values plus 4 times number of 4-byte (real) values] |
| TMBLNC | REAL*4 | Minimum time between launches (seconds) |
| WPNDAT | REAL*4 | Array: contains real weapon data for subroutine CHKSR1 |
| YIELD | REAL*4 | Yield in kilotons, read from RV Data File |

# APPENDIX G
# PARAMETER DEFINITIONS

This Appendix identifies and defines all parameters that are unique to the ALM CSC. Other parameters used by the ALM CSC are defined by the MPS Utilities support software and are described in Reference 3 (see MAXIMA.FIF); or are defined in the TARGET CSC Maintenance Manual (Reference 2) and are needed exclusively by TARGET modules used by the ALM to assess sortie feasibility.

A set of parameters are defined and used within COMMON block /WEAPON/. See Appendix F, where the parameters as well as the variables in the block are described. In most (but not all) modules the CHARACTER*6 parameter S is defined to be the module's name; e.g. in RALLOC, the following definition appears:

```
PARAMETER (S = 'RALLOC')
```

All other parameters used by the ALM are defined locally and apply only to the module in which they're defined.

# APPENDIX H
# DATA FLOW DIAGRAM SYMBOLOGY

The Data Flow Diagram (DFD) is a graphic presentation of the partitioning of a system into a network of processes and their data interfaces. The DFD is a representation of the transformation of data. It identifies processes to be performed and the data which flows between processes. The DFD does not indicate processing sequence or logical decisions. A DFD is composed of four basic elements: data flows, processes, data stores and sources/sinks. Figure H-1 depicts a simple DFD. The inputs to and outputs from a process are shown as data flows. The processes represent the transformation of data. The data stores represent a time delayed repository of data. The sources/sinks show, respectively, where the data required by the system comes from and where the data produced by the system goes.

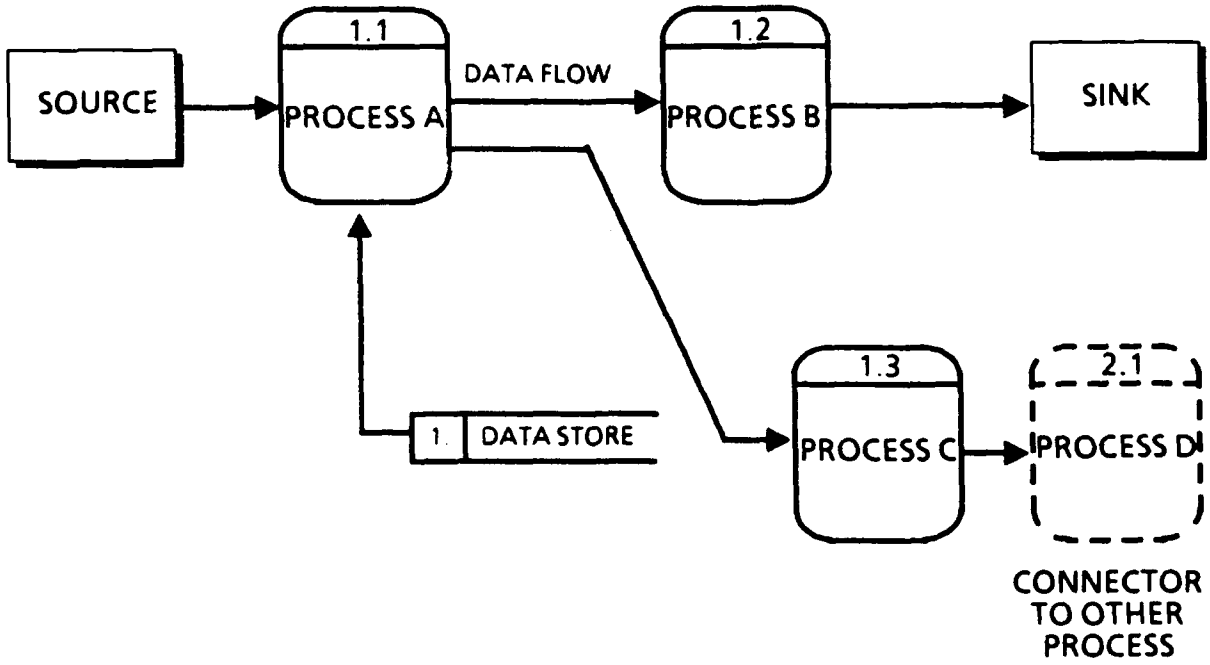



**FIGURE H-1 EXAMPLE DATA FLOW DIAGRAM**

# APPENDIX I
# FUNCTION CHART SYMBOLOGY

The function chart is a graphic presentation of the partitioning of a software system into functions. The function chart illustrates this partitioning by showing function hierarchy, organization and communication. It is a major tool used in software design. It is derived from the data flow diagrams developed during systems analysis. The function chart portrays the software system in manageable and meaningful partitions that can be evaluated with quality factors such as coupling and cohesion; and other design guidelines such as factoring, decision splitting, error handling, editing, system structure, information clustering, initialization, termination, fan-in, fan-out and module packaging. In a function chart, modules are represented by a rectangle with an identifier and descriptive title inside. Functions are represented by a simple rectangle. Connections between modules and functions are represented by lines joining them. The connection usually means that the module has "called" another to perform a function. Figure I-1 depicts a simple function chart. Figure I-2 defines function chart symbols.

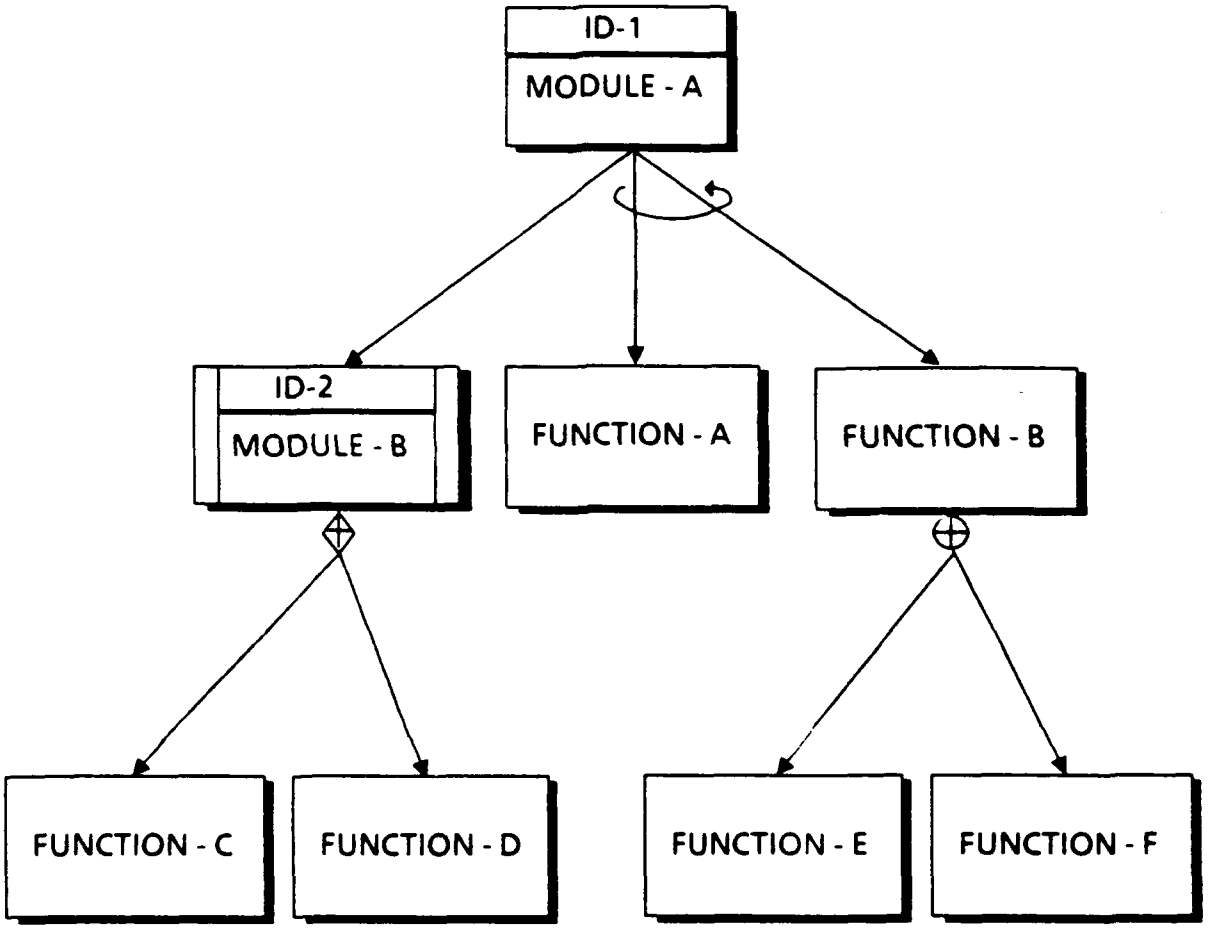
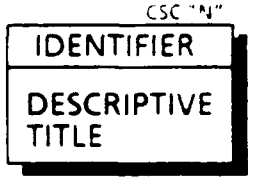



**FIGURE I-1 EXAMPLE FUNCTION CHART**




1 MODULE

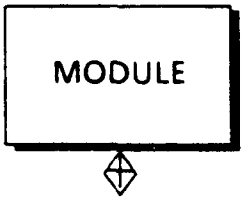
A CONTIGUOUS SEQUENCE OF PROGRAM STATEMENTS, BOUNDED BY BOUNDARY ELEMENTS, HAVING AN AGGREGATE IDENTIFIER. THE CSC "N" TAG IS USED AS AN INDEX TO TRACK THE CSC's STRUCTURE ACROSS MULTIPLE PAGES

2 MODULE - FUNCTION CALL

AN ARROW POINTING TO A MODULE OR FUNCTION DENOTES A REFERENCE TO THE AGGREGATE IDENTIFIER OF THAT MODULE OR FUNCTION

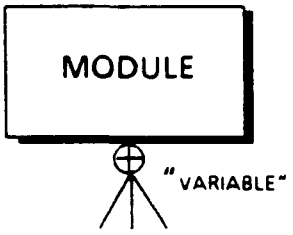
3 ITERATIVE INVOCATION (LOOP)

THE MODULE - FUNCTION REFERENCE ("CALL") IS IMBEDDED IN A LOOPING PROCEDURE. IT MAY ENCLOSE ANY NUMBER OF MODULE / FUNCTION CALLS. THE LOOP CONDITION MAY BE WRITTEN TO THE SIDE OF THIS SYMBOL



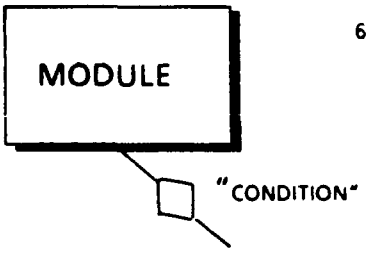

4 SELECTION OR TRANSACTION CENTER (INCLUSIVE "OR")

THE MODULE - FUNCTION CALL IS IMBEDDED IN A DECISION PROCEDURE. IT MAY ENCLOSE ANY NUMBER OF MODULE / FUNCTION CALLS



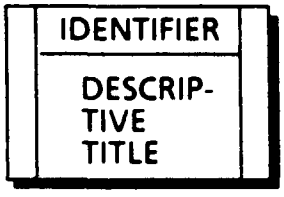

5 EXCLUSIVE "OR" MODULE CALL

ONLY ONE SUB-MODULE OR SUB-FUNCTION IS CALLED DEPENDING UPON THE VARIABLE VALUE




6 CONDITIONAL MODULE CALL

SUB-MODULE OR SUB-FUNCTION CALLED DEPENDING UPON BRANCHING CONDITION




7 PREDEFINED MODULE

ANY PREDEFINED OR PREEXISTING MODULE

**FIGURE I-2 FUNCTION CHART SYMBOLS**

# APPENDIX J
# FLOWCHART SYMBOLOGY

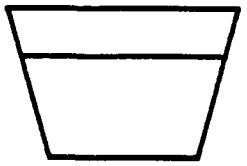
The flowchart is a graphic presentation of a software system that employs symbols and interconnecting lines to depict processing flow. The symbols are used to denote processes and operations, and arrowheads on the interconnecting lines show the direction of data flow and the sequence of operations.

Figure J-1 defines all symbols used in MPS flowcharts. Aside from the minor exceptions noted below, these symbols conform to Federal Information Processing Standards (FIPS). Three nonstandard flowcharting symbols are employed in the MPS to provide superior definition and to enhance clarity. Non-standard symbols are used to denote:

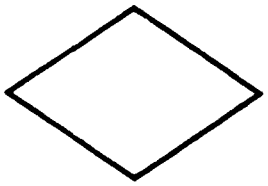
- Do loop boundaries. There is no recognized standard for identifying the scope of do loops. The symbology used is simple and is in widespread use at Kaman.
- Connector (In). The FIPS recognizes a single connector. However, greater clarity is achieved by using different symbols for in, and out, connectors.
- Module call. The FIPS module call symbol has been eschewed, in favor of the indicated symbol, for MPS application. The symbol employed allows one to easily identify module calls based on symbol shape.

Module entry or exit point. A point at which control is passed to a module (ENTER), or a point at which a module relinquishes control (RETURN).

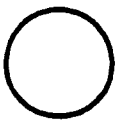
Module process. Specification of an operation or group of operations defined within the module.



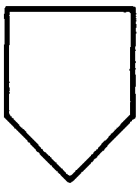
Module call. Invocation of a named process (subroutine) consisting of one or more operations or steps specified elsewhere.



Decision. A point in a module where several paths are possible, and the path selected is dependent on the state of a decision variable or variables.

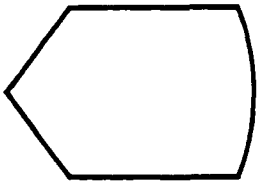


Connector (Out). A junction in the line of flow used to indicate passage of control to another part of the module.

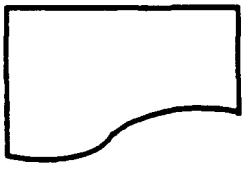


Connector (In). A junction in the line of flow used to denote a point where control is assumed.

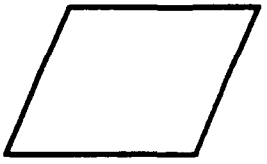
**FIGURE J-1 FLOWCHART SYMBOLS**



Display. I/O function in which information is displayed for human use at the time of processing.



Document. I/O function in which information is output to a hardcopy device.



Input/Output. The process of reading an input file or writing an output file.

Processing flow.

Do loop boundary.

**FIGURE J-1 FLOWCHART SYMBOLS (Concluded)**