AD-E100 069

COMPUTER SYSTEM MANUAL
CSM MM 9-77
VOLUME III
15 APRIL 1978

★
★ ★
★ ★ ★

COMMAND
& CONTROL
TECHNICAL
CENTER

C
C
T
C

THE QUICK-REACTING GENERAL
WAR GAMING SYSTEM (QUICK)

VOLUME III

SORTIE GENERATION SUBSYSTEM

DEFENSE
COMMUNICATIONS
AGENCY

PROGRAM MAINTENANCE MANUAL

78 06 27 113

COMMAND AND CONTROL TECHNICAL CENTER

Computer System Manual CSM-MM-9-77—VOL-3

15 April 1978

THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM

(QUICK) P

Volume III ● Weapon Allocation Subsystem ●

Program Maintenance Manual

SUBMITTED BY:

C. G. THOMPSON
Project Officer

APPROVED BY:

FREDERIC A. GRAF, JR.
Captain, U. S. Navy
Deputy Director, NMCS ADP

Copies of this document may be obtained from the Defense Documentation
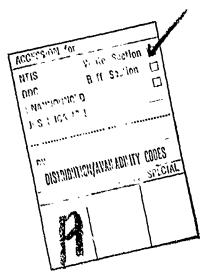Center, Cameron Station, Alexandria, Virginia 22313

Approved for public release; distribution unlimited.

ACKNOWLEDGMENT

CONTENTS

ILLUSTRATIONS

TABLES

ABSTRACT

The computerized Quick-Reacting General War Gaming System (QUICK) will accept input data, automatically generate global strategic nuclear war plans, provide statistical output summaries and produce input tapes to simulator subsystems external to QUICK. QUICK has been programmed in FORTRAN for use on the CCTC HIS 6000 computer system.

The QUICK Program Maintenance Manual consists of four volumes: Volume I, Data Management Subsystem; Volume II, Weapon/Target Identification Subsystem; Volume III, Weapon Allocation Subsystem; Volume IV, Sortie Generation Subsystem. The Program Maintenance Manual complements the other QUICK Computer System Manuals to facilitate maintenance of the war gaming system. This volume, Volume III, provides the programmer/ analyst with a technical description of the purpose, functions, general procedures, and programming techniques applicable to the programs and subroutines of the Weapon Allocation subsystem. Companion documents are:

    a. USERS MANUAL
        Computer System Manual UM 9-77, Volume I
        Computer System Manual UM 9-77, Volume II
        Computer System Manual UM 9-77, Volume III
        Computer System Manual UM 9-77, Volume IV
        Provides detailed instructions for applications of the system

    b. TECHNICAL MEMORANDUM
        Technical Memorandum TM 153-77
        Provides a nontechnical description of the system for senior
        management personnel

SECTION 1. GENERAL

1.1 Purpose

This volume of the QUICK Program Maintenance Manual describes the modules which are part of the QUICK Weapon/Allocation subsystem, detailing the modules, subroutines, and functions which it comprises. The information contained herein is presented on a module-by-module basis. The module-by-module discussions are structured so that a maintenance programmer can understand the program functions and programming techniques. The computer subjects are structured to inform the maintenance programmer of overall system programming techniques and conventions.

1.2 General Description

The Weapon Allocation subsystem operates using the integrated data base as defined by all modules up to PLANSET of the Weapon/Target Identification subsystem and produces a plan using the weapon resources specified to maximize the expected target value destroyed. The subsystem consists of modules PREPALOC, ALOC, EVALALOC, and ALOCOUT, as shown in figure 1. Figure 2 shows the relationship of the Weapon Allocation subsystem to other QUICK subsystems in terms of procedural and information flow.

The modules and supporting subroutines of this subsystem are used to define information for use in later processes and allocate given weapons to targets to optimize expected value destroyed. The integrated data base is updated as each module is exercised in sequence. The final output provides proper inputs necessary to execute the Sortie Generation subsystem.

The first module, PREPALOC, precomputes much of the information required by later processors. In addition, it provides capabilities for planning factor modification and fixed weapon assignment specification.

The next module, ALOC, performs the allocation of weapons to targets. Using a generalized Lagrange multiplier method, an optimal allocation is generated subject to several forms of user-input allocation constraints. These constraints include specification of minimum and maximum desired damage levels, restriction of weapons to specified subsets of the target system, and specification of weapons allocated to specific targets by the user. Within these constraints, the program generates the allocation which maximizes the expected value destroyed in the target system. Module ALOC is also referred to as the allocator.

The main function of module EVALALOC is to provide a summary of the allocation produced in module ALOC and to calculate an expected-value estimate of its results. In addition, the module has the capability of evaluating the effect upon the results of variations in input values for weapon and target parameters. Module EVALALOC may be run either before module ALOCOUT or after module PLANOUT.

1

SUBSYSTEMS	FUNCTIONAL PARTS

```
┌──────────────────────────────────────┐
│      DATA MANAGEMENT SUBSYSTEM        │
├──────────────────────────────────────┤
│  CENTRAL OPERATIONS PROCESSOR         │
│  DATA                                 │
│  EDITDB                               │
│  REPORT                               │
│  SRM                                  │
│  EIM                                  │
│  GENERAL UTILITIES                    │
└──────────────────────────────────────┘
```

EXECUTIVE SOFTWARE

DATA BASE PREPARATION

```
┌──────────────────────────────────────┐
│           WEAPON/TARGET               │
│      IDENTIFICATION SUBSYSTEM         │
├──────────────────────────────────────┤
│  JLM                                  │
│  DBMOD                                │
│  INDEXER                              │
│  PLANSET                              │
└──────────────────────────────────────┘
```

```
┌──────────────────────────────────────┐
│      WEAPON ALLOCATION SUBSYSTEM      │
├──────────────────────────────────────┤
│  PREPALOC                             │
│  ALOC                                 │
│  EVALALOC                             │
│  ALOCOUT                              │
└──────────────────────────────────────┘
```

PLAN GENERATION

```
┌──────────────────────────────────────┐
│      SORTIE GENERATION SUBSYSTEM      │
├──────────────────────────────────────┤
│  FOOTPRNT.                            │
│  POSTALOC                             │
│  PLANOUT                              │
│  PLOTIT                               │
└──────────────────────────────────────┘
```
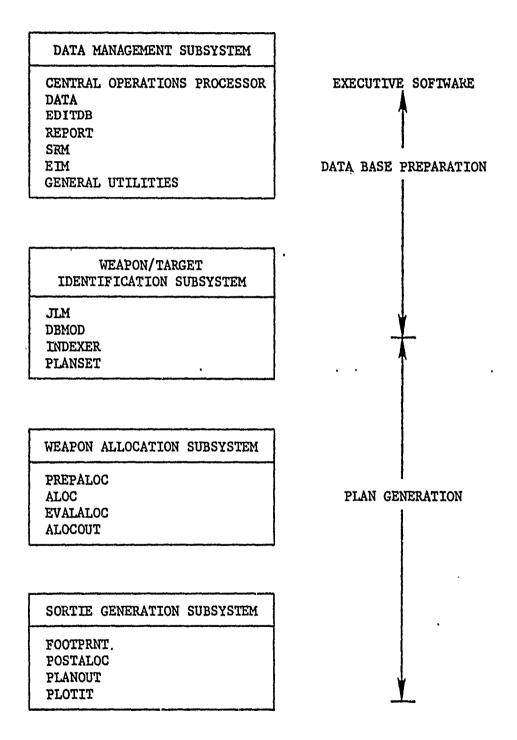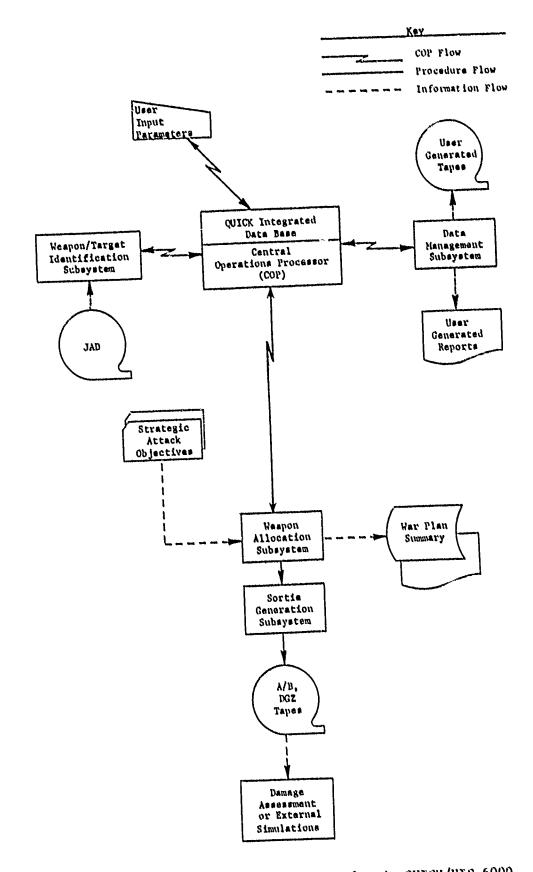
Figure 1. Major Subsystems of the QUICK System

COP Flow

Procedure Flow

Information Flow

User
Input
Parameters

User
Generated
Tapes

QUICK Integrated
Data Base

Central
Operations Processor
(COP)

Weapon/Target
Identification
Subsystem

Data
Management
Subsystem

JAD

User
Generated
Reports

Strategic
Attack
Objectives

Weapon
Allocation
Subsystem

War Plan
Summary

Sortie
Generation
Subsystem

A/B,
DGZ
Tapes

Damage
Assessment
or External
Simulations

Figure 2. Procedure and Information Flow in QUICK/HIS 6000

ALOCOUT optimizes the location of aim points for target complexes and collects all the strikes assigned to each weapon group by the allocator so that detailed plans for each group can be formulated by FOOTPRNT and POSTALOC.

1.3 Organization of Maintenance Manual, Volume III

Each major section of this manual details a module along with the sub-routines and functions which comprise the module. Major subsections are:

a. Module input - details what chains must be created prior to module execution

b. Module output - details what chains will be updated by each module

c. Functional description - details the macro function of the module and the associated major subroutines

d. Common blocks - detail the contents of all internal common blocks. All common blocks used to communicate with the COP are given in Program Maintenance Manual, Volume I, appendix A. These are: C10, C15, C20, C30, C40, C50, ERRCOM, INS, IPQT, OOPS, STRING

Within the QUICK system the COP is viewed as the operating program. Based on user direction, the COP will execute overlay links or modules which perform the objectives of the user requests. Each overlay link is called through knowledge of the command verb and within each link the first subroutine is called ENTMOD (for entry module). That is, there are as many subroutines called ENTMOD as there are modules. Con-fusion is avoided by executing the correct overlay link. Subroutine discussion, then, is initiated with ENTMOD whose meaning, or function, varies according to the overlay link.

Comments on the QUICK integrated data base can be found in Program Main-tenance Manual, Volume I, section 2. It will be assumed within this manual that the reader has an understanding of QUICK's data base.

Each section discusses the computer programming aspect for the appro-priate modules. Attached appendixes presents mathematical algorithms employed within the Weapon Allocation Subsystem.

4

SECTION 2. PREPALOC MODULE

2.1 Purpose

The purpose of this module is to perform preliminary calculations on the
weapon and target data as stored within the integrated data base. The
output data from PREPALOC will be in a form convenient for use by the
remaining processors of the plan generator. In addition, the user may
select options to modify some of the data at this stage of processing.

Module PREPALOC has three major capabilities: updating of geographic
and weapon group data, modification of target values and damage con-
straints and preparation of data for the fixed weapon assignment capa-
bility of program ALOC.

The basic raw geographic data must be data base defined prior to any
execution of PREPALOC. Using this data, PREPALOC will calculate and
store distances and attrition between all doglegs for use within proces-
sors to follow. Based on user inputs, the number of weapons within
bomber or missile MIRV weapon groups may be adjusted.

The second major capability of this module is the modification of the
target characteristics, VTO, MINKILL, and MAXKILL. VTO is the value of
the target relative to all the others. MINKILL is the minimum fraction
of target value that must be destroyed, and MAXKILL is the maximum de-
sired fraction of target value destroyed. Any of these parameters may
be changed for any target. The change requests can change these parame-
ters for a single target or for a set of targets. The set of targets
for which a change is requested is identified by target class, type, an
individual identifier (target designator code (DESIG)) or any combina-
tion of these. For complex targets, the class, type, designator code,
and index of each component will be checked to determine if a target
parameter for the complex is to be changed.

In addition, the user can specify the height of burst to be used in any
weapon/target combination. The user selects either a ground burst or an
air burst at the optimal air burst height. In the absence of any user
specification, the most damaging height of burst is used.

The third major capability is the request for allocation of specific
weapons to specific targets. This fixing of weapons to targets enables
the user to determine part of the weapon allocation while leaving the
allocation module free to determine the remaining allocation. In addi-
tion, the time of arrival at target or launch salvo number can be fixed
for missile weapons. This information will be passed to module PLANOUT
which will adjust launch time accordingly. The fixing of weapons remains
in effect for the remainder of the plan generation process. Later pro-
grams will retain the assignments as best possible. (For example, it is
possible to fix a set of weapons from a weapon group with multiple inde-
pendently targetable reentry vehicles (MIRV) in such a manner that there

5

are no feasible footprints that cover that target set adequately. In that case, some of the fixed assignment requests must be ignored.)

2.2 Input

The entire integrated data base must be completely defined prior to PREPALOC execution. This includes the storage of all targets, related geographic data, weapon type and group characteristics and other supporting data such as warhead and payload information.

2.3 Output

Creation of new records occurs if the user specified fixed assignments. For these cases, records called 'ASSIGN' which stores the fixed assignment under the proper target and weapon group linkage are created. Also, new records (RDDIST, TPDIST and TDDIST) defining the distance between each depenetration corridor recovery base intersection, each penetration corridor target intersection, and each target and the optimal depenetration corridor, are created.

If any of the target modification options are employed, the necessary target records will be modified accordingly. Also, weapon group attributes may be altered if overallocation is specified.

For all executions, distances and attrition rates associated with each penetration corridor will be calculated and stored. Similarly, depenetration distances between doglegs as well as the distance from depenetration corridor to recovery bases are stored.

2.4 Concept of Operation

The flow of execution within PREPALOC is strictly sequential. Subroutine ENTMOD reads user's inputs, stores values, and executes each major (see below) subroutine. Once a major subroutine has been executed and a return to ENTMOD made, that subroutine will not be executed again.

2.5 Identification of Subroutine Functions

2.5.1 Subroutine FACTORCG. Called by ENTMOD immediately after all text English adverbs have been processed. FACTORCG will read any user target modification request and modify the appropriate IDS records.

2.5.2 Subroutine FIXWEP. User fixed assignments are read by this subroutine and the IDS record called ASSIGN created.

2.5.3 Subroutine PENROUT. For each penetration corridor, distance and attrition for each dogleg within a corridor is stored.

2.5.4 Subroutine DEPROUT. This subroutine calculates depenetration corridor, recovery base distance. For each depenetration recovery base intersection, a new IDS record called RDDIST is created.

2.5.5 <u>Subroutine WEPREP</u>. Weapon group counts are updated, if the user specified overallocation.

2.5.6 <u>Subroutine TGTPREP</u>. In addition to performing summary prints, fixed assignment records modification continues by adding the salvo number, if necessary.

2.6 PREPALOC Internal Common Blocks

All common blocks used internally by PREPALOC are given in table 1. For definition of common blocks that communicate with the COP, see Program Maintenance Manual, Volume I.

Table 1. Module PREPALOC Common Blocks
(Part 1 of 2)

ASSOCIATED COMMON	VARIABLE OR ARRAY	DESCRIPTION
ASMTYPE	ASMTYPE(20)	ASM type names
CLASSCOM	CLASSNAM(15)	Target class names
	CLASSREF(15)	Header reference codes for target classes
	NTARCLS	Number of target classes
CRLENGTH	CRLENGTH(30)	Precorridor distance; indexed by penetration corridor number
DISTEF	DISTEF(50)	Length of depenetration corridor
	DISTEG(50)	Distance from depenetration corridor entry to recovery point
GAMFLAG	GAMFLAG(9)	Flag indicating value for planning parameter was input. Planning parameters are: INITSTRK, CORMSL, CORBOMB, PEXBOMB, EXNBOMB, PEXMIRV, EXNMIRV, PEXMISS, EXNMISS
IGPREF	IGPREF(250)	Reference codes for weapon groups
IONPRT	IGEOPRT, ICRPPRT	If zero, suppress nonstandard geography and weapon group print
	ITARPRT(2,2)	Lower and upper target number print requests
ISIMTYPE	ISIMTYPE(100)	Weapon system type name
IWEPREF	IWEPREF(100)	Reference codes for weapon type records
NFIXREQ	NFIXREQ	Number of fix assignments implemented
NUMCOR	NCORR	Number of penetration corridors
	NDPEN	Number of depenetration corridors
PAYTYPE	PAYTYPE(40)	Payload type names
REPOINTS	RFLAT(20) RFLONG(20)	Refuel latitude and longitude points

8

Table 1.   (Part 2 of 2)
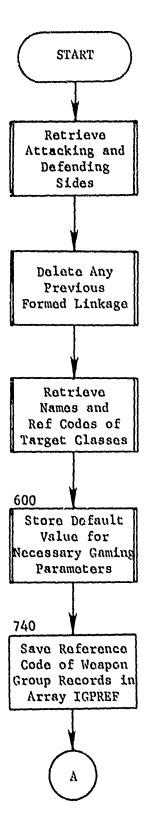
ASSOCIATED COMMON	VARIABLE OR ARRAY	DESCRIPTION
SUMNEW	SUMNEW	Sum of values after implementing value change requests
WAROUT	IWARFL	Logical unit number for war gaming output
WHTYPE	WHTYPE(50)	Warhead type names

2.7 Subroutine ENTMOD

PURPOSE: To control overall flow of processing

ENTRY POINTS: ENTMOD (first subroutine called when overlay link
                           PREP is executed

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C20, C25, C30, CLASSCOM, ERRCOM, GAMFLA,
                           IGPREF, IONPRT, ISIMTY, IWEPRE

SUBROUTINES CALLED: CINSGET, DEPROUT, DIRECT, DLETE, FACTORCG, FIXWEAP,
                           HDFND, HEAD, INSGET, KEYMAKE, MODFY, NEXTTT, RETRV,
                           TGTPREP, WEPPREP
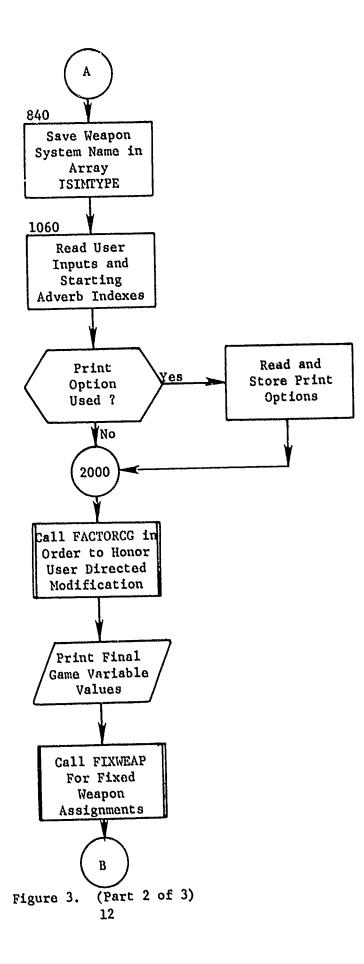
CALLED BY: COP

Method:

ENTMOD retrieves and stores target class names and associated reference
codes into arrays CLASSNAM and CLASSREF. Upon storage completion, each
major subroutine (see figure 3) is executed and then processing ends for
PREPALOC.

START

Retrieve
Attacking and
Defending
Sides

Delete Any
Previous
Formed Linkage

Retrieve
Names and
Ref Codes of
Target Classes

600
Store Default
Value for
Necessary Gaming
Parameters

740
Save Reference
Code of Weapon
Group Records in
Array IGPREF

A

Figure 3. Module PREPALOC (Part 1 of 3)

A

840 → Save Weapon System Name in Array ISIMTYPE

1060 → Read User Inputs and Starting Adverb Indexes

Print Option Used ? — Yes → Read and Store Print Options

No ↓

2000

Call FACTORCG in Order to Honor User Directed Modification

Print Final Game Variable Values

Call FIXWEAP For Fixed Weapon Assignments

B

Figure 3. (Part 2 of 3)

12

```
                    ( B )
                     │
                     ▼
         ┌───────────────────────┐
         │    Call PENROUT       │
         │  for Penetration      │
         │      Corridor         │
         │    Calculations       │
         └───────────────────────┘
                     │
                     ▼
         ┌───────────────────────┐
         │    Call DEPROUT       │
         │  for Depenetra-       │
         │   tion Corridor       │
         │    Calculations       │
         └───────────────────────┘
                     │
                     ▼
         ┌───────────────────────┐
         │    Call WEPPREP       │
         │    for Weapon         │
         │    Calculations       │
         └───────────────────────┘
                     │
                     ▼
         ┌───────────────────────┐
         │    Call TGTPREP       │
         │    for Target         │
         │    Calculations       │
         └───────────────────────┘
                     │
                     ▼
          ╱──────────────────╲
         ╱       Print         ╲
        ╱        Target         ╲
        ╲       Planning        ╱
         ╲    Factor List      ╱
          ╲──────────────────╱
                     │
                     ▼
         ┌───────────────────────┐
         │    Accumulative       │
         │    Target Value       │
         │   by Class Into       │
         │      CLASVAL          │
         └───────────────────────┘
                     │
                     ▼
          ╱──────────────────╲
         ╱       Print         ╲
        ╱       Geography        ╲
        ╲         Data           ╱
          ╲──────────────────╱
                     │
                     ▼
            ╭──────────────╮
            │    RETURN     │
            ╰──────────────╯
```

Figure 3.   (Part 3 of 3)

2.8 Subroutine DEPROUT

PURPOSE: To compute and print depenetration corridor data

ENTRY POINTS: DEPROUT

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C30, DISTEF, ERRCOM, IONPRT, OOPS,
                        REPOINTS

SUBROUTINES CALLED: DISTF, HDFND, NEXTTT, RETRV, STORE, ORDER
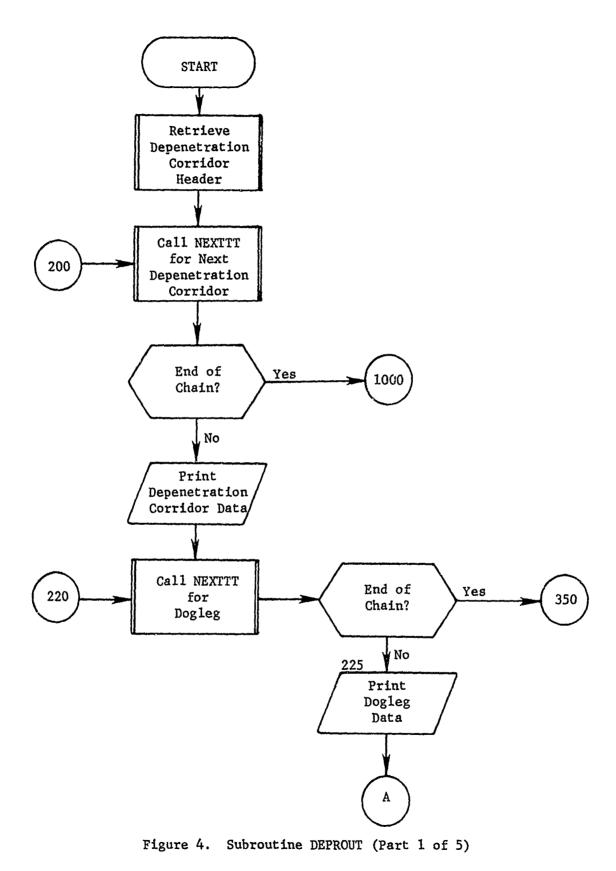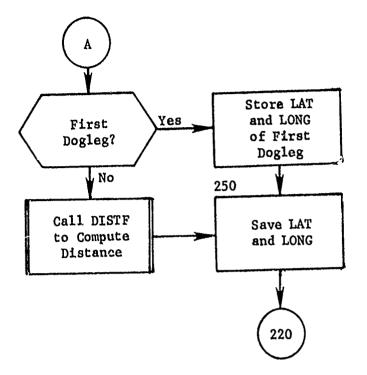
CALLED BY: ENTMOD (of PREPALOC)

Method:

DEPROUT calculates and stores depenetration corridor information and
recovery base information associated with each depenetration corridor
as well as storage of refuel points.

Individual depenetration corridors are chained and corridor length deter-
mined and stored. Following which, each recovery base associated with
the corridor is queried and distance from depenetration exit point to
recovery bases stored.

Subroutine DEPROUT is illustrated within figure 4.

Figure 4.   Subroutine DEPROUT (Part 1 of 5)

Figure 4. (Part 2 of 5)

16

Figure 4. (Part 3 of 5)

17

```
         ( 570 )
            |
            v
    +-----------------+
    | Sort            |
    | Distances       |
    | to Recovery     |
    | Bases           |
    +-----------------+
            |
            v
    +-----------------+
    | Store           |
    | Distance        |
    | Within Record   |
    | RDDIST          |
    +-----------------+
            |
            v
    +-----------------+
    |Compute Distance |
    | From Corridor   |
    |Entry to Recovery|
    | Base (DISTEG)   |
    +-----------------+
            |
            v
         ( 200 )
```

Figure 4.   (Part 4 of 5)

Figure 4. (Part 5 of 5)

19
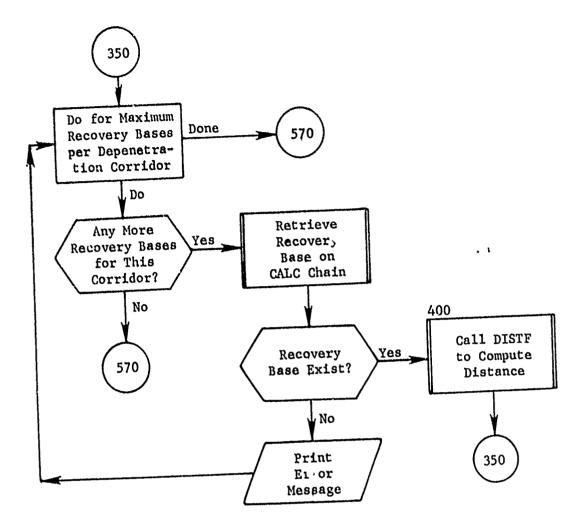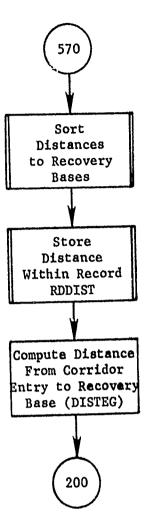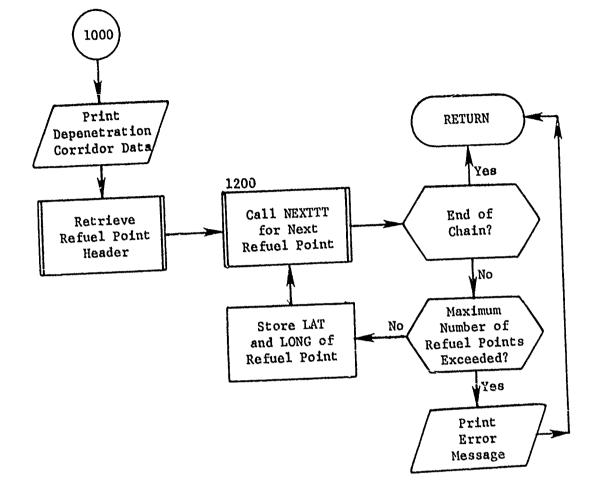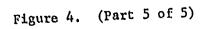
2.9 Subroutine FACTORCG

PURPOSE:	To read and process user input factor change requests
ENTRY POINTS:	FACTORCG
FORMAL PARAMETERS:	LOCSET - Pointer to INSGET's arrays for location of 'SETTING' adverb
COMMON BLOCKS:	C10, C15, C30, CLASSCOM, ERRCOM, GAMFLAG, ISIMTYPE, OOPS
SUBROUTINES CALLED:	CINSGET, HDFND, ITLE, MAKECHG, MODFY, NEXTTT, RETRV
CALLED BY:	ENTMOD (of PREPALOC)

Method:

FACTORCG reads the user directed factor change requests and calls sub-routine MAKECHG to implement each change. VALUE, MINKILL, and MAXKILL may be changed. Also, if it is desired to override the calculated optimal height-of-burst (attribute IDHOB) this may also be accomplished. These attributes may be reset through various combinations of target data subsetting. Permissible requests include the setting of attributes.

o DESIG - A single target record is to be updated

o TYPE - All targets that equal the input TYPE are to be updated

o CLASS - All targets that equal the input target class name are to be updated

o CNTRYL - The height-of-burst of all targets located within the input country location are to be updated

o IREG - The height-of-burst of all targets located within the input region are to be updated

The last two requests recognize only height-of-burst requests. Other-wise, any combination of target subsetting is permissible but there is a ranking order in the final storage of input values. The order of priority is: DESIG, TYPE, CLASS, CNTRYL, IREG. That is, if a given target is to have an attribute updated by more than one input target set, the cited order applies. For instance, consider inputs that request updating attribute VALUE to 40 for TYPE=TITAN and to 50 for CLASS=MISSILE. Since TITAN's are in fact missiles, the ranking order resolves any conflicts and, accordingly, all target records where CLASS=MISSILE will have VALUE=50 except for those records where TYPE=TITAN in which case VALUE is set to 40.

All code implemented within FACTORCG up to statement number 1500 reads the input requests (by calling subroutine INSGET) and upon proper definition, the changes are made by calling subroutine MAKECHG.

Changes are not made as read. This is because of the rule that once a complex target has been changed, its components may not be separately changed for the same factor. (All of its components are, however changed by a similar ratio as the complex target as part of the change to the complex target.) Changes are made first for individual targets, then for targets based on TYPE, then CLASS and so on. Local parameter ICRIT equals 1, 2, 3, 4, or 5 defining if the latest input defines targets to be updated on a region, country, class, type, or DESIG, respectively basis. A second parameter called NOWCRIT (and assuming the same values as ICRIT) is initially set to 5 to show that only targets specified by DESIG are to be updated. If ICRIT does not equal NOWCRIT, processing is delayed and if this condition exists array ICRFIRST (ICRIT) is set to the first location into INSGET's array that defines a change request at ICRIT level.

Upon processing all inputs, NOWCRIT is decremented by one and processing reinitiated for that level of priority.
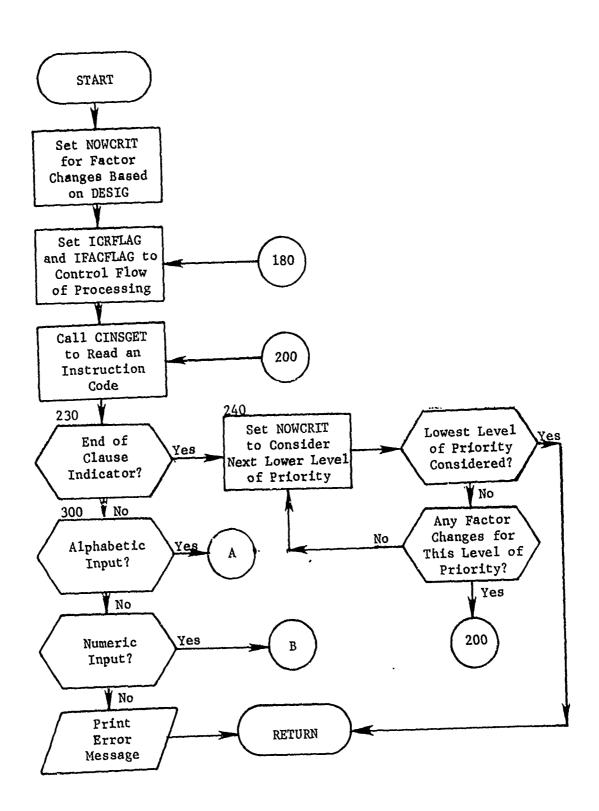
The generalized nature of inputs does not demand any one order of input definition. That is, for say a DESIG, VALUE intersection setting, the user may within the command sentence define either DESIG or VALUE initially; neither order of input is more powerful than the other. Therefore, the design must recognize that changes may not be honored until necessary data has been read. Accordingly, the following parameters are used to control processing:

o   IFACTOR - =1, updating attribute VALUE
              =2, updating attribute MINKILL
              =3, updating attribute MAXKILL
              =4, updating attribute IDHOB

o   ICRFLAG - =0, a subset of targets pertaining to an input factor
                   has not been read
              =1, a subset of targets has been read and was of the
                   correct processing priority
              =2, a subset of targets has been read but it was not
                   the correct processing priority (or a typographi-
                   cal error was detected). Request change is de-
                   layed

o   IFACFLAG - =0, an attribute to be changed has not been read
               =1, an attribute to be changed has been read and may
                   be processed
               =2, an attribute to be changed has been read but a
                   typographical error exists

21

Upon proper conditions, a change request is honored. If an individual target record is to be updated (ICRIT=1), the record is retrieved and MAKECHG is called for target modifications. Otherwise the correct target record level must be retrieved, for subroutine MAKECHG will chain the individual target record chain with the assumption the next highest IDS record level has been retrieved.

Subroutine FACTORCG is illustrated in figure 5.

Figure 5. Subroutine FACTORCG (Part 1 of 10)
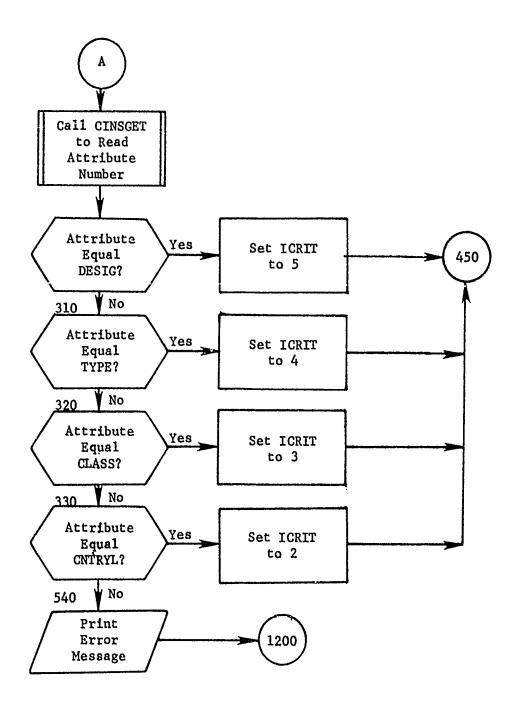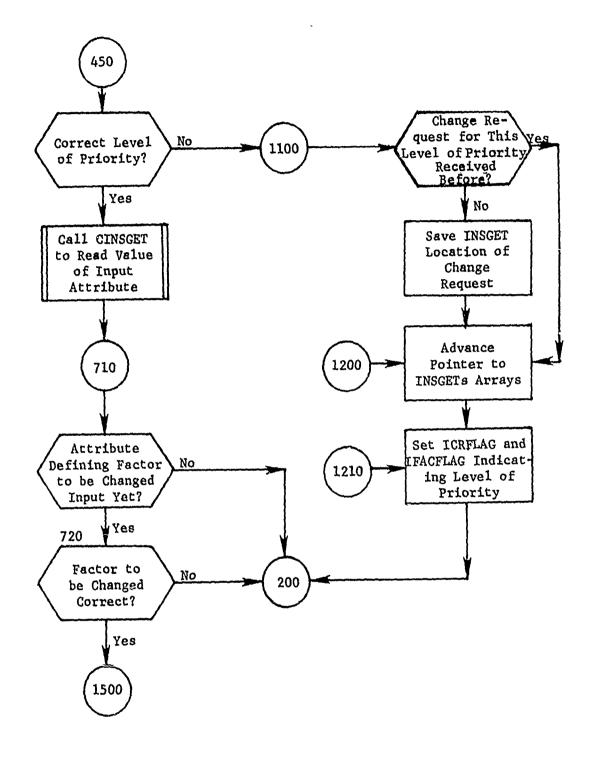
Figure 5. (Part 2 of 10)

Figure 5. (Part 3 of 10)

Figure 5. (Part 4 of 10)

Figure 5. (Part 5 of 10)

```
        ( 760 )
           │
           ▼
    ┌─────────────┐
    │ Call CINSGET│
    │ to Read Change│
    │ Value Factor│
    └─────────────┘
           │
           ▼
      Attribute
   Defining Subset      No
    of Targets      ─────────────┐
    Input Yet?                   │
       │ Yes                     │
       ▼                         │
  764                            │
   Was Attribute                 │
   Defining Subset   No          ▼
   of Targets of  ─────────►  ( 200 )
   Correct Level
   of Priority?
       │ Yes
       ▼
  768
  ┌─────────────┐
  │ Adjust Value│
  │ of Factor If│
  │ Not in Correct│
  │    Range    │
  └─────────────┘
       │
       ▼
   ( 1500 )
```
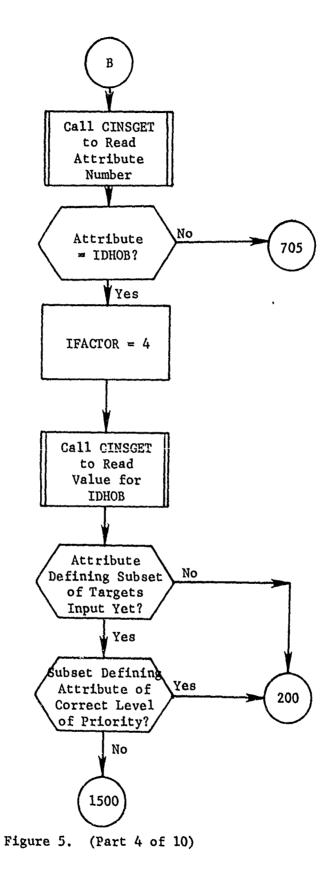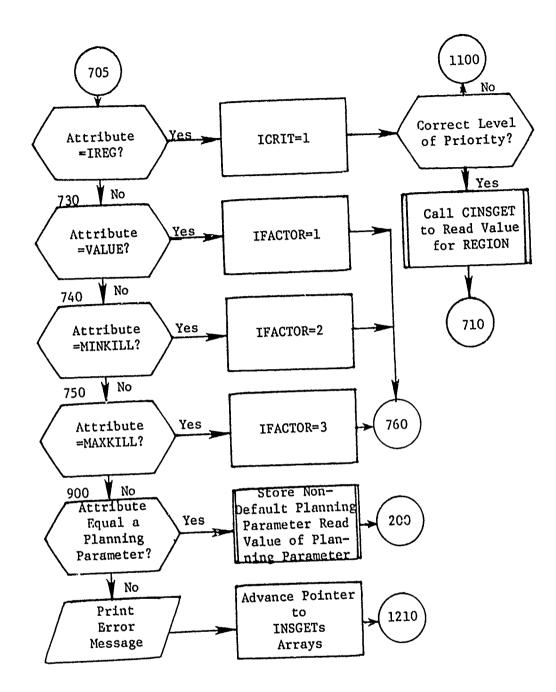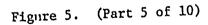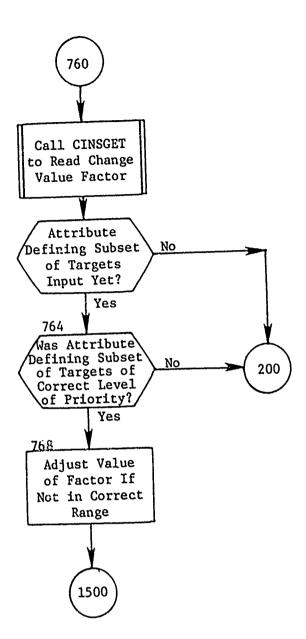
Figure 5.   (Part 6 of 10)

28
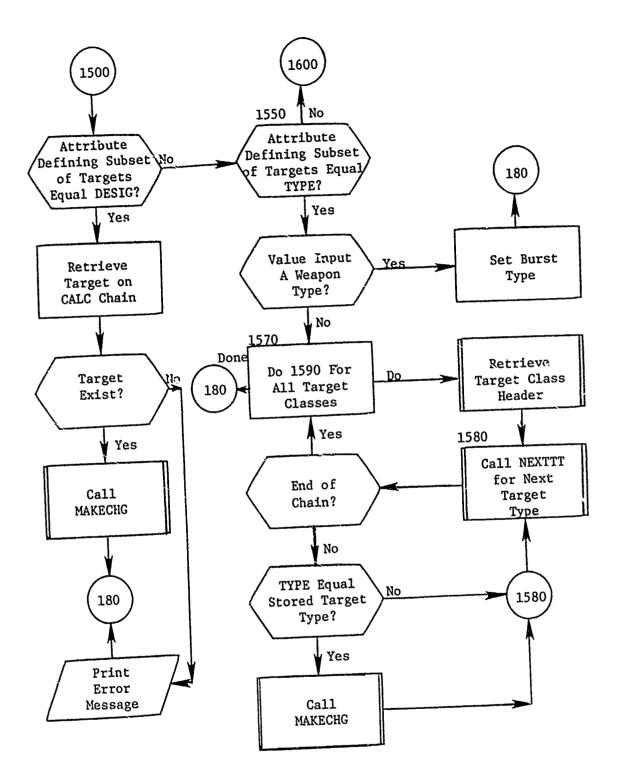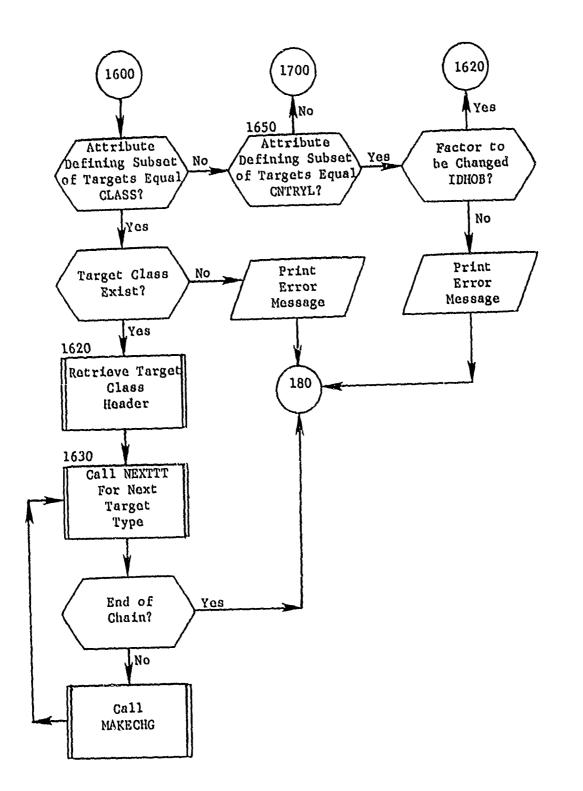
Figure 5. (Part 7 of 10)

Figure 5. (Part 8 of 10)
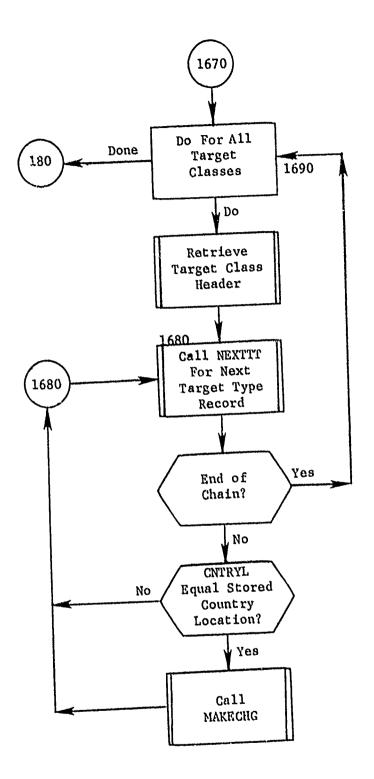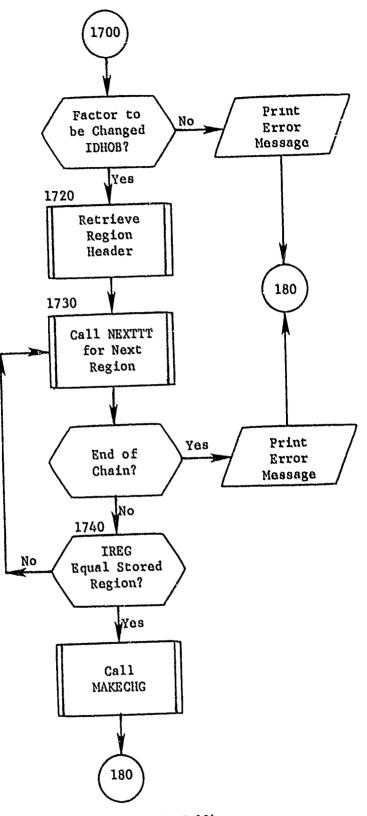
Figure 5. (Part 9 of 10)

31

Figure 5. (Part 10 of 10)
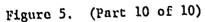
2.10   Subroutine FIXWEP

PURPOSE: To read and process fix assignment requests

ENTRY POINTS: FIXWEP

FORMAL PARAMETERS: LOCFIX - Pointer to INSGET's arrays for location
           of 'FIX' adverb

COMMON BLOCKS: C10, C30, ERRCOM, IGPREF, OOPS

SUBROUTINES CALLED: CINSGET, DIRECT, HEAD, KEYMAKE, MODFY, NEXTTT,
           RETRV, STORE

CALLED BY: ENTMOD (of PREPALOC)

Method:

Subroutine FIXWEP creates record type FIXASG records for all user re-
quested fix assignments as defined by a clause introduced by the adverb
FIX. Each created record stores the weapon group number and, if defined,
the downtime or salvo number.

The FIX clause recognizes attributes DESIG, GROUP, and the optional down-
time attributes ARRIVE or SALVO.   Insertion of values for these two
attributes (and optionally four) is sufficient for record creation. How-
ever, to provide ease of input values, the user may specify two DESIG's
for one GROUP input. This mode of operation implies that a range of
fixed assignments will be made and that range defined as all the targets
that fall within the interval of the two DESIGs. The subroutine first
reads INSGET's arrays and checks for values of necessary attributes and
then, if error free, a fixed assignment record is created.

Since attributes may be defined in any order (that is GROUP appears be-
fore DESIG or vice-versa) the implemented design must delay processing
until all attributes necessary are defined. The following local param-
eter control input processing:

  o BEGDESIG - Zeroed at the start of the input search for each
          phrase and reset to the value of the first input
          DESIG

  o TERDESIG - Set to the value of the second DESIG, if it exists

  o KGROUPNO - Requested weapon group number

  o IOPTION - =1, at start of each phrase;
          =2, if a range of DESIG's was input
          =3, if ARRIVE defined
          =4, if range of DESIG's and ARRIVE defined

33

Fixed assignments are now stored within the data unless:

a. If an attempt is made to fix a bomber weapon on a target with more than 30 fix requests, the request will be ignored, since only targets with terminal ballistic missile defenses undergo a saturation missile attack. The allocation procedure will not allow a bomber to participate in such an attack.

b. If an attempt is made to fix more weapons than are present in a group, the excess requests are ignored.

c. If two DESIGs were input and the alpha portions did not match, the requests are ignored.
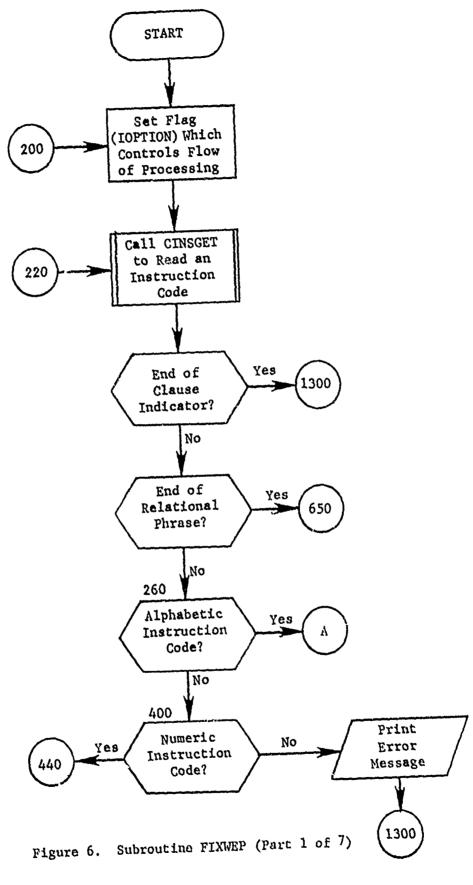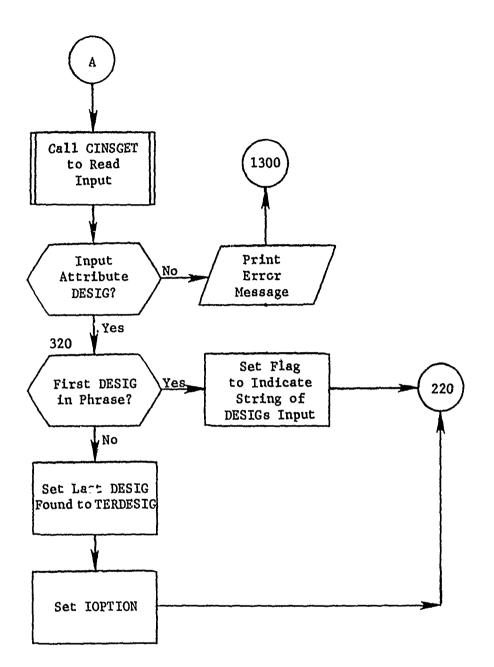
d. If non-lead (either for complexes or multiplier) targets were to be fixed and the representative target could not be found, the request is ignored.

e. If the target record or if the group number is invalid, the request is ignored.

Subroutine FIXWEP is illustrated in figure 6.
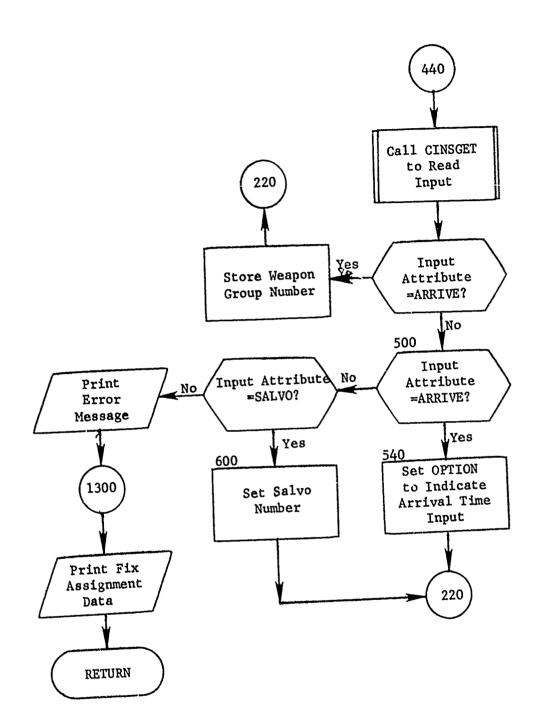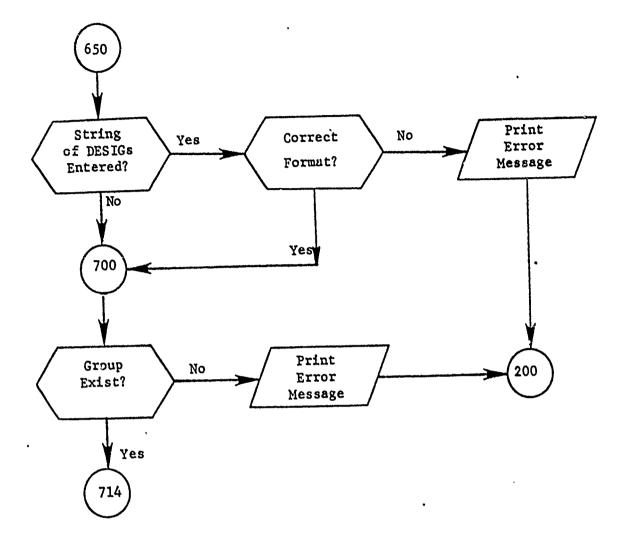
Figure 6.   Subroutine FIXWEP (Part 1 of 7)

35

Figure 6. (Part 2 of 7)

Figure 6. (Part 3 of 7)

Figure 6. (Part 4 of 7)

Figure 6. (Part 5 of 7)

Figure 6. (Part 6 of 7)

Figure 6. (Part 7 of 7)
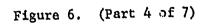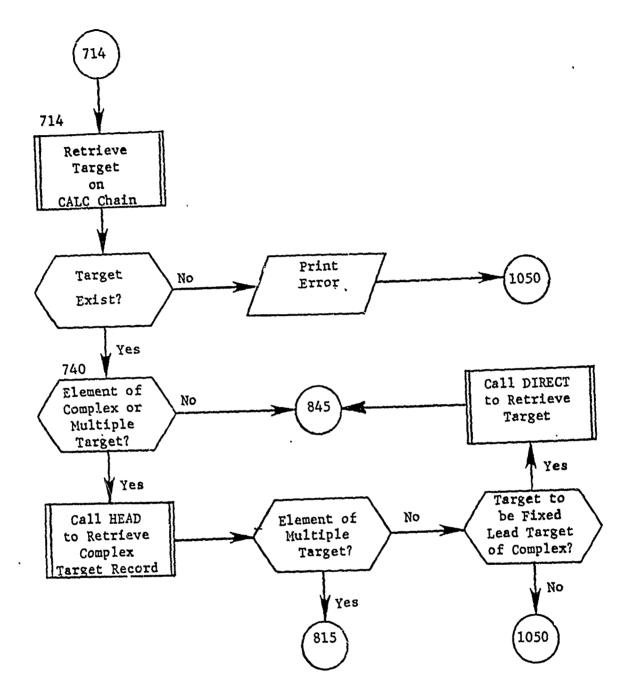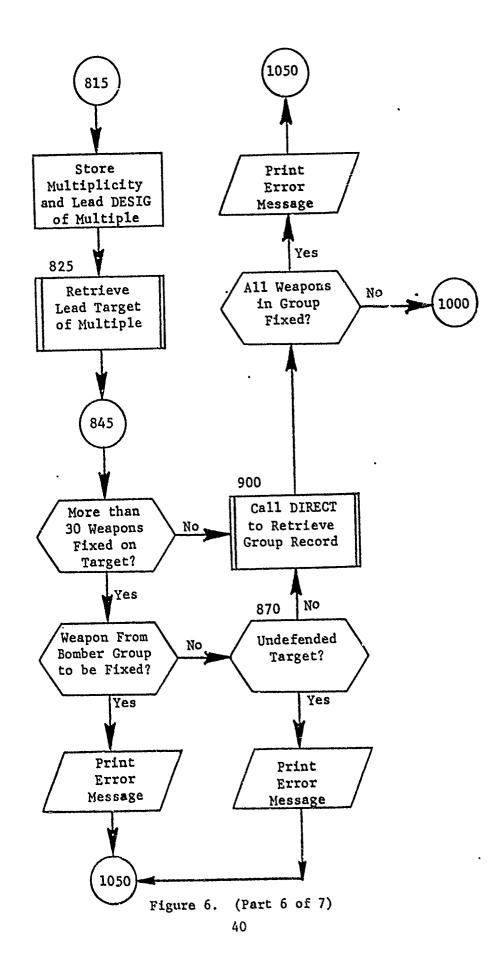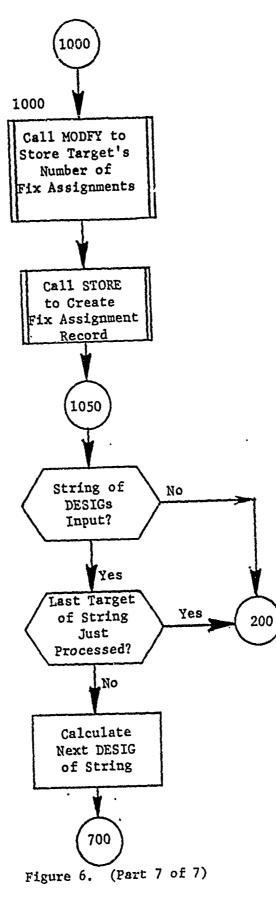
2.11 Subroutine MAKECHG

PURPOSE: To make changes to the data base for valid factor
                        change requests

ENTRY POINTS: MAKECHG

FORMAT PARAMETERS: ICRIT -Indicates type of subset of targets to be
                               changed (1, 2, 3, 4, 5 for region, country
                               location, class, type, single target,
                               respectively)
                        CRWORD -Value of subset
                        IFACTOR-Indicates attribute to be changed (1, 2,
                               3, 4 for VALUE, MINKILL, MAXKILL, height
                               of burst, respectively)
                        FACWORD-New value of attribute

COMMON BLOCKS: C10, C30, SUMNEW

SUBROUTINES CALLED: DIRECT, HEAD, MODFY, NEXTTT

CALLED BY: FACTORCG

Method:

Subroutine MAKECHG modifies target record(s) as directed by subroutine
FACTORCG. Depending upon the level of processing (parameter ICRIT)
either the TGTTGT or TGTREG chain is queried. When FACTORCG calls
MAKECHG the correct next highest IDS record has been defined. By
stepping through the targets, modification is accomplished.

If a factor is changed on the main target of a complex, the factors for
each component are changed appropriately. For example, if the value of
the complex is doubled, the value of each component is doubled. This
procedure is followed to allow the user to change one factor for the
entire complex and other factors by component. If a factor is changed
for a complex component individually (not through the main target)
that factor is checked on all the remaining components. The method
used in determining VALUE, MINKILL, and MAXKILL for a complex target
is identical to that used in subroutine CALCOMP of module PLANSET. For
height-of-burst specifications, only the main target of a complex is
checked.

If attribute VALUE is the factor to be updated, parameter SUMNEW is re-
defined for further normalizing calculations.

Whenever a factor is changed for a target, the attribute IDHOB is
'marked' so that the same factor is not changed again by a change
request of lower priority. IDHOB is 'marked' by packing a 1 in an
octal digit as shown below:

42

IFACTOR	Factor Changed	Position of Octal Digit
1	VALUE	000 001 000 000
2	MINKILL	000 010 000 000
3	MAXKILL	000 100 000 000
4	IDHOB	001 000 000 000

The formula

$$IDHOB/MSHIFT(IFACTOR)-8*IDHOB/MSHIFT(IFACTOR+1)$$

merely yields the value to the appropriate octal digit (of IDHOB) given IFACTOR.
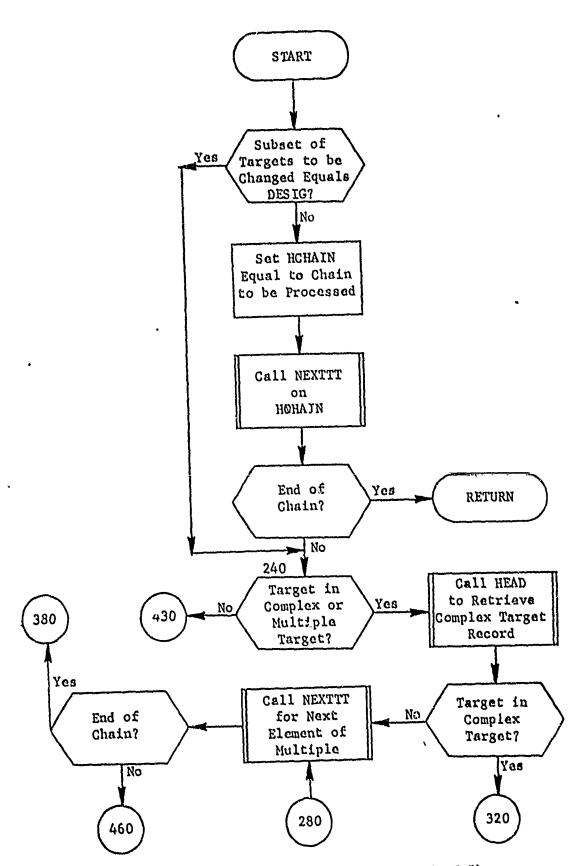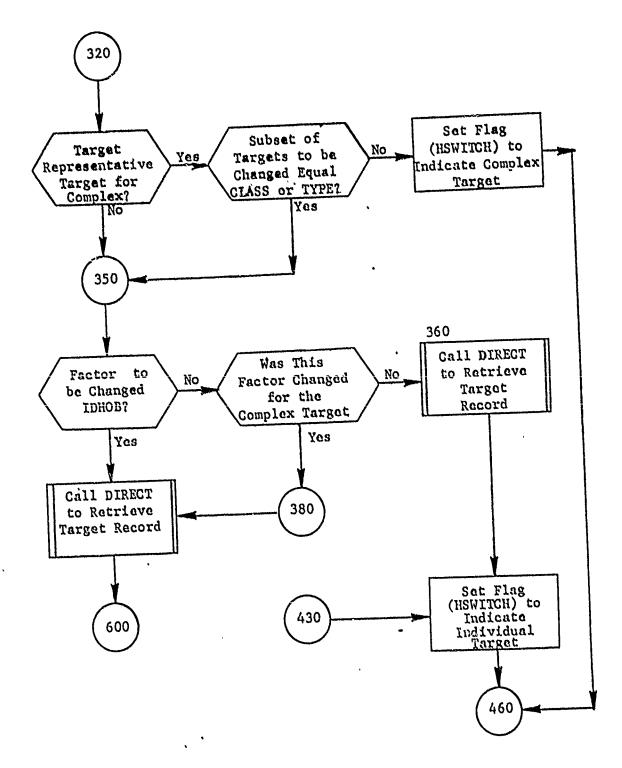
Subroutine MAKECHG is illustrated in figure 7.
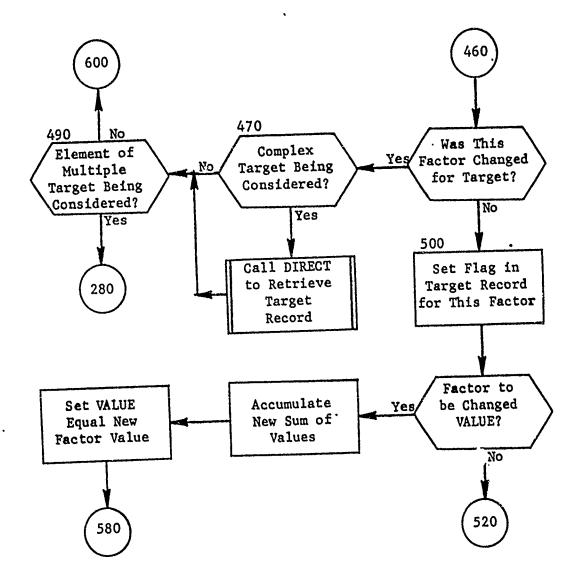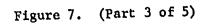
Figure 7. Subroutine MAKECHG (Part 1 of 5)

Figure 7. (Part 2 of 5)

600

490   No
Element of
Multiple
Target Being
Considered?
   Yes

280

470
Complex
Target Being
Considered?
   No
   Yes

Call DIRECT
to Retrieve
Target
Record

460

Was This
Factor Changed
for Target?
   Yes   No

500
Set Flag in
Target Record
for This Factor

Set VALUE
Equal New
Factor Value

Accumulate
New Sum of
Values

Factor to
be Changed
VALUE?
   Yes   No

580

520

Figure 7. (Part 3 of 5)

46

Figure 7. (Part 4 of 5)

Figure 7. (Part 5 of 5)

48
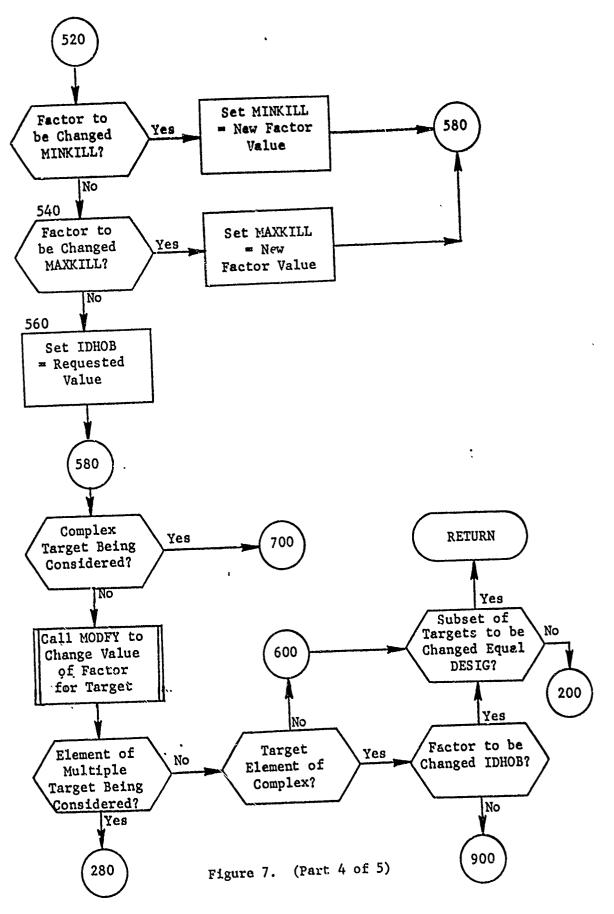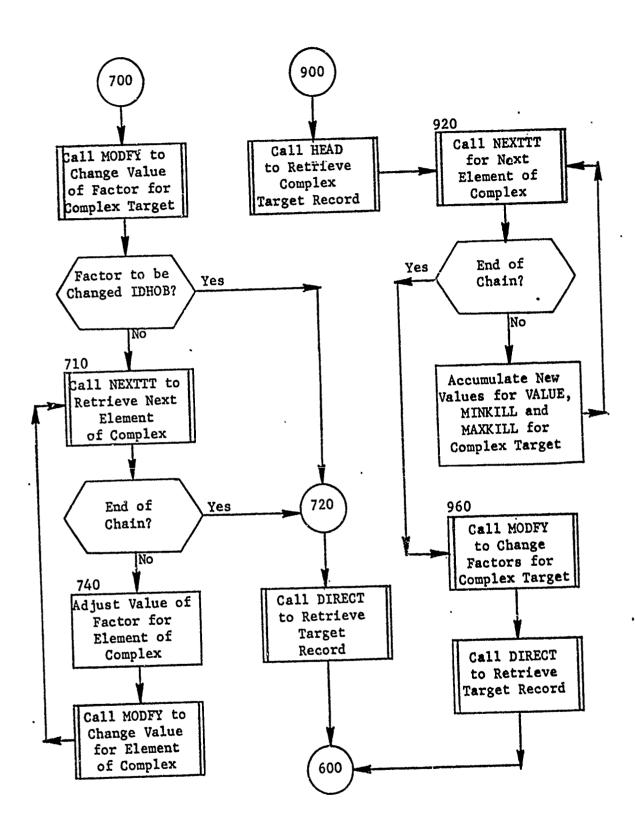
2.12 Subroutine PENROUT

PURPOSE: To compute penetration corridor data and print
                     results.

ENTRY POINTS: PENROUT

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, CRLENG, IONPRT

SUBROUTINES CALLED: DISTF, HDFND, MODFY, NEXTTT, RETRV

CALLED BY: ENTMOD (of PREPALOC)

Method:
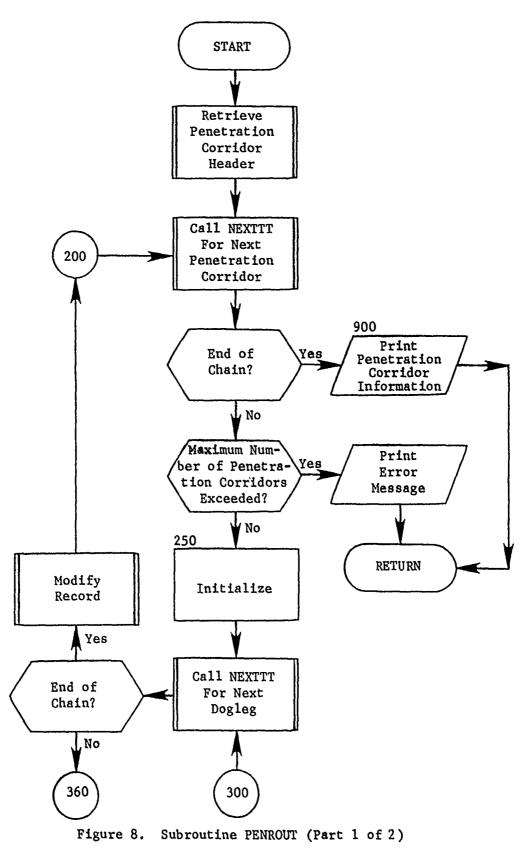
Subroutine PENROUT retrieves each penetration corridor record and cal-
culates the distance (DEFDIST) and attrition (ATTRPRE) in each precor-
ridor leg. Results are stored within record PENCRD. If user direct,
prints are produced.

Subroutine PENROUT is illustrated in figure 8.

Figure 8. Subroutine PENROUT (Part 1 of 2)
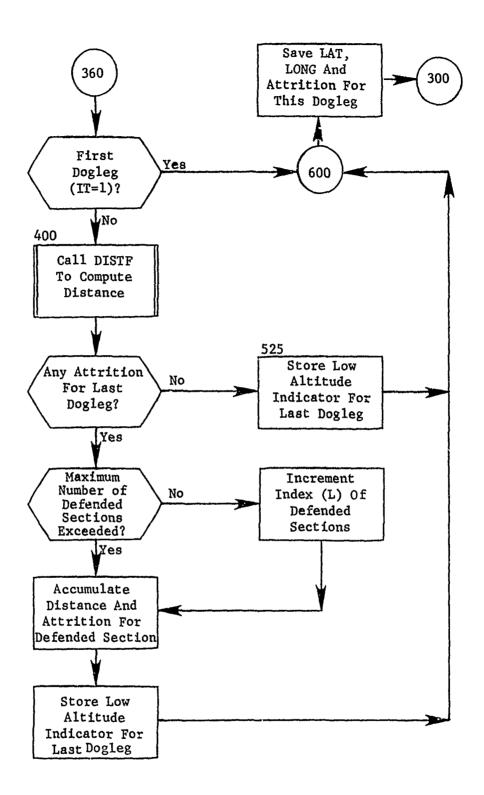
Figure 8. (Part 2 of 2)

2.13 Subroutine TGTPREP

PURPOSE:	To update target attributes and perform salvo calculations.
ENTRY POINTS:	TGTPREP
FORMAL PARAMETERS:	None
COMMON BLOCKS:	C10, C15, C30, DISTEF, IONPRT, ISIMTYPE, SUMNEW
SUBROUTINES CALLED:	DIRECT, DISTF, DLETE, HDFND, HEAD, IGET, IPUT, KEYMAKE, MODFY, NEXTTT, RETRV, STORE, TOFM
CALLED BY:	ENTMOD (of PREP)

Method:

Reference Codes of target records to be processed are contained within the LIXSTXX chain. Each target is retrieved and attributes modified. If the user requested a height-of-burst for the target being processed, it is stored accordingly. Also, each target value is renormalized in order to guarantee the sum of all the target values equal 1000.

The distance from the target through the depenetration corridor and the distance to recovery is computed and the minimum distance stored within the TDDIST record. The penetration point associated with the target is the point which minimizes the sum:
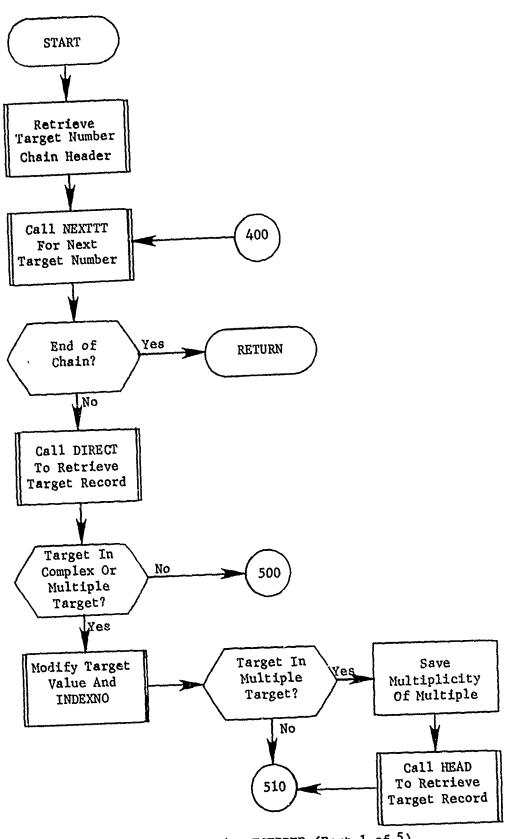
$$(2*DISTD) + DISTR$$

where DISTD is the distance from target to the depenetration point and DISTR is the distance from depenetration point to recovery.

Also the distance from each penetration corridor to the target and the corridor attrition are placed within the TPDIST record.

If there are fixed weapon requests for the target being processed, the MYASGN chain is queried for definition of specified downtimes and, if applicable, the salvo launch number is determined. Fixed assignment modified is made to the ASSIGN record.

Subroutine TGTPREP is illustrated in figure 9.
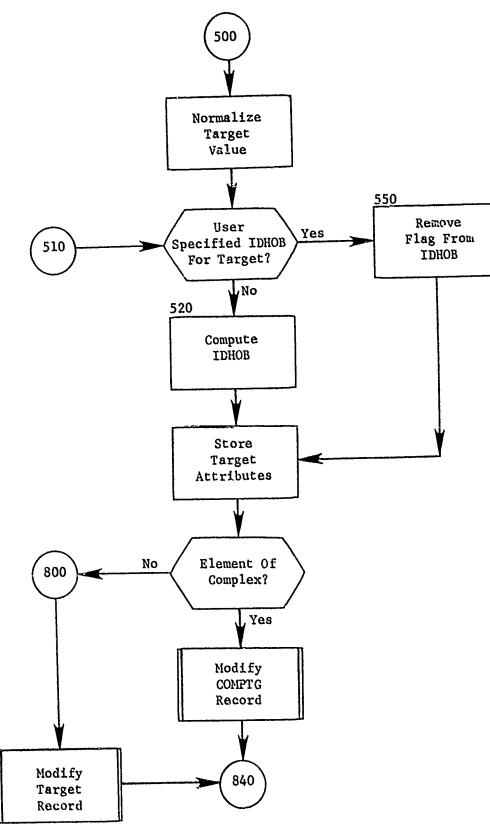
Figure 9. Subroutine TGTPREP (Part 1 of 5)
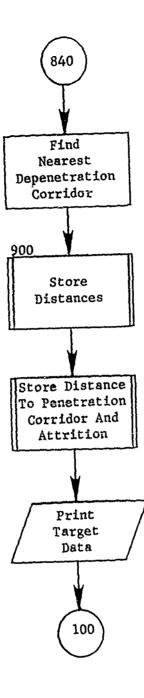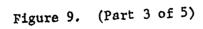
Figure 9. (Part 2 of 5)

Figure 9. (Part 3 of 5)

Figure 9. (Part 4 of 5)

Figure 9. (Part 5 of 5)
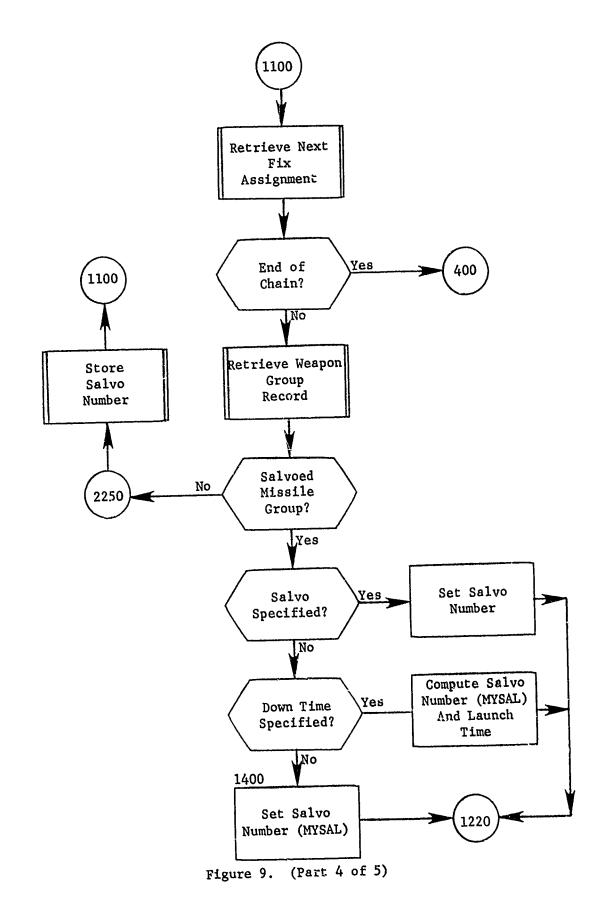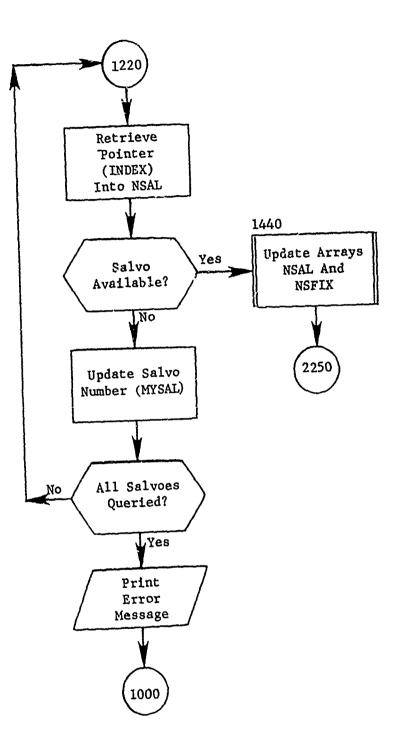
2.14 Subroutine WEPPREP

PURPOSE: To update weapon group data and print data.

ENTRY POINTS: WEPPREP

FORMAL PARAMETERS: None

FORMAL PARAMETERS: None

COMMON BLOCKS: ASMTYP, C10, C15, C30, CRLENGTH, IGPREF, IONPRT,
                      ISIMTYPE, NUMCOR, REPOINTS

SUBROUTINES CALLED: DIRECT, DISTF, HDFND, HEAD, MODFY, NEXTTT, RETRV

CALLED BY: ENTMOD (of PREP)

Method:

The number of weapons in each group is increased according to the
position specified in common /GAMEVAR/. The destruction before launch
probability is then modified by a factor to maintain the same number of
expected launched weapons as before.
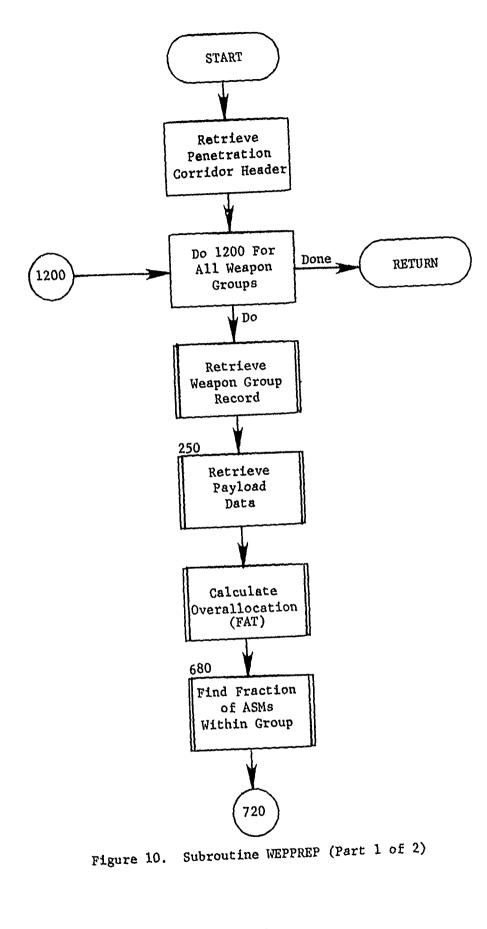
For bomber groups, the basic overallocation is reduced if there are
less than 15 bombers in the group. If so, the overallocation is multi-
plied by the ratio of the number of vehicles to 15. This has the effect
of reducing the overallocation for small bomber groups.

For bomber weapon groups that contain ASMs, array EXPASM defines the
fraction of weapons in each group that are ASMs rather than bombs.

Arrays MAXSLV and NSAL are calculated for salvoed missile groups.
MAXSLV contains the maximum salvo number for each salvoed group and
NSAL the number of weapons in each salvo. The NSAL array is packed
into three words, four bits per salvo.

Upon storing all data, weapon group information is written onto BASFIL
followed by parameters that described the percentage of weapon overallo-
cation and, finally, naval weapon group data is written.

Subroutine WEPPREP is illustrated in figure 10.

58

Figure 10. Subroutine WEPPREP (Part 1 of 2)
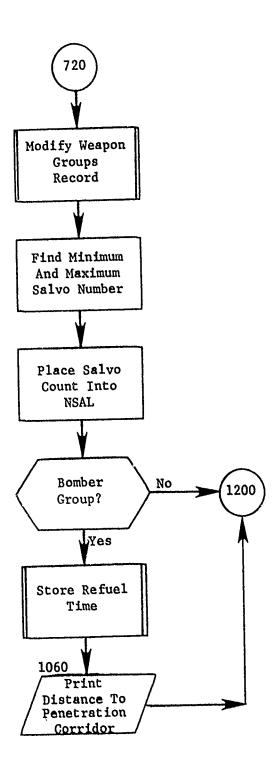
Figure 10. (Part 2 of 2)

SECTION 3. ALOC MODULE

3.1 Purpose

The major purpose of this module is to determine the optimal allocation of weapons to targets, using a Lagrange multiplier technique. The weapons are divided into weapon groups -- each group containing weapons of the same characteristics which are geographically close. Thus, except for time launch interval constraints for some "salvoed" missiles, weapons are considered identical within groups. Each target is considered individually for weapon assignment. The order of investigation is the order of the TARCDE records on the LISTXX chain which was determined by the PLANSET module. When all targets have been processed, another pass over this chain begins. This process continues until the Lagrange method has allocated all the weapons to targets. The assignments are stored as ASSIGN records in the integrated data base during the process.

The user is able to specify weapon assignments through the FIX adverb to the PREPALOC module. In this case the ALOC module will optimally assign those weapons which have not been fixed. In addition, there are capabilities which allow the user to modify weapon range values, to restrict the use of MIRV weapons by target class, and to restrict the use of any weapon group by the value of either of the target attributes FLAG or CNTRYL.

3.2 Input

The precondition of the integrated data base required is that the PREPALOC module has to have been executed. Furthermore, there is an optional input file -- the Weapon/Target Data File. The Weapon/Target Data file contains the information relating each weapon group to each target. The Weapon/ Target Data file, if not input, is created by the FRSTGD subroutine on pass one and may be retained for later executions of ALOC. One record is produced for each target whose length depends upon the number of weapon groups and the targets number of hardness components (length = number of groups x (3 + 2 x number of hardness components) + 1). This is the file which is created in pass one and may also be used in subsequent runs (see RECALC Mode: Users Manual, UM 9-77, Volume III). The format for the Weapon/Target Data file -- file code 15 -- appears in table 2.

3.3 Output

As a result of its execution the ALOC module creates ASSIGN (record type 70) records in the integrated data base. Further, the attribute NUMALOC is updated in every group to reflect the actual number of weapon allocated from that group.

3.4 Concept of Operation

In order to conserve storage, ALOC is broken up into two main overlays. The first overlay is called ALCINT. This overlay reads any user input,

Table 2. Format of Weapon/Target Data File -- File Code 15

WORD	DESCRIPTION
1	Target Number
2 -(NWEPGRP*+ 1)	Time of arrival of group or target
(NWEPGRP + 2) (2 x NWEPGRP + 1)	Corridor used by group or reason group is inactive
(2 X NWEPGRP + 2) -(3 x NWEPGRP + 1)	Penetration probability of weapon to target
(3 x NWEPGRP + 2) -(4 x NWEPGRP + 1)	Kill probability of weapon against first hardness component
(4 x NWEPGRP + 2) -(5 x NWEPGRP + 1)	Kill probability of weapon against second hardness component**
(5 x NWEPGRP + 2) -(6 x NWEPGRP + 1)	***Alternate kill probability of weapon against first hardness component
(6 x NWEPGRP + 2) -(7 x NWEPGRP + 1)	Alternate kill probability of weapon against second hardness component*

---

* Number of weapon groups

** Does not appear if target has only one hardness component

*** (4 x NWEPGRP + 2) -(5 x NWEPGRP + 1) if target has only one hardness component

including restrictions, modifications, print request and so on. Furthermore, the weapon data is extracted from the integrated data base. The second overlay (ALCMUL) controls the determination of the allocation. The driver routine of this overlay is MULCON. Within the second overlay there are four segments. The first, FGD, obtains target data for pass one. The second, SGD, obtains target data for passes two and beyond. The third segment, STAL, principally routines STALL, WAD and WADOUT, allocates to targets without terminal ballistic missile defenses. The fourth segment, DEFAL, principally routines DEFALOC and RESVAL handles ballistic missile defended targets.

3.4.1 Overlay ALCINT. The routines in this overlay are straightforward and need little explanation beyond that below. However, the routine DATGRP places information on a random access file (file code 25) which is used by the FGD segment. Table 3 shows the format and content of this file. This random access file is indexed on group number.

3.4.2 Overlay ALCMUL. The design of the weapon-to-target allocator utilizes a hierarchy of subroutines operating at different levels of detail. Figure 11 illustrates this hierarchy. The major functions associated with these subroutines are summarized below and related to the overall concept in subsequent paragraphs.

Subroutine MULCON is the first subroutine in the hierarchy and is responsible for the control and adjustment of the Lagrange multipliers. MULCON monitors the rate at which various classes and types of weapons are being allocated to the target system and makes appropriate adjustments in the values of the Lagrange multipliers. In this role, MULCON does not need any detailed information concerning actual allocation. It is concerned only with the actual rate of allocation of the available inventory as the targets are processed. To obtain the assignment of weapons to each successive target, MULCON simply calls subroutine STALL (Single Target Allocator) for targets without missile defenses, or subroutines STALL and DEFALOC if the target is defended. STALL and DEFALOC utilize the current values of the multipliers to make an allocation to the next target, then return control to MULCON.

The data acquisition for the allocation process is performed by the FRSTGD routine on pass one and SCNDGD on all other passes. Each of these routines brings in the proper IDS records for the next target and prepares the weapon data for that particular target. The Weapon/ Target Data File is read (or on the first pass in the RECALC mode calculated). On the first pass, FRSTGD then creates a record on file code 21.which, principally, contains the INACTIVE array (see table 4). If the user has requested range modification, FRSTGD may also write a record onto file code 22 in the same format as file code 15 (table 2). This new file serves as a source for replacement record for file code 15. SCNDGD reads each of these files in order to obtain the appropriate information.

63

Table 3. Random Access File from DATGRP

WORD	DESCRIPTION
1	Group type index number
2-10	Logical flag restrictions
11-160	Country location restrictions (logical switches matching countries in Block CNCLS)
161	Switch -- true if group is a restricted MIRV
162-185	Logical MIRV restriction switches
186	Range multiplier
187	Refueled range multiplier
188	Minimum range replacement value
189	NALTDLY for group
190	ALTDLY for group
191	GLAT for group
192	GLONG for group
193	GREFCODE for group
194	GYIELD for group
195	RANGE for group
196	CEP for group
197	SPEED for group
198	RANGED for group
199	RANGER for group
200	RNGMIN for group
201	GREFTIME for group
202	TOFMIN for group
203	CMISS for group
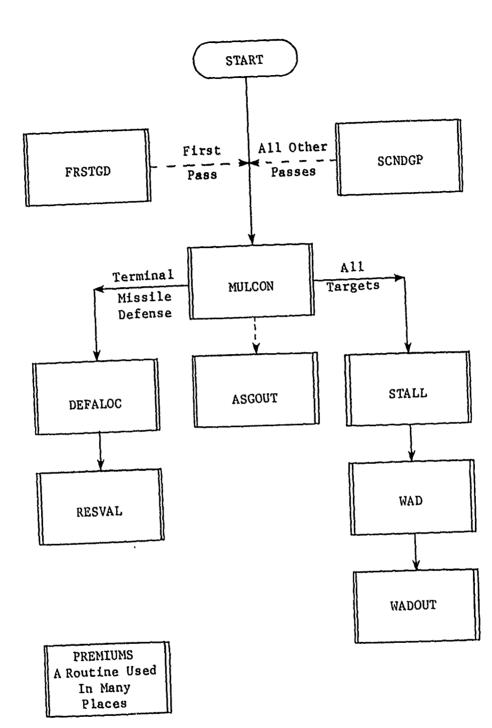
Figure 11. ALCMUL Calling Sequence Hierarchy

Table 4. INACTIVE Array File (File Code 21)

WORD	DESCRIPTION
1	Target number
2	Length of file 15/22 record
3	Logical switch -- if true, file 22 contains alternate data record
4-253	INACTIVE array

The ASGOUT subroutine is called by MULCON to insert the allocation to a particular target in the integrated data base. ASGOUT makes sure that the allocation indicated for the target is that which has just been completed.

Subroutine STALL is the next subroutine in the hierarchy when dealing with targets without missile defenses. STALL utilizes the values of the multiplier supplied by MULCON and generates an appropriate allocation of the weapons to be specified as single target. It is not responsible for computing payoffs and is not responsible for actually adding or deleting weapons. When STALL has determined that a single weapon should be added, or deleted, it calls the Weapon Addition and Deletion subroutine WAD. WAD then adds or deletes any weapon as specified and corrects the residual target value. In addition, before returning to STALL, WAD examines every other relevant weapon group and calculates the potential change in payoff, if a weapon from that group was added or deleted. This information on potential payoffs is used by STALL in determining whether other weapons are to be actually added or deleted. When STALL has achieved an allocation of weapons to the target appropriate for the current values of the multipliers as supplied by MULCON, it returns control to MULCON.

While the allocations generated by STALL are determined by the values of the Lagrange multipliers and the target payoff functions, there is no requirement for STALL to be involved in the calculation of these quantities. Thus, the structure of STALL can be independent of the details of the operation of either WAD or MULCON.

Subroutine WAD (weapon addition and deletion) is the next subroutine in the hierarchy and is responsible for the mechanics of addition and deletion of weapons and for the actual calculation of payoff for targets without missile defenses.

Subroutine WADOUT is called by WAD to summarize the output of WAD for STALL. WADOUT calculates an overall benefit for using each weapon by adding the current premium for using weapons from that group to the potential payoffs computed by WAD. These benefits are then compared with the current prices (or Lagrange multipliers) to produce the summary data actually used by STALL. Thus STALL, in fact, is attempting to maximize (PAYOFF + PREMIUM - COST) rather than just (PAYOFF - COST). The introduction of the premium provides a flexibility which is used to accelerate convergence to an allocation that exactly matches the stockpile.

Subroutine DEFALOC performs the same function as Stall for targets with terminal ballistic missile defense. The complication that necessitates a separate subroutine is the nonconcave payoff function for defended targets. DEFALOC determines whether it is more profitable to attack the target with missiles until the interceptors are exhausted than to use the STALL allocation. If it is not profitable to exhaust the defense, then the allocation job is turned over to STALL, after setting the missile penetration parameters to reflect leakage through the defense. In the

event an exhaustion tactic is most profitable, DEFALOC calls subroutine RESVAL t> calculate the payoff against the defended target with a specified mix of weapons.

Subroutine RESVAL calculates the damage to a ballistic missile-defended target when attacked with a specified mix of weapons. The attacking missile payloads may contain decoys and electronic penetration aids which degrade interceptor effectiveness. The target is defended with a prespecified nominal number of terminal interceptor salvos with kill probability PKTX against unhardened warheads. Uncertainties may be introduced into the number of interceptors by specifying two probabilities PK(1) and PK(2) that the actual number of interceptors will be RX(1) lower or RX(2) higher than the nominal number.

All of those factors discussed under WAD are considered in RESVAL except correlations in weapon failures.

Subroutine PREMIUMS is the final subroutine in the hierarchy. It is called to calculate the current premiums for adding or deleting any weapon. The premium reflects whether the weapon is currently overallocated or underallocated. The size of the premium and the way it is calculated change as the allocation progresses.
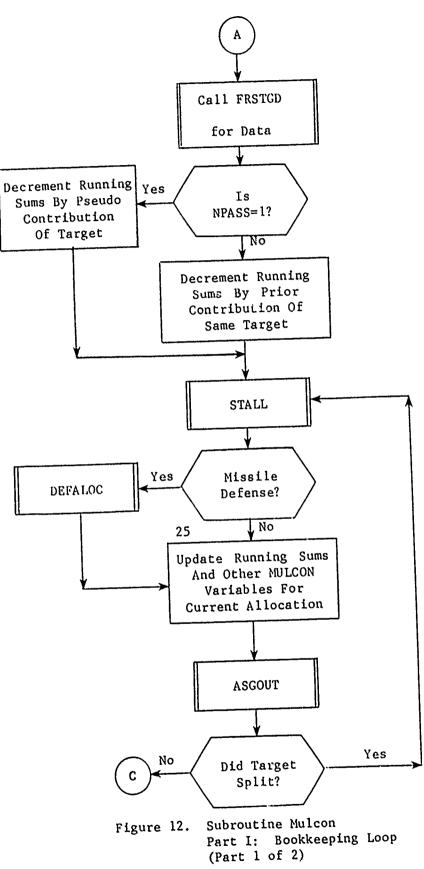
The remainder of this section treats the subroutines one at a time.

3.4.2.1 _Subroutine MULCON_. The flow of operations in MULCON is illustrated in figure 12. The diagram is broken into two parts. Part I is the main bookkeeping loop. Part II is the computational loop.

The loop shown at the beginning of part I of figure 12 has two branches. The left branch is used only on the first pass over the target system. All succeeding passes use the downward branch. On the first pass, the raw data on target characteristics for each target are read in by subroutine FRSTGP. Then the basic information on capabilities of each weapon with respect to the target is computed. Since these data are independent of the allocation to the target, they are stored on files to avoid recomputation of the data on later passes. Just before weapons are actually allocated to the target, the allocation previously recorded for the target (in the initial pseudoallocation) is removed by deleting it from the running sum used to estimate allocation rate. That is, the contribution of the target i to $\sum_i N(i,J) * W(J)$, known as RUMSUM, * and

W(i), known as WTSUM, is removed.

In block 25, after STALL and/or DEFALOC has been called for an allocation to the target, the running sums are augmented by the new contribution of the target i using the new (and usually much larger) target weight W(i). Subroutine BOMPRM is called to update the ASM allocation fraction for the bomber groups. Finally, subroutine ASGOUT updates the integrated data file with the new allocation.
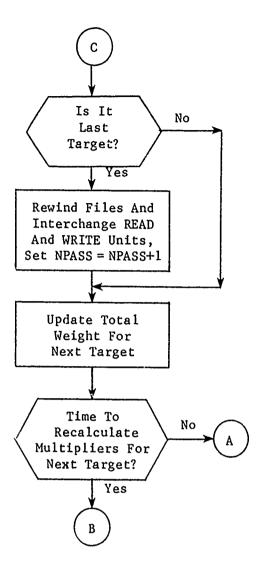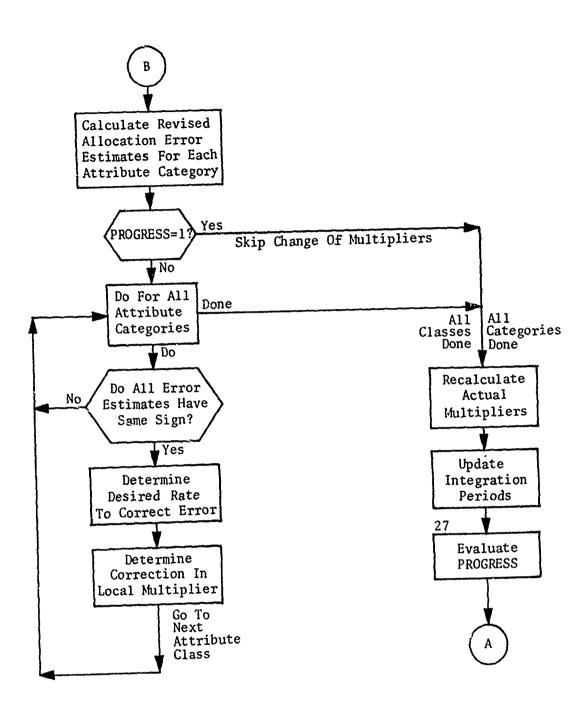
68

Figure 12.   Subroutine Mulcon
Part I: Bookkeeping Loop
(Part 1 of 2)

Figure 12. (Part 2 of 2)

Figure 12. Part II: Computation Loop

Again, after the new allocation has been made, it is added into the running sums. Of course, multiple targets (single target records on the file which represent several identical targets at slightly different locations) are subtracted from and added to the sums as multiple targets. During the c osing phase, these targets may be separated to allow different allocations to the separate targets. Provision is made at the end of ASGOUT t. recycle and process later elements of the same multiple target if such a split occurs during the allocation to the target. Before passing on to the next target, the current value of the target weight is revised.

After every two to four targets, the Lagrange multipliers are updated by transferring control to the multiplier computation loop shown on figure 12. The error in the rate of allocation for each collection of weapons J is estimated. Three separate estimates are made corresponding to differing rates of increase of the target weights. If all estimates have the same sign, then a small adjustment of the multiplier in the indicated direction is made.

The revised local multipliers are then used to recalculate the Lagrange multipliers. During the closing phase (PROGRESS = 1), the local multipliers are not changed so the actual multipliers remain unchanged. The rate of change of the target weight is adjusted depending on the apparent size of the current error in the allocation rates.

Finally the progress of the allocation is evaluated and flags are set if the mode of operation is to change.

3.4.2.2 _Subroutine STALL_. STALL is basically a very simple routine. It is not responsible for computing payoffs. The payoffs are computed by WAD and summarized for STALL by WADOUT. Thus, the input to STALL consists only of the summary data provided by WADOUT. These data consist of only the following variables:

a. PPMX the maximum potential profit available if a weapon is added, and TPPMX the index to that weapon

b. PVRMX the maximum efficiency for any potential weapon that could be added, and IPVRMX the index to that weapon

c. DPMN the minimum differential profit produced by any weapon on the target, and IDPMN the index to that weapon.

Actually, of course, the profits and efficiencies mentioned above are based on a modified payoff (or BENEFIT) computed by WADOUT, which includes the actual payoff from WAD together with the premiums for staying close to the desired allocation rates. Thus, throughout the allocation, the operation of STALL remains the same; it simply tries to maximize this modified payoff. Changes in the mode of the allocation are thus accomplished in WADOUT simply by changing the way the payoffs are modified, so no change in the logic of STALL is required.

To obtain the initial values of these quantities, STALL makes an initialization call on WAD. Then it adds the weapons which have been fixed to the target by the user. STALL also calls subroutine INITSAL. This routine initializes the Lagrange multipliers and preferred salvo indicator for the salvoed missiles.

On the basis of the values delivered by WADOUT, STALL decides whether to add a weapon. After each call on WAD to add or delete a weapon, a new set of variables is delivered by WADOUT, and STALL uses this revised information to decide whether more weapons should be added or deleted and finally when to terminate the allocation.

Figure 13 illustrates the operation of STALL. As part I of the flow chart shows, a special option has been provided so that, in the case where IVERIFY = 2 and PROGRESS = 2, the normal operation is short circuited and STALL simply duplicates the previous allocation to the target so that with one additional pass the allocation can be evaluated with different correlation coefficients.

In all other cases, the normal allocation procedure is used. This procedure consists of four parts:

I. Set up and single weapon allocation phase

II. Fixed weapon processing

III. Multiple weapon laydown loop

IV. Multiple weapon refinement loop.

The most time-consuming part of subroutine STALL is the multiple weapon refinement loop. This phase tests many permutations of weapon assignments which could be made. The full testing of every single target allocation can considerably slow down the operation of the allocator. Therefore, a way of terminating the testing has been provided, through the user-input parameter QUALITY. The maximum number of weapons which will be removed in any testing process is not permitted to exceed NUM * QUALITY where NUM is the number of weapons allocated to the target. There QUALITY is a measure of the fraction of the weapons which can be removed. If QUALITY is set to 0, the refinement operation is skipped e; rely.

Since the program provides only finite arrays to list the number of weapons assigned to a target and for WAD to compute surviving target value for different times of arrival, there is always the possibility that the arrays provided might be exceeded. Therefore, before any weapon is added, a test is made to be sure it will not overflow these arrays.

If the maximum number of weapons would be exceeded, then the least profitable individual weapons are removed to make way for the most profitable weapons. If the number of time-of-arrival columns would be exceeded,

73
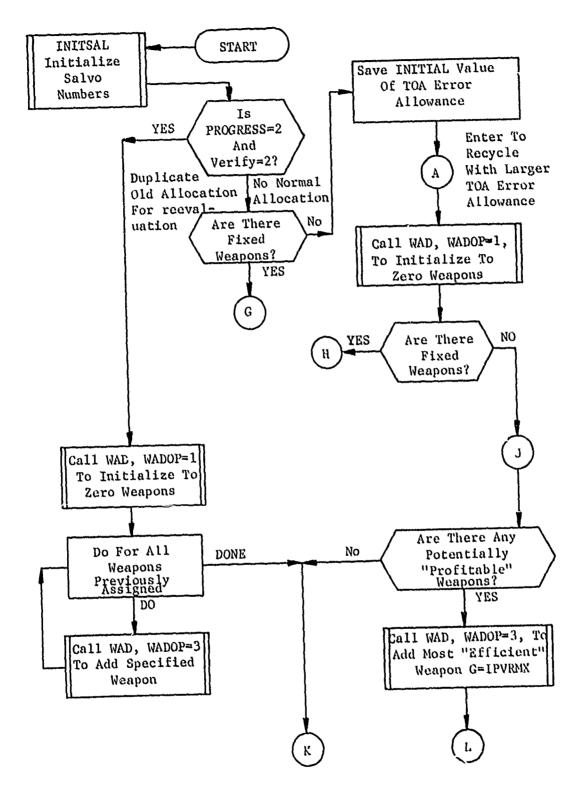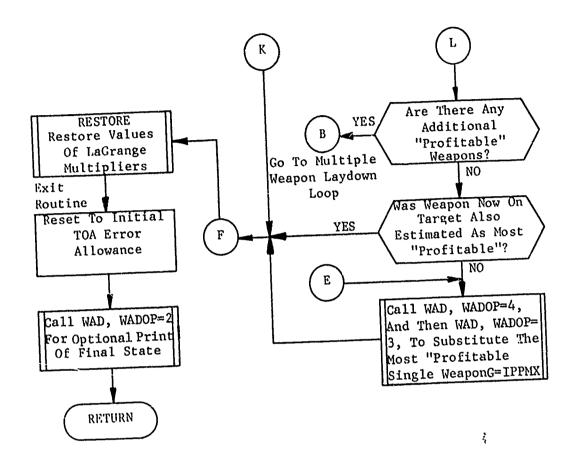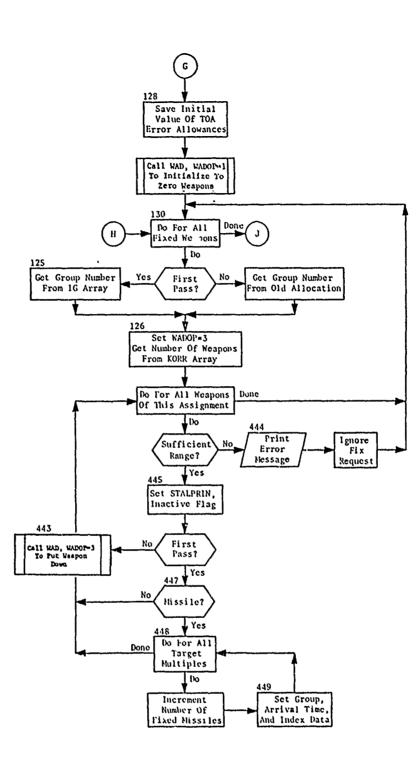
Figure 13. Subroutine STALL
Part I: Setup and First Weapon
(Part 1 of 2)

74

Figure 13. Part I
(Part 2 of 2)
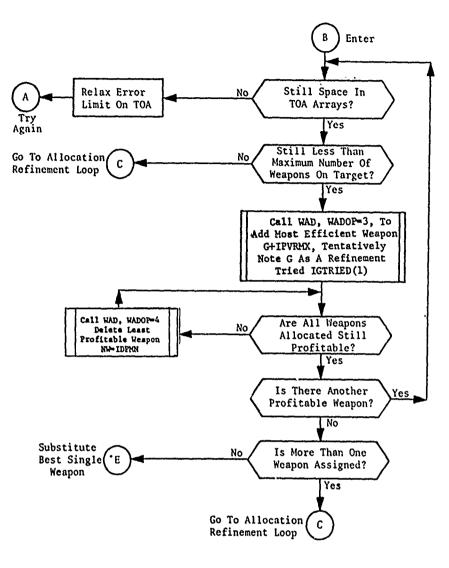
Figure 13.   Part II: Fixed Weapon
Assignment Processing

76

Figure 13. Part III: Multiple
Weapon Laydown

46 Set Maximum Number Of Weapons To Remove; Set Number Of Weapons Tested (KOUNT=0); And Set Verification Pointer IPOINT

C

49 Store As Only Verified Weapon IGTRIED(1)=G

47 Initialize To Only One Weapon Verified And Stored

52 Delete Least Profitable Weapon NW=IPPMN

57 Store As A Verified Weapon And Increment Number of Weapons Verified And Stored (NTRIED And NSTORE)

51 Reset Number Of Weapons Verified And Stored To Zero

D

48 All Weapons Now Allocated Profitable? — No → 51; Yes ↓

Is This The Same Weapon As That Just Deleted In Verification Effort? — Yes → 57; No →

Select Best Single Weapon

58 Is Only One Weapon Now Allocated? — Yes → **E**; No ↓

54 Any Other Potentially Profitable Weapons? — No → 58; Yes ↓

56 Call WAD, WADOP=3 Add Most Profitable Weapon G=IPPMX

65 Increment Total Weapons Tested (KOUNT)=KOUNT+1

55 Still Space In TOA Arrays? — Yes → 59; No ↓

59 Still Less Than Maximum Number Of Weapons On Target? — Yes → 56; No ↓

66 Is KOUNT·GT·QUALITY*NUM Or Have All Weapons Been Verified? — Yes → **F**; No ↓

85 Relax TOA Error Limit

Have All Weapons Been Removed Once? — Yes → **F**; No ↓

Increment Pointer, IPOINT To Next Group To Verify; If End Of List, Recycle

A Start Over

Would Weapon Just Added Be Removed Again? — Yes → **F**; No ↓

D

Has A Weapon From This Same Group Already Been Verified? — No → Call WAD, WADOP=4 To Delete Weapon To Be Verified; Yes ↓

Substitute Most Profitable Weapons, Regardless Of Efficiency

Increment Number Of Weapons Verified NTRIED

Figure 13. Part IV: Allocation Refinement Loop

78

the error criterion for treating slightly different times of arrival in the same column is relaxed, but the allocation to this target must be reinitialized and repeated.

3.4.2.3 Subroutine WAD: When the next target is to be processed, WAD is called with the control variable WADOP set to 1. This results in the initialization of the allocation for that target, starting with zero weapons assigned. STALL examines these data and determines what to do. On all succeeding calls, the print options 9 and 22 are available to print the decision made by STALL, together with the data (previously given by WAD to WADOUT) on which the decision was based. If STALL decides to add a weapon, WAD is called with WADOP = 3 and with the variable G specifying the group from which the weapon is to be added. If STALL later decides to delete a weapon, WADOP is set to 4, and the variable NW is set to specify which weapon in the list of those assigned (the NWth weapon in the list) is to be deleted. Finally, when STALL decides the allocation is complete, a dummy call WADOP = 2 is made to permit a print of the data which caused STALL to terminate the allocation. The option WADOP = 5 should never be used. It is provided simply to catch any erroneous calls (WADOP > 4) on WAD.

Each of the main options (WADOP = 1,3,4) causes control to pass to an internal control routine, which in turn makes calls on appropriate local subroutines in WAD. The structure of these routines is governed strongly by the objective of speed and efficiency, and need not be dealt with here.

3.4.2.4 Subroutine WADOUT: Much of the logic of the subroutine is concerned with the decisions of which weapon groups to make INACTIVE to save time in the computations in WAD. The variable INACTIVE outside of WADOUT is interpreted as true so long as it is not 0. That is, only weapon groups for which INACTIVE = 0 are processed. If INACTIVE = 100, it implies that the range is inadequate, and the weapon will be inapplicable regardless of its Lagrange multiplier. This value of INACTIVE is set permanently where the weapon-target interactions are computed in subroutine GETDTA. Inactive = 2000 or 30000 is used to note weapons that are inapplicable because their cost (LAMEF) is too high for the target. These must be reconsidered each time a target is processed. The flow of the program is illustrated in figure 14.

Basically, the subroutine begins with a do-loop over all weapons currently assigned, to compute the marginal BENEFIT associated with these weapons. At the same time, it tags all the weapons assigned by setting INACTIVE to -100 to avoid any possibility that any weapons currently assigned will be made inactive. It then enters a do-loop over all potential weapons to evaluate the marginal potential BENEFIT associated with these. It skips any weapons already set inactive (INACTIVE = 100,2000,30000) and processes the remainder (INACTIVE = -100,0). It also skips salvoed missiles if there is no available salvo because of

79
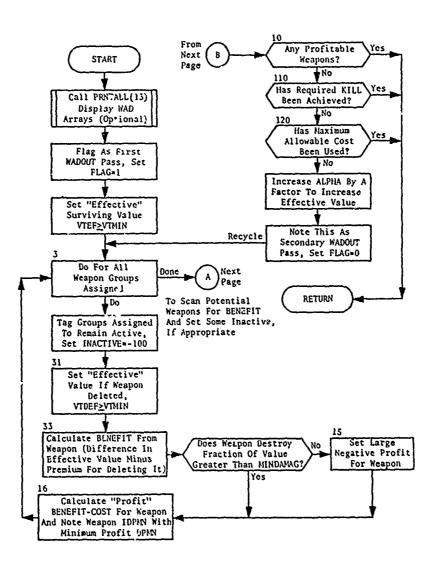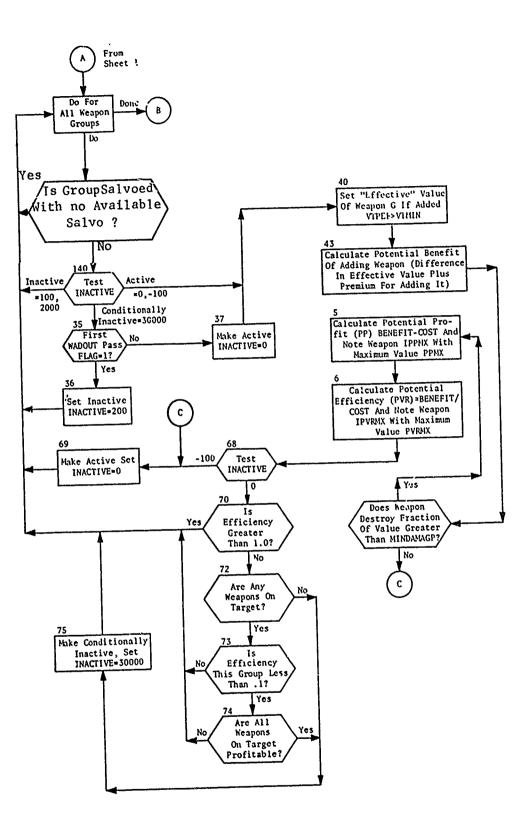
Figure 14.   Subroutine WADOUT (Part 1 of 2)

Figure 14. (Part 2 of 2)

salvo overallocations in all salvos. Those not currently assigned to the target (INACTIVE ≠ - 100), and not showing potential profitability are considered to determine whether they should be set inactive.

If a weapon is not potentially profitable when no weapons are assigned to the target, it is presumed safe to set it inactive (INACTIVE = 30000). If there are other weapons on the target that are unprofitable, the decision to make a weapon inactive is postponed until these are removed. Otherwise, any weapon whose cost is a factor of 10 higher than its potential payoff is made INACTIVE -- it being presumed that even with replacement (or substitution) of weapons by STALL such a weapon is very unlikely to become attractive. The value of INACTIVE, however, is set to 30000 (conditionally INACTIVE) rather than 2000. Ordinarily this is treated exactly the same as INACTIVE = 2000. However, if before exiting from WADOUT it is found necessary to recycle (using a revised value of ALPHA in order to achieve a specified MINKILL), then a flag, REEVAL, is set and those weapons with INACTIVE = 30000 are reconsidered. In some cases, they may turn out to still be applicable because of the increase in effective target value. However, if an exit from WADOUT occurs with INACTIVE ≠ 0, it must never be set back to 0 during the remainder of this allocation to the target. Otherwise, incorrect computation would occur -- as a result of trying to do various steps in the computation in WAD despite having omitted earlier steps when the weapon was treated as inactive.

Therefore, any weapon group encountered by WADOUT with INACTIVE = 30000 is immediately set to INACTIVE = 2000 unless it is a case of recycling with a new value of ALPHA.

WADOUT, however, has two other subsidiary responsibilities. It is responsible for modifying, when necessary, the six variables transmitted to STALL, so that STALL will not try to exceed the MAXKILL specified for a target, and so that it will continue to add weapons until any specified MINKILL is achieved.

To make it possible for QUICK to match target kill requirements specified for simpler models, where both correlations and time dependence are ignored, a user-input parameter IMATCH is provided in which MINKILL and MAXKILL are interpreted in terms of an oversimplified target kill estimate VTZO.

If the parameter IMATCH is set nonzero by the user, the internal parameters VTMIN and VTMAX are modified. These parameters are defined as:

$$VTMIN = \text{Original target value} * (1 - MAXKILL)$$

$$VTMAX = \text{Original target value} * (1 - MINKILL)$$

Their default values are VTMIN = 0, VTMAX = original value.

If the IMATCH parameter is used, VTMIN and VTMAX retain their default values until the 0th order calculation VTZO indicates that MINKILL or MAXKILL have been reached. Then VTMIN or VTMAX is set correspondingly and thereafter operates as usual.

In any case, WADOUT modifies its calculations so that every weapon placed on target destroys at least that percentage of original target value specified by the user-input parameter MINDAMAG.

3.4.2.5 <u>Subroutine PREMIUMS</u>. The purpose of this subroutine is to compute the payoff premium required by STALL to avoid unduly large deviations from the desired allocation rate. During the closing phase, the premium is also used to cause STALL to close in to an allocation that exactly meets the stockpile constraints. PREMIUMS is called with one parameter which specifies for which group a new evaluation of the premiums (PREMIUM(G) and DPREMIUM(G)) is desired. The call results in the replacement of the old value of these premiums by a new value.

3.5 <u>Common Block Definitions</u>

The common blocks internal to the ALOC module are shown in table 5.

Table 5. ALOC Module Common Blocks
(Part 1 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
ALERUN	ALERREST(380,3)	Estimate of allocation errors
	RUNSUM(380,3)	Running sum of target weight times weapons allocated
C33	NBLN	Number of ballistic interceptors
	TMULT	Target multiplicity (0 if multiple target becomes split)
	VT	Target value remaining
	TGTWT(3)	Weighting values
	PAYOFF	Target value destroyed
	COST	Sum of weapon values allocated
	PROFIT	PAYOFF - COST
	DPROFIT	Change in PROFIT from last pass
	WRTEST	Test parameter
CNCLS	CNTRY(150)	List of country codes
	CLSS(20)	List of target classes
	NCNT	Number of country codes
CONTRO	WADOP	WAD option
	PROGRESS	Measure of allocation progress
	NPASS	Allocation pass
CORSTF	COSCOR(30)	Cosine of corridors orientation point latitude
	DPLAT(30)	Difference between latitude of orientation and origin
	DPLONG(30)	Difference between longitude of orientation and origin
	CORLN(30)	Distance from orientation to origin
	TDEFDIST(30)	Total precorridor attrition
	ATTROCOZ(30)	Corridor attrition rate
	ATTRSUZ(30)	Corridor suppression rate
	HILOATZ(30)	Ratio of high to low attrition
	CRLENGTH(30)	Corridor length

84

Table 5. (Part 2 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
CORSTF (cont.)	ORGLAT(30)	Origin latitude
	ORGLONG(30)	Origin longitude
	DISTCDZ(30)	Distance from origin to target
	ATTRAD(30)	Sum of precorridor and corridor attrition
	CROSSDST(30)	Perpendicular distance from axis to target
	ENTLAT(30)	Entry latitude
	ENTLONG(30)	Entry longitude
	DISTDG	Distance from target to recovery base
	MAXCOR	Number of corridors
CURSUM	CSALL	Number of targets assigned to all weapons
	CSREG(5)	Number of targets assigned to weapons from a given region
	CSCLAS(2)	Number of targets assigned to weapons from a given class
	CSTYPE(120)	Number of targets assigned to weapons of a given type
	CSGRP(250)	Number of targets assigned to weapons from a given type
	CSOTH(2)	Number of targets assigned to weapons with a given alert status
DEFCOM	RATM	Highest return rate from DEFALOC
	ISALFX(250)	Storage for fixed salvo numbers
	NSL(250)	Number of salvoed weapons available
	RATE(250)	Rate of return for missile on defended target
DEFRES	NOWEP(250)	Number of weapons assigned by DEFALOC
	VTDX	Surviving target value
	NTX(3)	Terminal defender estimates
	PX(3)	Probability of NTX

Table 5. (Part 3 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
DYNAMIC		Contains allocation
	IG(30)	Group assigned
	KORRX(30)	Corridor assigned
	RVALX(30)	Relative value of assignment
	PENX(30)	Penetration probability
	TOARR(30)	Time of arrival
	ISAL(30)	Salvo number
	NUMFIX	Number of fixed assignments
	NUM	Number of assignments
FIL21	JTG	Target number
	ILENTH	Length of record on file 15 (or 22)
	I22SW	Indicates need to read file 22
	ICTIVE(250)	Storage for INACTIVE array
FIRST	FIRST	Indicates first target of pass
	END	Indicates end of target list
	F22LSW	Indicates file 22 in use
FLGSTF	LFLAG(63)	Packed logical flag restrictions
	LCNTRY(1042)	Packed logical location restrictions
	NMMRV	Number of restricted MIRV groups
	MRNAM(100)	Payload table name of restricted MIRV
	LCLAS(67)	Packed logical MIRV class restrictions
	RNMUL(250)	Range multiplier
	RNRMUL(250)	Refueled range multiplier
	RNMIN(250)	Range minimum replacement
FORMTT	INWORD	Value which needs a format
	NFORMAT	Format for INWORD
GRPHDR	IWGHDR	IDS Reference Code for weapon group header

86

Table 5. (Part 4 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
GRPSTF		Contains record from file 25
	ITYPE	Group's type index
	LFLAG(9)	Flag restrictions
	LCNTRY(150)	Location restrictions
	LMRSW	Indicate if group is restricted MIRV
	LCLASS(24)	MIRV restrictions
	RMUL	range multiplier
	RRMUL	Refueled range multiplier
	RMIN	Minimum range replacement
	GPARAM(15)	Contains the following group parameters in order: NALTDLY, ALTDLY, GLAT, GLONG, GREFCODE, GYIELD, RANGE, CEP, SPEED, RANGED, RANGER, RNGMIN, GREFTIME, TOFMIN, CMISS
INITSW	RECALC	Indicates RECALC mode
	PUNSW	Indicates output of final lambda's is desired
	FLAGSW	Indicates flag restrictions
	LOCRSW	Indicates location restrictions
	RMODSW	Indicates range modifications
	MRVRSW	Indicates MIRV restrictions
	PUNIT	Logical unit on which final lambdas are to be output
LACB	LALL	Lambda for all weapons
	LAREG(5)	Lambda for a given region
	LACLAS(2)	Lambda for a given class
	LATYPE(120)	Lambda for a given weapon type
	LAGRP(250)	Lambda for a given group
	LAOTH(2)	Lambda for a given alert status
MULTIP	CTMULT	Current target multiplicity
	NSPLITS	Number of splits in current multiple target

87

Table 5.   (Part 5 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
MULTIP (cont.)	ISPLIT	Index of current split
	NSPREC	Index of file 25 record used for this target
	KLMULT	Split range indicator
NALLY	NALL(250)	Number from group allocated on this pass
	RNALL(250)	Number from group currently allocated
NOWPS	NOALL	Number of weapons total
	NOREG(5)	Number of weapons in a given region
	NOCLAS(2)	Number of weapons in a given class
	NOTYPE(120)	Number of weapons of a given type
	NOGRP(250)	Number of weapons in a given group
	NOOTH(2)	Number of weapons with a given alert status
PAYOFF	OPROFIT	Profit from old allocation
	SPAYOFF	Sum of all payoffs
	SUMCOST	Sum of all costs
	SPROFIT	Sum of all profits
PAYSAV	GSCC(100)	CCREL for payload
	GSREL(100)	REL for payload
	GSEASM(100)	EXPASM for payload
	GSLINT(100)	LCHINT for payload
	NGSWHD(100)	NWHDS for payload
	NGSDEC(100)	DECOYS for payload
	GPAYALT(100)	PAYALT for payload
	GYLDASM(100)	ASM yield for payload
	IWHOB(100)	Payload height of burst 0 = for ground 1 = for air 2 = if not preset

Table 5. (Part 6 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
PNAV	GSPKNAV(100)	PKNAV for payload
PREMS	PREMIUM(250)	Premium for using weapon
	DPREMIUM(250)	Premium for deleting weapon
	SUMPREM	Sum of premiums
	TBENEFIT	Total benefit
PRNTCN	IDO(40)	Print request activation switch
	INDEXPR(40)	Print request selection number
	JPASS(40)	Print request first pass
	JTGTP(40)	Print request first target
	LPASS(40)	Print request last pass
	LTGT(40)	Print request last target
	KTGTFREQ(40)	Target print frequency
	ICOUNT(40)	Print request frequency counter
	MAXREQ	Maximum number of requests
	MPRNT	Number of array entries
	NREQ	Number of requests
PRTMUL	PROCMULT	Current fraction of multiple target class
	DELTEFF	Increase in profit/VALWPNS
	SDELTEFF	Sum of DELTEFF
	VALWPNS	Sum of all weapon values (lambdas)
	VALERR	Value of surplus plus deficit weapons
REFPNT	RFLAT(10)	Refuel point latitude
	RFLONG(10)	Refuel point longitude
SALVO	NSALW	Number of salvoed weapon groups
	MXSAL(75)	Maximum salvo number per weapon group
	NSALAL(450)	Running sums of salvo allocation (six words per salvoed group — packed four sums per word)
	LXIHAVE(50)	Packed logical switch indicating salvo with weapons

Table 5.   (Part 7 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
SALVO (cont.)	SAVLAM(250)*	Storage for salvoed weapon lambdas (contains average payload difference for bombers - AVDE)
	MYSAL(250)*	Available salvo (contains bomb/ASM selection for bombers - ISETPAY)
	P(250)*	Balance parameter (contains current utilization of ASMs for bomber - FASM)
SPLITS	ZZ(203)	Buffer for file 25
	SPLTMD	Switch to indicate data modified since last read
	NBLNX	Number of splits
	TMX	Not used
	INDEX	File 25 index
	STARG(3)	Starting target numbers of split
	IOFF	Offset of data in ZZ
	NTOTGT	Number of targets - all splits
	SPDAT	Not used
SMATAD	SMNOMIRV(3)	SMAT parameters for non-MIRVs
	SMATMIRV(3)	SMAT parameters for MIRVs
SURPW	SURPWP(250)	Estimated weapon surplus
TABLE	TABLE(101)	Table of square root law K-factors
TGTSAV	TGTLAT	Target latitude
	TGTLONG	Target longitude
	TGTCLS	Target class name
	VO(2)	Target value per hardness component
WADFIN	VTP(250)	Value remaining at target after weapon added
	DELVT(30)	Difference in surviving value
	NUMO	Numbers of old allocations
	IGO(30)	Group numbers of old allocation

* For bomber groups these arrays are equivalenced to arrays AVDE, ISETPAY and FASM.

Table 5. (Part 8 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
WADFIN (cont.)	IOP	Number of adds and deletes on this target
	IOPS	Sum of IOP
WADLOC	NWP(10)	Number of weapons in TOA set
	VALQ(10)	Unattritioned value at TOA
	MU(10,2)	Sum of means through TOA set
	SIG(10,2)	Sum of variance through TOA set
	V(11,2)	Unattritioned component value
	S(10,2)	Component survival probability through TOA set
	VS(10,2)	$=(V(N,JH)-V(N+1,JH) * S(N,JH)$
	VSN(11,2)	$= VSN(N-1,JH) + VS(N-\text{`} JH)$
	ITOA(250)	TOA index for group
	IADDTOA(250)	1 if new TOA set required
	SIGP(250,10,2)	Increase in variance for TOA set if weapon added
	DSIG(250,2)	Temporary contribution of weapon
	SIGD(30,10,2)	Change in variance if weapon deleted
WADOTX	DVRMX	Maximum efficiency
	IPVRMX	Index of weapon achieving PVRMX
	PPMX	Maximum profit
	IPPMX	Index of weapon achieving PPMX
	DPMN	Minimum profit
	IDPMN	Index of weapon achieving DPMN
	NUMMAX	Maximum number of weapon allowed per target
	NW	Number on target
	TPMX	Largest potential profit
	NTOA	Number of TOA sets
	NOTAMAX	Maximum TOA sets
	VTMIN	Lower target destruction minimum

Table 5. (Part 9 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
WADOTX (cont.)	VTMAX	Maximum acceptable surviving target value
	ALPHA	Factor on value required to justify VTMAX
	VTEF	Maximum of target value remaining and VTMIN
	VTZO	Total surviving target value
	VTO	First target component value
	STALPRIN	Stall print code
	G	Group Number
	N	Allocation number
WADWPN	INACTIVE(250)	Active group switch (0 = Active)
	TOA(250)	Time of arrival on target
	TVALTOA(250)	Value of target at arrival time
	VTOA(250,2)	Value per component at TOA
	MUP(250,2)	Contribution of weapon to mean if added
	RISK(6,250,2)	Relative risk of weapon interaction
	SSIG(250,2)	Square root of ln of SSKP
	MORR(250)	Optimal corridor
	PEX(250)	Penetration probability
	XMUP(250,2)	MUP for alternate warhead - (ASM for bomber group, single warhead for MRV)
	ILAW	Damage law in use
WEPSAV	IP(250)	Group payload index
	GSSBL(250)	Group SBL
	IGTYP(250)	Group type index
	IGLERT(250)	Group alert status index
	IGREG(250)	Group region index
WPFIX	NWPNS(250)	Number of weapons in group
	NFIXEZ(250)	Number of fixed assignments

Table 5. (Part 10 of 10)

BLOCK	ARRAY OR VARIABLE	DESCRIPTION
WPFIX (cont.)	NMTYP(120)	Type names
	LAM(250)	Group lambda
	ITCL(120)	Weapon type class index (1=missile, 2=bomber)
WTS	WTFAC(3)	Divide into old weight for commeasurability
	WTRATE(3)	Rate of increase of weights
	WTSUM(3)	Sum of weights
XFPX	PENALT(30)	Penetration probability for corridor

3.6 Subroutine ENTMOD

PURPOSE: Entry module for ALOC

ENTRY POINTS: ENTMOD (first subroutine called when overlay ALOC
                            is executed)

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, CNCLS, GRPHDR, INITSW, LACB, NOWPS, PAYSAV,
                            PRNTCN, SALVO, SMATAD, TABLE, WEPSAV, WPFIX,
                            OOPS

SUBROUTINES CALLED: INITIAL, MULCON, RANSIZ

CALLED BY: MODGET

Method:

First RANSIZ is called to set the record size of file 25. Next the
ALCINT overlay is read in and executed. If no input error has been
detected, the ALCMUL overlay is executed. Finally, the final multi-
pliers are written (or punched) if the user has so directed.

Subroutine ENTMOD is illustrated in figure 15.
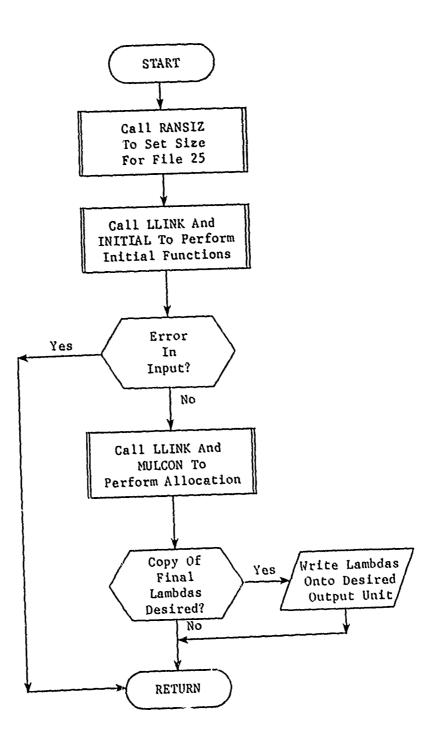
Figure 15.   Subroutine ENTMOD (ALOC)

3.7 Subroutine INITAL[*]

PURPOSE: Driver for initialization overlay

ENTRY POINTS: INITAL (first subroutine called when overlay
ALCINT is executed)

FORMAL PARAMETERS: None

COMMON BLOCKS: C15, C30, C45, INITSW, LACB, SMATAD

SUBROUTINES CALLED: DATGRP, FLOCRS, HDFND, INSGET, INTPRN, MRVRST,
PRNPUT, RDMUL, RDPRNZ, RDSET, RDSMAT, RETRV,
RNGALT, SETABL, TIMEME, TIMEPR
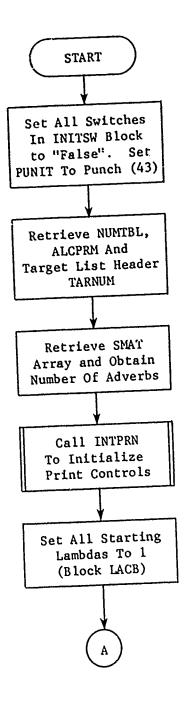
CALLED BY: ENTMOD

Method:

First all switches are initialized to a "False" value. Next, various
headers and tables are retrieved. Thus each input clause in turn is
retrieved. Each clause is analyzed by an appropriate subroutine except
the RECALC and PUNCH clauses which are handled by INITAL. Once all
clauses have been examined, the input is displayed by PRNPUT. Then
DATGRP is called to retrieve the group data and set up the payload and
type tables. Finally, the SMATAD block is built, and SETABL called if
the square root law is being used.

Subroutine INITAL is illustrated in figure 16.

---

[*]First subroutine of overlay ALCMUL.

96

Figure 16. Subroutine INITAL
(Part 1 of 4)
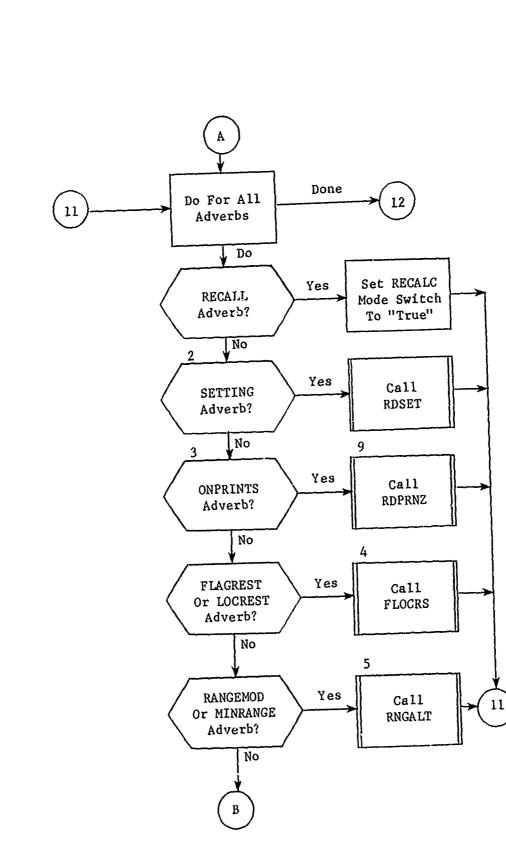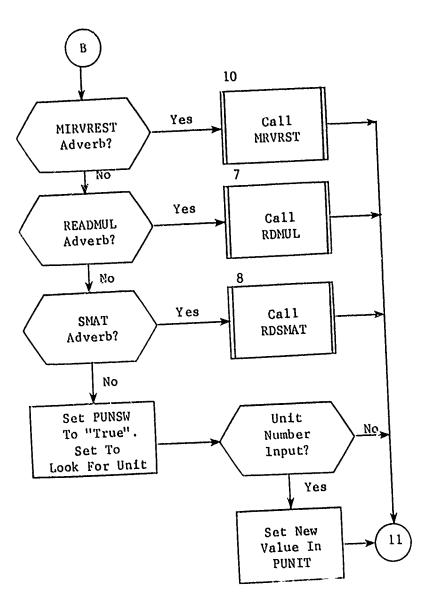
Figure 16. (Part 2 of 4)

Figure 16. (Part 3 of 4)
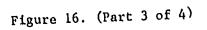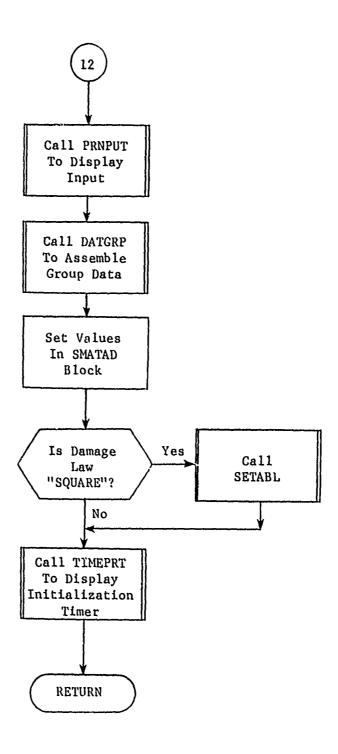
Figure 16. (Part 4 of 4)

3.7.1 Subroutine CNCLST

PURPOSE: Build tables of the classes and country locations in the target list

ENTRY POINTS: CNCLST

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C30, CNCLS

SUBROUTINES CALLED: DIRECT, HEAD, NEXITT

CALLED BY: FLOCRS, MRVRST
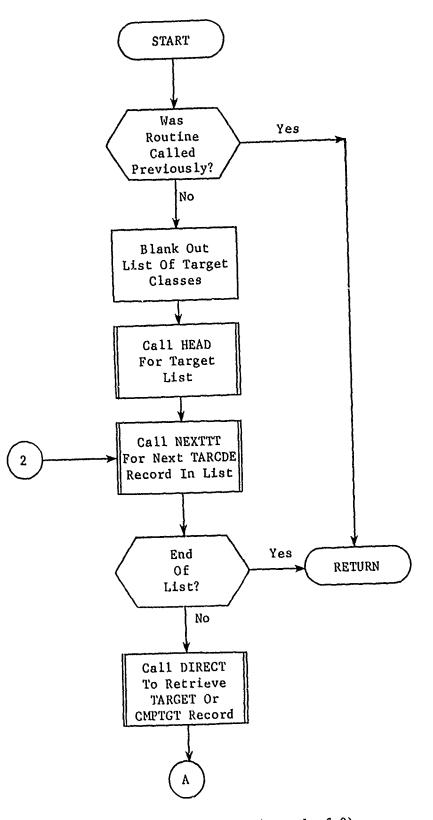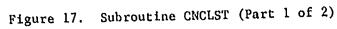
Method:

First the DONE switch is checked to see if call is necessary. Then the target list is examined. Each member of the chain may represent either a single target or a complex. If the entry is a single target its class is recorded in the CLSS list. In either case, the country location is added to the country location list (CNTRY) if it was not previously entered.

Subroutine CNCLST is illustrated in figure 17.

Figure 17.   Subroutine CNCLST (Part 1 of 2)
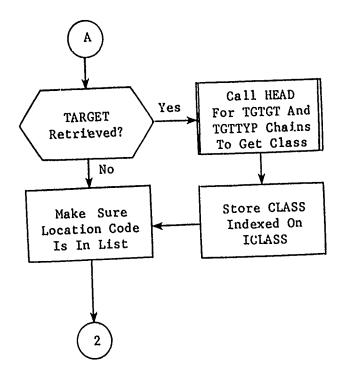
Figure 17. (Part 2 of 2)

3.7.2 Subroutine DATGRP

PURPOSE: Assemble weapon data, build payload tables and
                   initialize salvo arrays

ENTRY POINTS: DATGRP

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C30, CNCLS, FLGSTF, GRPHDR, INITSW,
                   LACB, NOWPS, PAYSAV, SALVO, WEPSAV, WPFIX

SUBROUTINES CALLED: DIRECT, GLOG, HDFND, HEAD, NEXTTT, RETRV, SLOG

CALLED BY: INITAL

Method:

After various counters are set to zero, the following process is fol-
lowed for all weapon groups. First, the TYPE and CLASS are determined
and some group attributes are saved. Next the payload for the group is
examined and compared with the payloads of previous groups. New pay-
loads are stored and old payloads are indicated via the index in array
IPAY. Next the values for file 25 are determined. This file contains
any flag, location or MIRV restrictions, range modifications and group
parameters that will be required on the first pass. Next, the weapon
totals in block NOWPS are updated and the group lambda calculated. Fin-
ally, if this is a salvoed group, the salvo arrays are initialized. This
final process includes the unpacking of NSAL (9 weapon salvos per word)
and repacking into NSALAL (4 weapon salvoes per word).

Subroutine DATGRP is illustrated in figure 18.
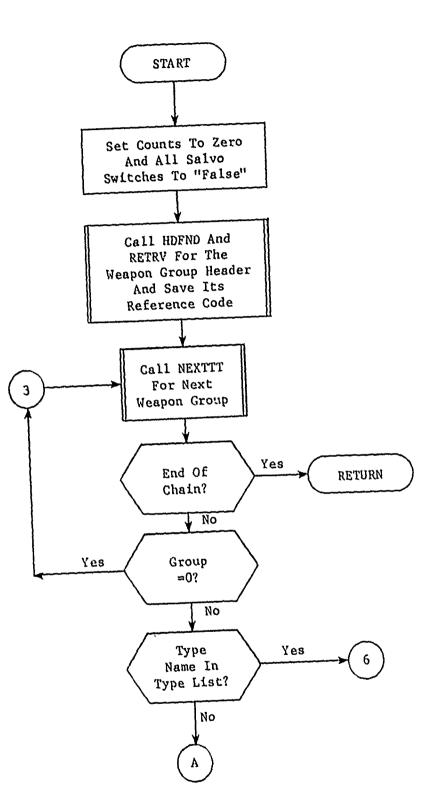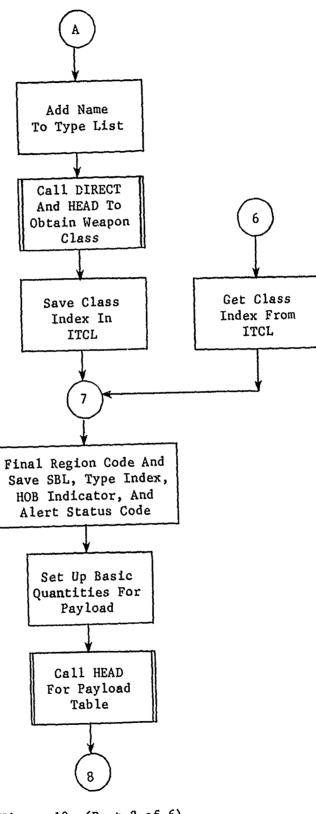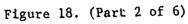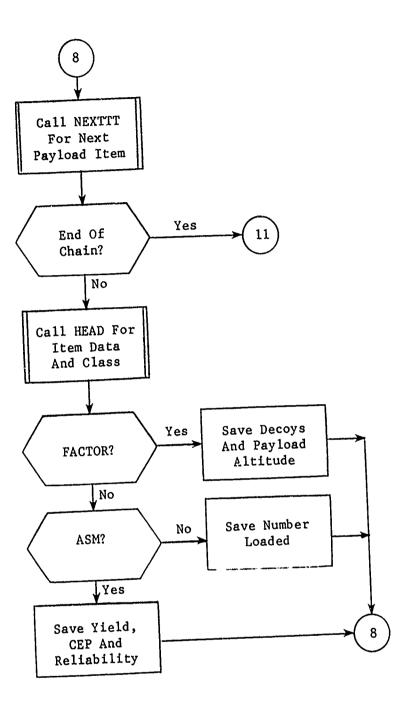
Figure 18. Subroutine DATGRP (Part 1 of 6)

Figure 18. (Part 2 of 6)

```
                    ( 8 )
                      |
                      v
          +=====================+
          ||   Call NEXTTT      ||
          ||   For Next         ||
          ||   Payload Item     ||
          +=====================+
                      |
                      v
              /----------------\        Yes
             <   End Of          >----------> ( 11 )
              \   Chain?        /
                \--------------/
                      | No
                      v
          +=====================+
          ||  Call HEAD For     ||
          ||  Item Data         ||
          ||  And Class         ||
          +=====================+
                      |
                      v
              /----------------\        Yes    +------------------+
             <    FACTOR?        >----------->  | Save Decoys      |
              \                 /               | And Payload      |
                \--------------/                | Altitude         |
                      | No                      +------------------+
                      v
              /----------------\        No     +------------------+
             <     ASM?          >----------->  | Save Number      |
              \                 /               | Loaded           |
                \--------------/                +------------------+
                      | Yes
                      v
          +------------------+
          | Save Yield,      |
          | CEP And          |-------------------------> ( 8 )
          | Reliability      |
          +------------------+
```
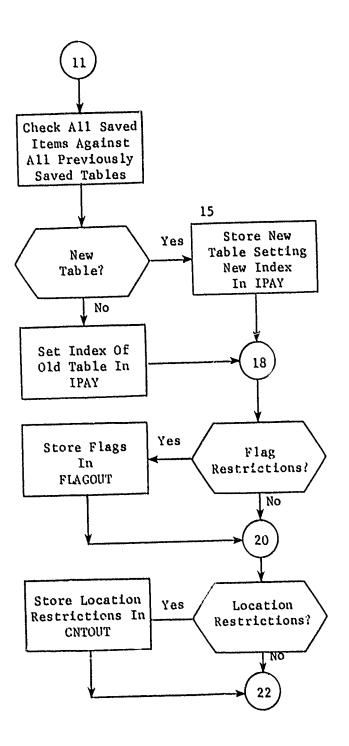
Figure 18. (Part 3 of 6)
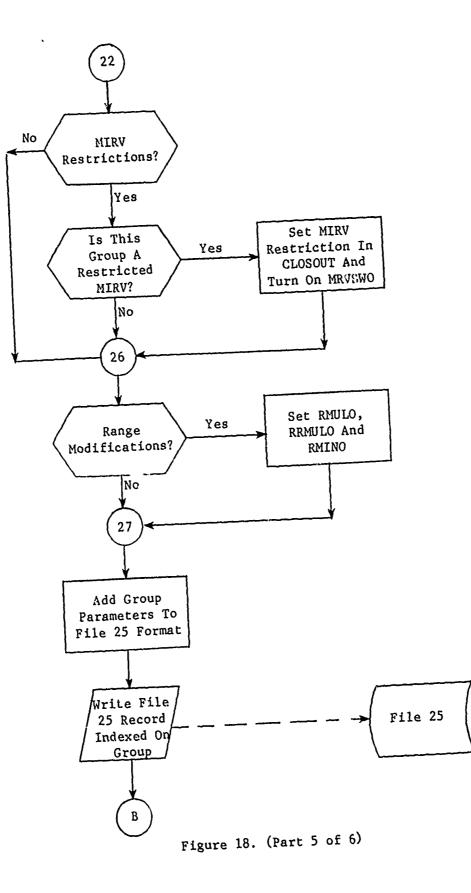
Figure 18. (Part 4 of 6)

Figure 18. (Part 5 of 6)

109
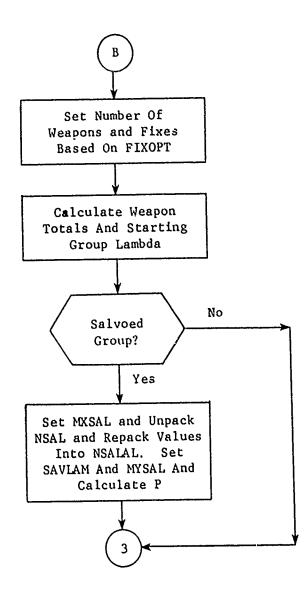
Figure 18. (Part 6 of 6)

3.7.3 Subroutine FLOCRS

PURPOSE:	To set flag and location restriction switches based on input clauses
ENTRY POINTS:	FLOCRS
FORMAL PARAMETERS:	NDEX - Starting point of clause IXBR - 1 = FLAGREST call 2 = LOCREST call
COMMON BLOCKS:	CNCLS, FLGSTF, INITSW, OOPS, ZEES
SUBROUTINES CALLED:	INCLST, INSGET, SLOG
CALLED BY:	INITAL

Method:

The method is the same for both types of call. First, if this is the first call of this type, all switches are set to indicate no restrictions. Next, the input is analyzed collecting group numbers, setting the ITYP switch (used to distinguish INCLUDE and EXCLUDE functions) and collecting either flag numbers or indexes of country codes. When either the clause ends or a new set of restrictions begins, the last set is used to supply values for the switches involved.

Subroutine FLOCRS is illustrated in figure 19.
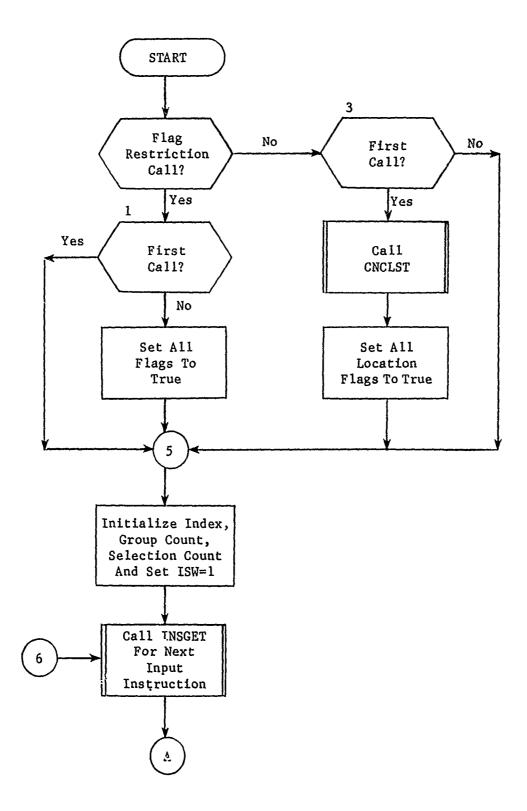
Figure 19. Subroutine FLOCRS (Part 1 of 4)

112

Figure 19. (Part 2 of 4)

113
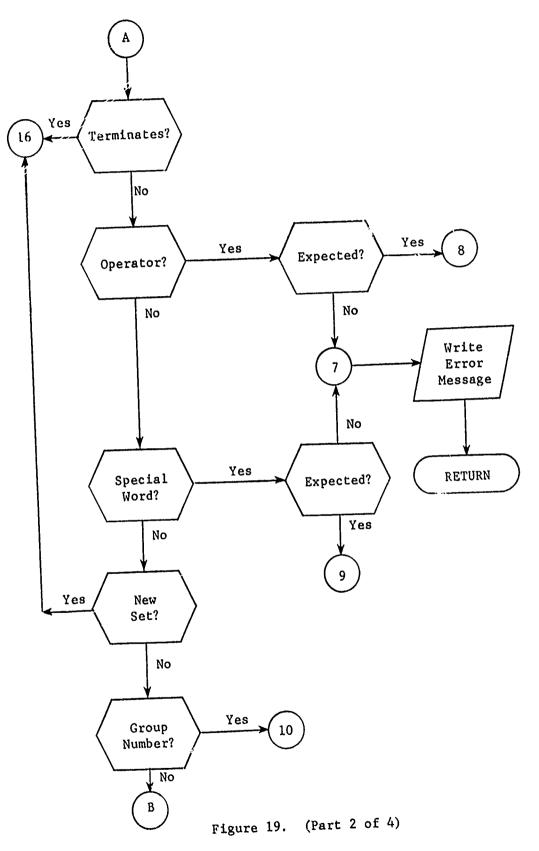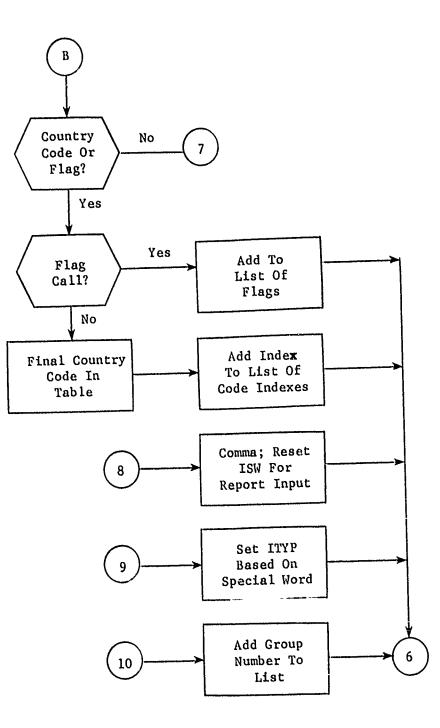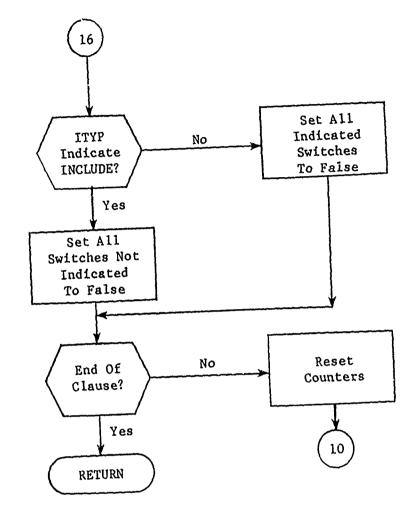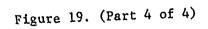
Figure 19. (Part 3 of 4)

Figure 19. (Part 4 of 4)
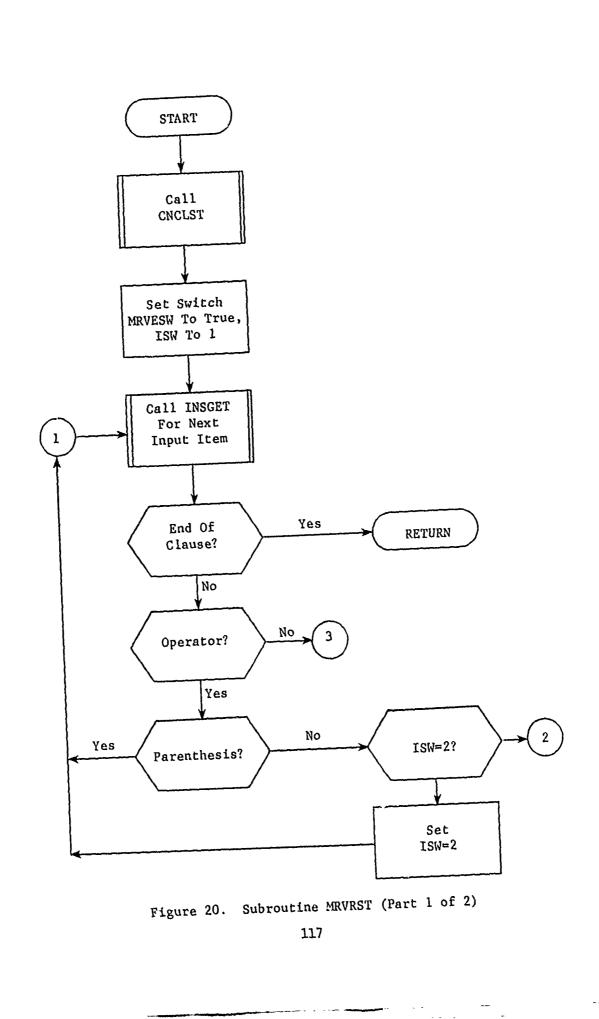
3.7.4 Subroutine MRVRST

PURPOSE: To read user input MIRV restrictions

ENTRY POINTS: MRVRST

FORMAL PARAMETERS: NDEX - Index of beginning of MIRVREST clause

COMMON BLOCKS: CNCLSZ, FLGSTF, INITSW, OOPS, ZEES

SUBROUTINES CALLED: CNCLST, INSGET, SLOG

CALLED BY: INITAL

Method:

After calling CNCLST to build the class and country code lists, the
MIRVREST clause is examined. Each MIRV payload table name is compared
to those previously entered. New names are added to the list of pay-
load table names and all switches pertaining to the new name are set
to indicate exclusion. The class names which follow the payload table
name in the input clause are used to reset the exclusion switches.

Subroutine MRVRST is illustrated in figure 20.

116

Figure 20. Subroutine MRVRST (Part 1 of 2)

117
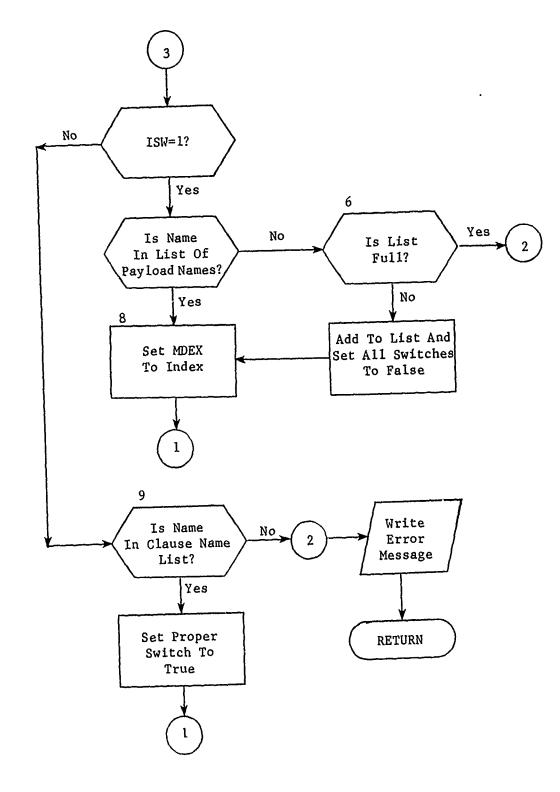
Figure 20. (Part 2 of 2)

118

3.7.5 Subroutine PRNPUT

PURPOSE: To display input user options

ENTRY POINTS: PRNPUT

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, C45, CNCLS, FLGSTF, INITSW, PRNTCON
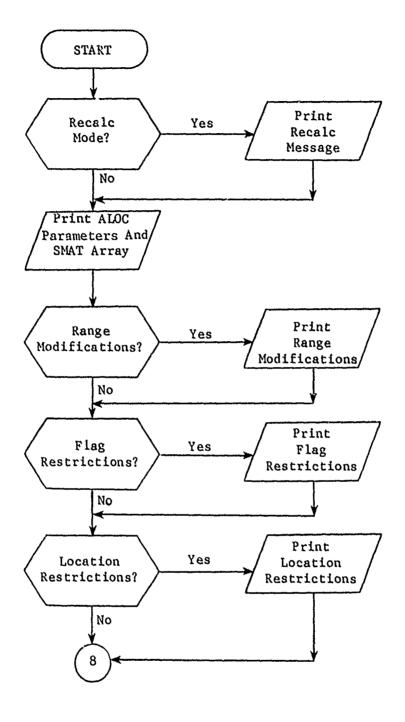
SUBROUTINES CALLED: GLOG

CALLED BY: INITAL
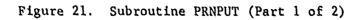
Method:

This routine prints a formatted display of allocation parameters (including the SMAT array) and selected print requests. Displays also appear of any user input restrictions or range modification if there has been input of this type.
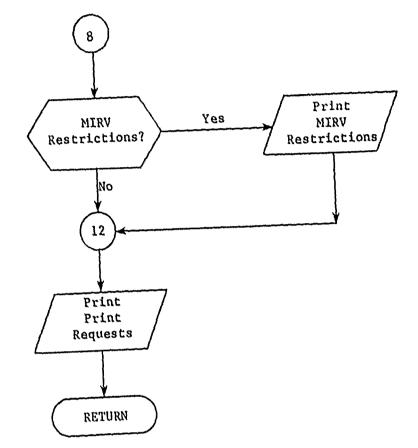
Subroutine PRNPUT is illustrated in figure 21.

Figure 21.   Subroutine PRNPUT (Part 1 of 2)

Figure 21.   (Part 2 of 2)
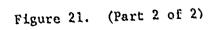
3.7.6 Subroutine RDMUL
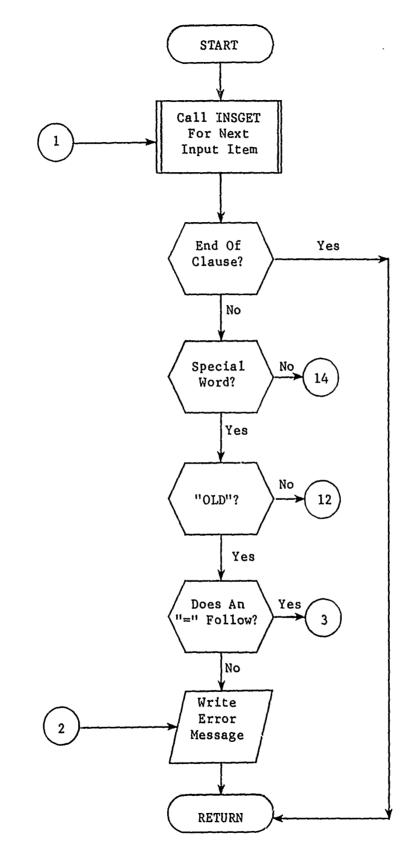
PURPOSE: To read user inputs for starting weapon values
                         (lambdas)

ENTRY POINTS: RDMUL

FORMAL PARAMETERS: NDEX - index of beginning of READMUL clause

COMMON BLOCKS: LACB, OOPS, ZEES

SUBROUTINES CALLED: INSGET

CALLED BY: INITAL
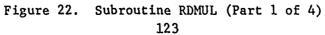
Method:

The READMUL clause is scanned and its instructions carried out. Data
will appear in two forms. First a logical unit containing previously
saved lambdas may be read. The format of this input is compatible with
the output of the PUNCH adverb.

The second type of input consists of an identifying item (such as GROUP)
an index and a value. These values are entered in block LACB as per the
identifier and the index.

Subroutine RDMUL is illustrated in figure 22.

Figure 22.   Subroutine RDMUL (Part 1 of 4)
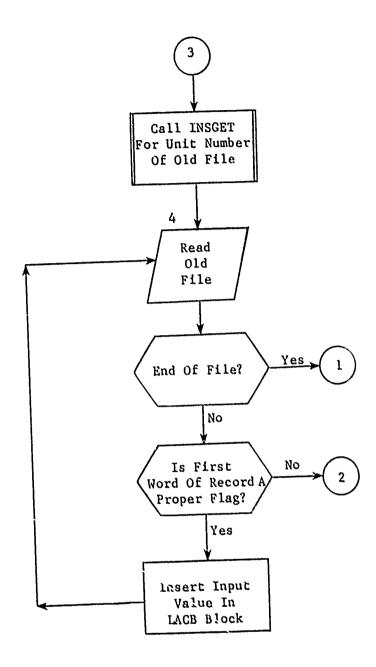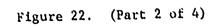
Figure 22. (Part 2 of 4)
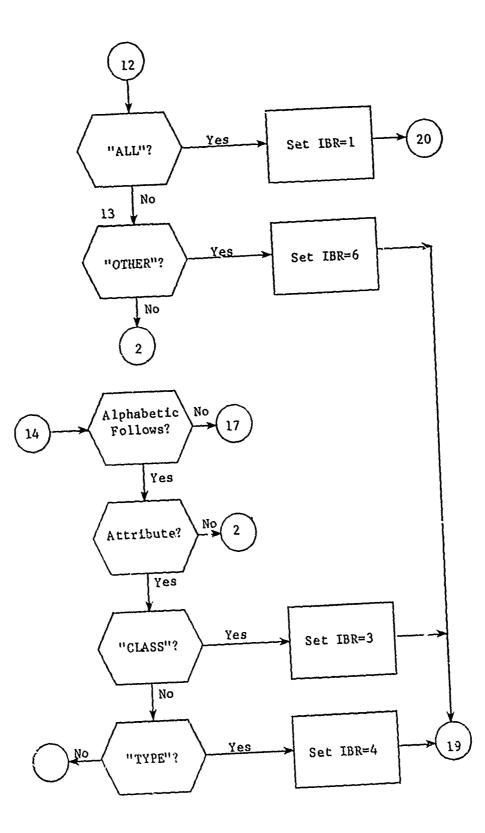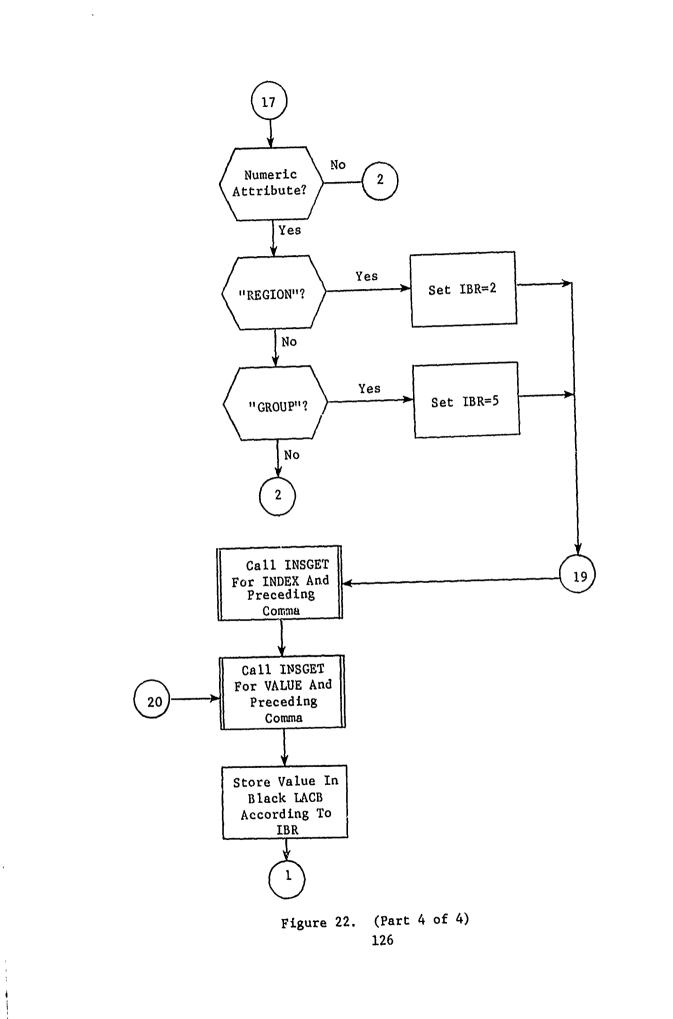
Figure 22. (Part 3 of 4)

Figure 22.   (Part 4 of 4)

126

3.7.7 Subroutine RDPRNZ

PURPOSE:	To set print options based on user input
ENTRY POINTS:	RDPRNZ, INTPRN
FORMAL PARAMETERS:	NDEX - Index of beginning of ONPRINTS clause
COMMON BLOCKS:	OOPS, PRNTCN, ZEES
SUBROUTINES CALLED:	INSGET
CALLED BY:	INITAL
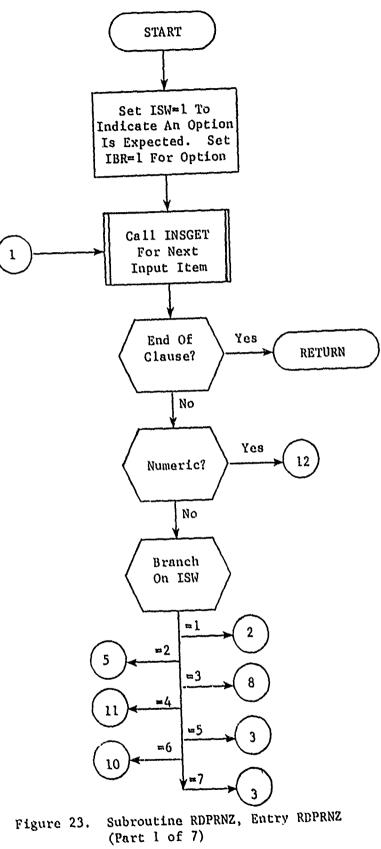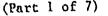
Method:

The method is best explained by examination of figure 23. As each item
is read from the clause, it is interpreted according to the current set-
tings of ISW and IBR. ISW keeps track of what the next item is expected
to be. IBR signifies whether the numeric items refer to option numbers,
targets or passes.

The INTPRN entry initializes the print options.

Subroutine RDPRNZ is illustrated in figure 23.
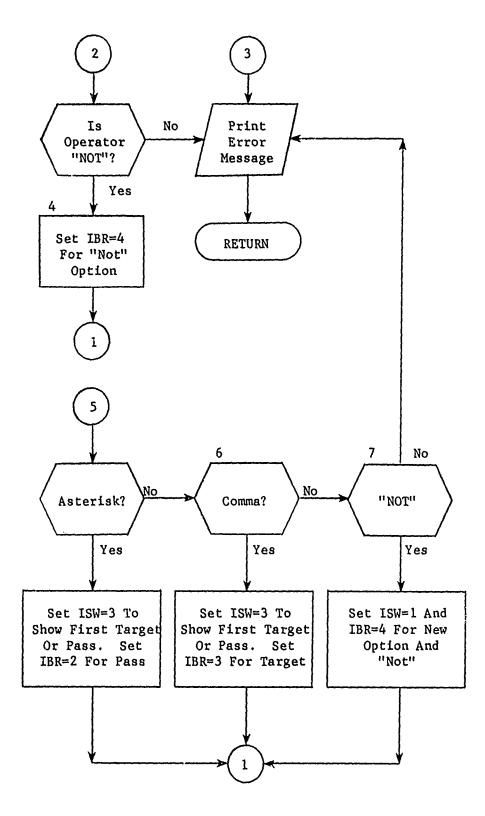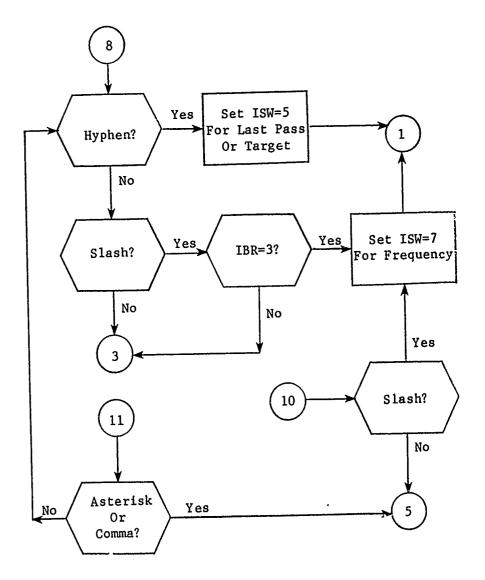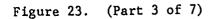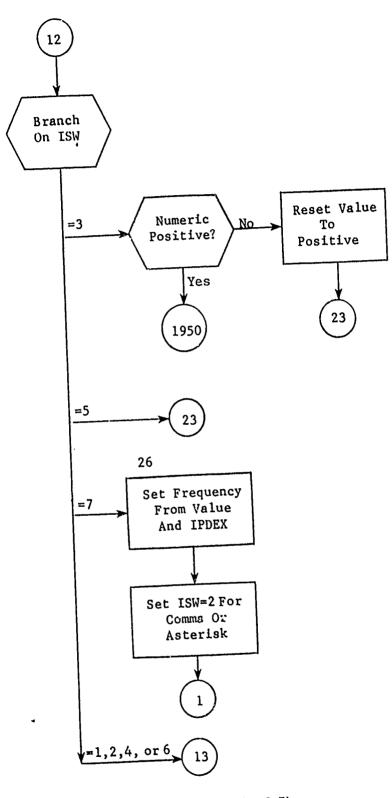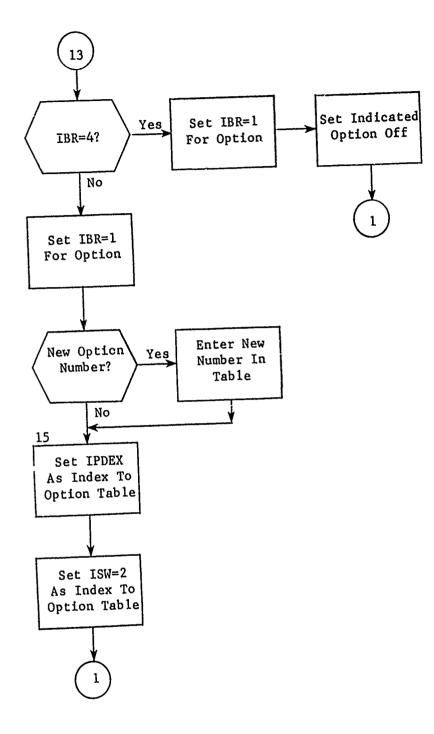
127

Figure 23. Subroutine RDPRNZ, Entry RDPRNZ
(Part 1 of 7)

Figure 23. (Part 2 of 7)

129

Figure 23. (Part 3 of 7)

130

Figure 23. (Part 4 of 7)

Figure 23. (Part 5 of 7)

Figure 23. (Part 6 of 7)

133

```
                    ┌─────────────┐
                    │    START    │
                    └─────────────┘
                           │
                           ▼
              ┌──────────────────────────┐
              │  Set All Options Off.     │
              │  Set  Standard  Limits    │
              │  And  Frequencies  For    │
              │       All Options         │
              └──────────────────────────┘
                           │
                           ▼
              ┌──────────────────────────┐
              │  Turn On Options 1,2,     │
              │     4 and 16.   Set       │
              │  Frequency For 4 At 50    │
              └──────────────────────────┘
                           │
                           ▼
                    ┌─────────────┐
                    │   RETURN    │
                    └─────────────┘
```

Figure 23.   Entry INTPRN (Part 7 of 7)
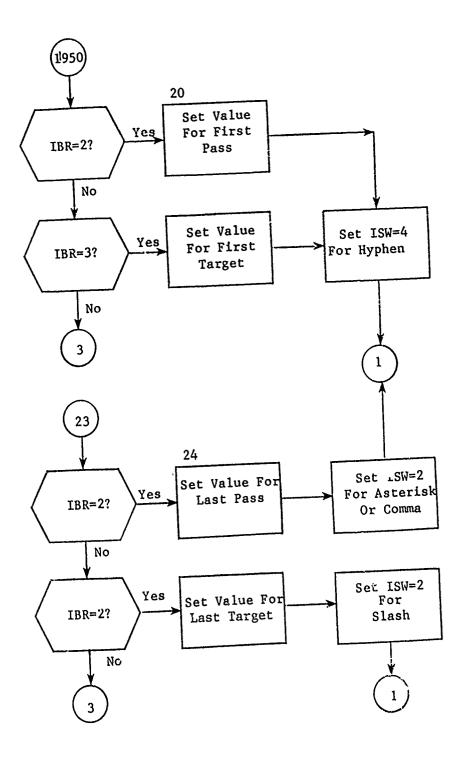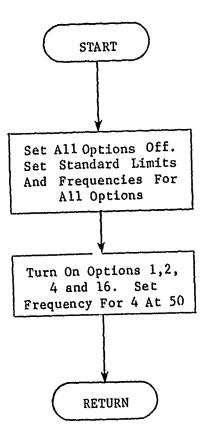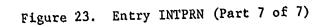
3.7.8 Subroutine RDSET

PURPOSE: To read input SETTING clause

ENTRY POINTS: RDSET

FORMAL PARAMETERS: NDEX - Index of beginning of SETTING clause

COMMON BLOCKS: C30, OOPS, ZEES
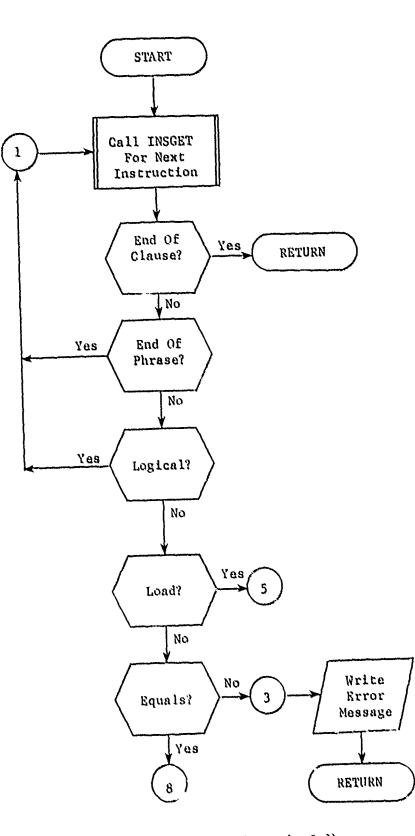
SUBROUTINES CALLED: INSGET, UNCODE

CALLED BY: INITAL

Method:

The SETTING clause is read. Each "load command" is examined to see if it pertains to ALOC parameters. If so the accompanying "equals command" is used to supply a new value.

Subroutine RDSET is illustrated in figure 24.
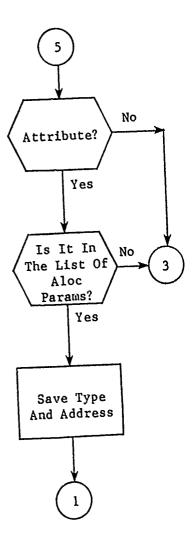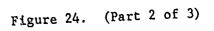
Figure 24. Subroutine RDSET (Part 1 of 3)

136

Figure 24. (Part 2 of 3)
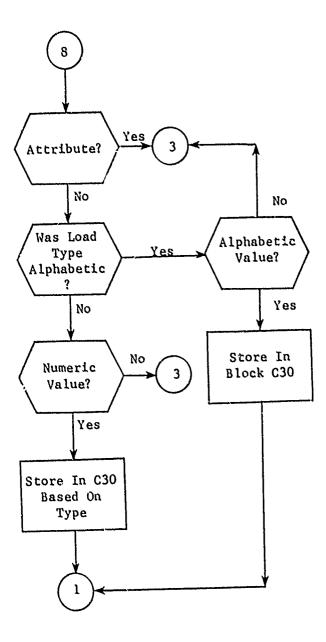
Figure 24. (Part 3 of 3)
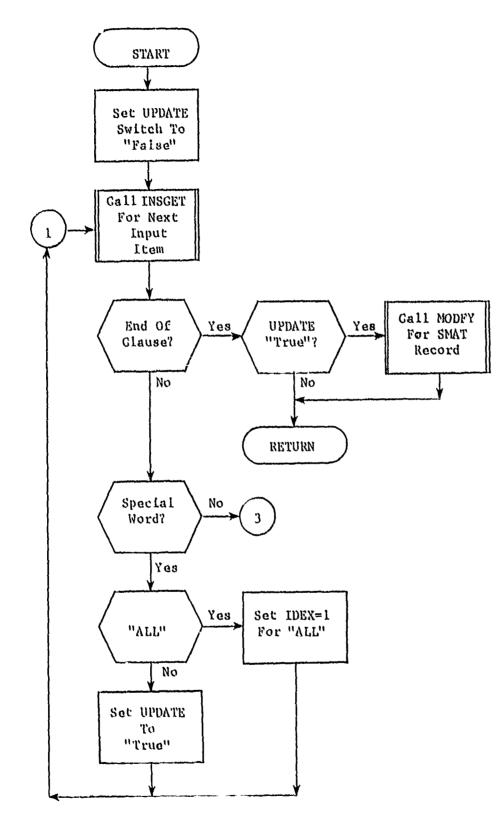
3.7.9 Subroutine RDSMAT

PURPOSE:	To read user input values for the SMAT array
ENTRY POINTS:	RDSMAT
FORMAT PARAMETERS:	NDEX - Index to beginning of SMAT clause
COMMON BLOCKS:	C45, OOPS, ZEES
SUBROUTINES CALLED:	INSGET, MODFY
CALLED BY:	INITAL

Method:

The SMAT clause is examined item by item. Ignoring commas, each alpha-
betic or attribute causes either the IDEX or JDEX index to be set. The
special word "ALL" also causes IDEX to be set. A numeric value, when
encountered, is inserted at the current settings of IDEX and JDEX, i.e.,
SMAT (IDEX, JDEX) = value. If the "UPDATE" special word is encountered
at any point, the SMAT IDS record is modified at the end of the clause.
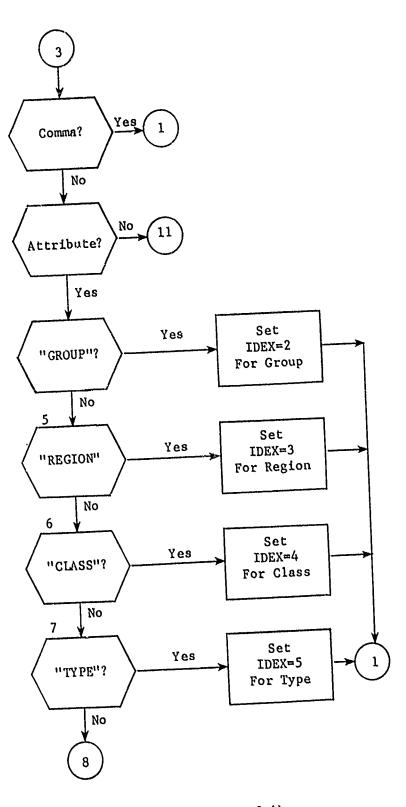
Subroutine RDCSMAT is illustrated in figure 25.

Figure 25. Subroutine RDSMAT (Part 1 of 4)

140
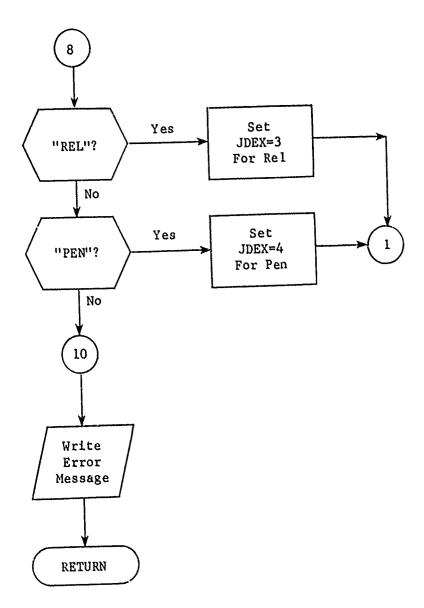
Figure 25.   (Part 2 of 4)

Figure 25.   (Part 3 of 4)
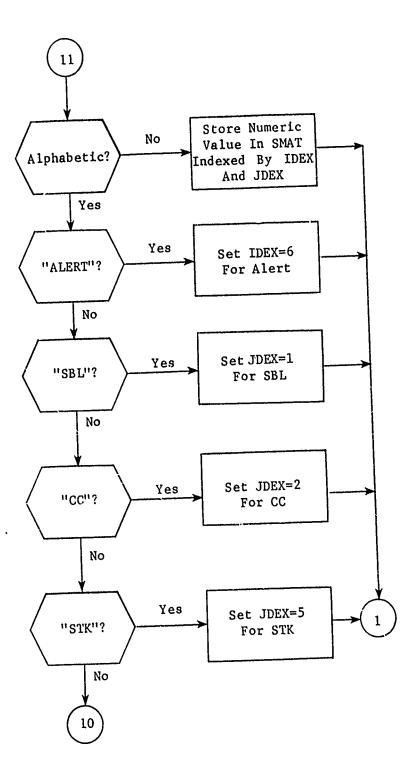
Figure 25. (Part 4 of 4)

3.7.10 Subroutine RNGALT

PURPOSE: To read user input modification to range, refueled
                     range and minimum range

ENTRY POINTS: RNGALT

FORMAL PARAMETERS: NDEX - Index of beginning of clause
                     IXBR - 3 for MINRANGE call
                            6 for MODRANGE call

COMMON BLOCKS: FLGSTF, INITSW, OOPS, ZEES

SUBROUTINES CALLED: INSGET

CALLED BY: INITAL

Method:

On the first call the switch RMODSW is set and all values are initial-
ized. Then the input clause is scanned. ISW is set to indicate the
next expected item. When a group number is read, the previous quantities
are stored depending upon whether the call was from a MINRANGE or MODRANGE
call.

Subroutine RNGALT is illustrated in figure 26.
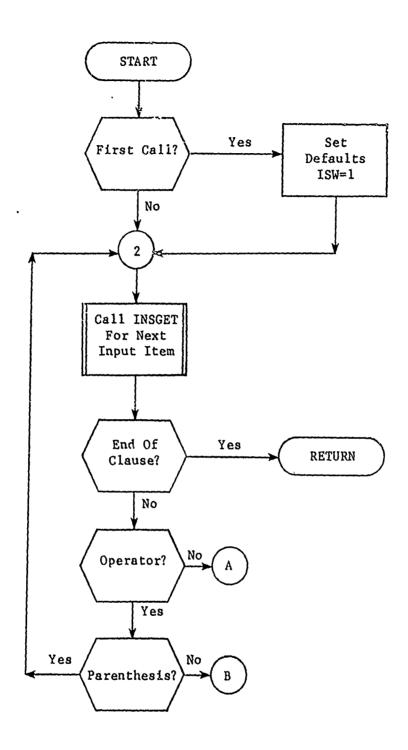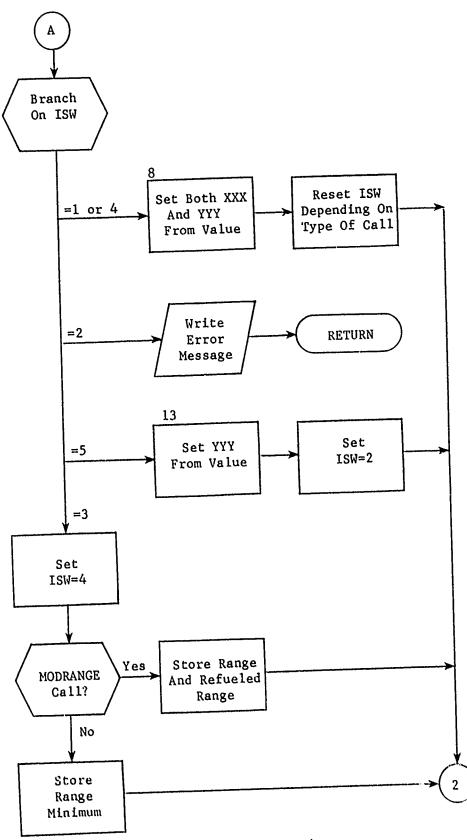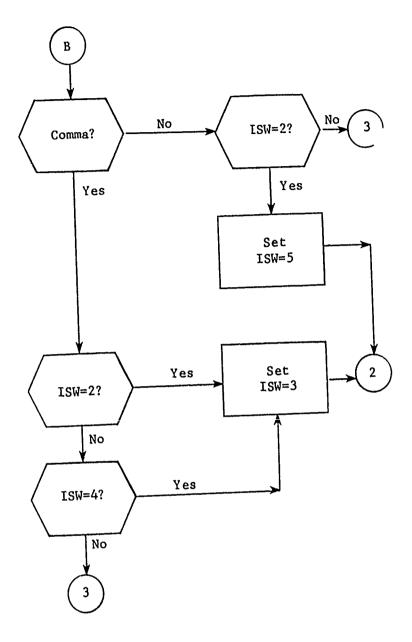
Figure 26. Subroutine RNGALT (Part 1 of 3)

145

Figure 26.   (Part 2 of 3)

146

Figure 26.   (Part 3 of 3)

3.7.11 Subroutine SETABLE

PURPOSE: This routine initializes the table which is used
                       to calculate the weapon kill factors used in the
                       square root damage law.

ENTRY POINTS: SETABLE

FORMAL PARAMETERS: None

COMMON BLOCKS: TABLE

SUBROUTINES CALLED: None

CALLED BY: INITAL

Method:

This subroutine fills common /TABLE/ with the data needed for the square
root damage law. The array, TABLE, contains values for weapon kill fac-
tors which will produce single shot survival probabilities between 0.0
and 1.0. The table entries are defined as follows:

> TABLE(i) is the square root kill factor for a single shot survival
> probability of $(i-1)*.01$. There are 101 entries in the table.
> Let:

$$SSSP = \text{Single shot survival probability}$$
$$TB = \text{TABLE array entry}$$

Then:

$$SSSP = (1+TB)*\exp(-TB).$$

During processing, function TABLEMUP will use this table to compute weap-
on kill factors. A simple heuristic root finder is used by SETABLE to
construct the table. The procedure is as follows:

Define $x_0 = 1.0$

$$SSSP = \text{Input single shot survival probability}$$

$$S_i = (1+x_i)*\exp(-x_i) \quad \text{(see statement 1).}$$
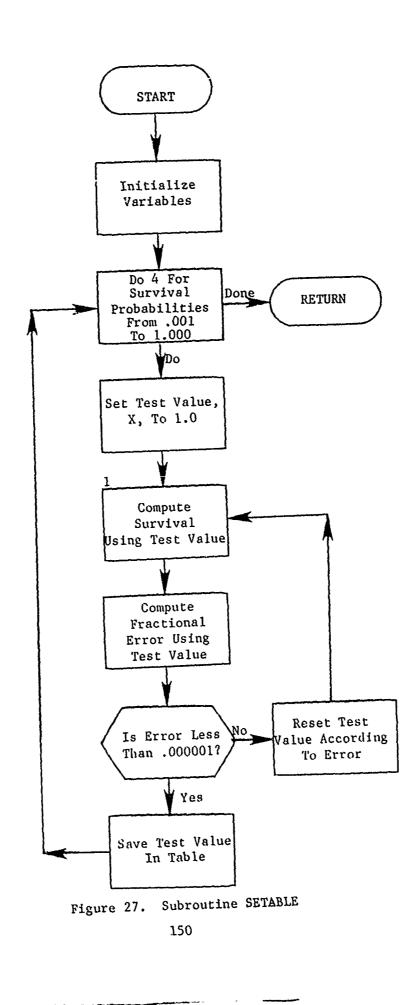
The procedure iterates on x_i such that

$$x_{i+1} = x_i + \frac{(1+x_i)}{x_i} * ERR \quad \text{(see statement 2)}$$

where $ERR = (S_i - SSSP)/S_i$.

The procedure ends when ERR .000001. The table value is the x_i which produced the S_i such that ERR met that condition (see statement 3).

Subroutine SETABLE is illustrated in figure 27.

Figure 27. Subroutine SETABLE

150

3.7.12 Subroutine TIMEPRT

PURPOSE: This routine prints the amount of time spent in
                   the options which precede the ALLOCATE function.

ENTRY POINTS: TIMEPRT

FORMAT PARAMETERS: None

COMMON BLOCKS: None

SUBROUTINES CALLED: TIMEME

CALLED BY: INITAL

Method:

This routine first calls utility subroutine TIMEME with a -2 argument
to stop the clock while the heading is being printed. After the head-
ing is complete, two calls on TIMEME are made to restart the clock (argu-
ment of -3) and print the time spent in each option (argument of 0).

Subroutine TIMEPRT is illustrated in figure 28.
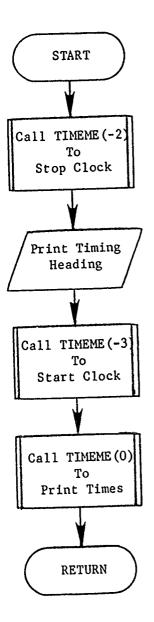
Figure 28. Subroutine TIMEPRT

152

3.8 Subroutine MULCON[*]

PURPOSE: The primary purpose of MULCON is to adjust the Lagrange multipliers during the allocation. It also monitors the progress of the allocation, and controls the flow of the module throughout the allocation.

ENTRY POINTS: MULCON

FORMAL PARAMETERS: None

COMMON BLOCKS: ALERUN, C10, C30, C33, CONTRO, CORSTF, CURSUM, DYNAMI, FIRST, GRPHDR, LACB, MULTIP, NALLY, NOWPS, PAYOFF, PAYSAV, PREMS, PRTMUL, REFPNT, SALVO, SPLITS, SURPW, TARREF, TGTSAV, WADFIN, WADOTX, WADWPN, WEPSAV, WPFIX, WTS

SUBROUTINES CALLED: ADDSAL, ASGOUT, BOMPRM, DEFALOC, DIRECT, FRSTGD, LLINK, MODFY NEXTTT, PRNTALL, PRNTCON, PRNTNOW, SCNDGD, STALL, TIMEME

CALLED BY: ENTMOD (ALOC)

Method:

MULCON proceeds by making multiple passes through the target list until the correct number of weapons have been allocated. During this process, the Lagrange multipliers are modified to force the allocation to converge to the given stockpile.

The flow of operations in MULCON is illustrated in figure 29. The first sheet represents the flow in five parts of the program; Part I, the initialization phase; Part II, first pass processing; Part III, the main flow; Part IV, processing after allocation; and Part V, multiplier adjustment. The flow diagrams are annotated with statement references to the FORTRAN listing and are in sufficient detail to be largely self-explanatory. In the following sections comments are made only where the flow diagrams require some explanation.

Part I: Initialization

The routine begins by calling utility subroutine TIMEME to initialize the clock. NPASS and ITGT are initialized so that the print control routine PRNTCON will know what prints to activate, and then PRNTCON is called to do so. A call for a summary print of starting multipliers (PRNTALL(11)) follows.

---

[*] First routine of overlay ALCMUL.

The sensitivity of the allocation errors to the value of the local multipliers is estimated. With a fairly large value for the linear premium PRM all sensitivities are about the same independent of the size of the group; thus, PARTIAL(J) is simply set to -1.0.

RUNSUM and WTSUM are initialized as if the starting pseudoallocation were exact, but the weight given to that allocation is reduced by the square root of the number of targets so that it does not take too long for data on actual allocation rates to produce a significant effect on the estimated rates.

Part II: First Pass Processing

This section deals with the target processing on the first pass. On this pass FRSTGD is called which processes the target, brings in the weapon data and, if necessary, calculates the Weapon/Target Data File. FRSTGD also reads in any fixed allocation. Finally, the pseudoallocation is removed from the running target weights.

Part III: Main Flow (After First Pass)

This section deals with the target processing on passes after the first. First a series of calls to PRNTALL performs intertarget prints. Next PRNTCON is called to reset print controls. Then, if this is not the first pass processing continues. SCNDGD is called to process target data, bring in the weapon related data and read in the old allocation. If PROGRESS is 2 the variable IVERIFY is checked. If this is a verification pass the limits are checked and if not achieved the pass continues. If all processing is complete PRNTNOW is called for final print. Then the weapon group chain is cycled and the attribute NUMALOC set equal to RNALL for each group.

However, if processing is to continue, the termination section is ignored and the program gets ready to generate a new allocation to replace the one just read. On the first pass the old allocation is a pseudo-allocation and the replacement is done elsewhere (see Part II). The replacement is accomplished by removing the contribution of the old allocation[*] to all running sums before the new allocation is generated. This can be done in this simple way because the values of COST, PAYOFF, PROFIT, TGTWT, and DPROFIT, then in memory, are those just read in from the TARCDE record and so they correspond to the old allocation. Since the quantities RUNSUM and WTSUM were divided by WTFAC at the end of the last pass, the old TGTWT must also be divided by this same factor to make it commensurate before it is subtracted out

The reason that REVCOST must be computed is that the values of the multipliers have probably changed (unless PROGRESS = 1.0) since the prior

---

[*] The fixed weapons are ignored because they do not contribute to the running sums.

154

allocation; consequently a revised cost (REVCOST) of the allocation based on the new multipliers is of interest, and is probably different than the old cost COST. The reason for the test on PROGRESS before correcting the cumulative differential profit will be discussed in connection with Part IV of the program (part 5 of figure 27).

Subroutine ADDSAL is used to maintain the sums for the salvoed groups.

Part IV: Processing After Allocation

Before calling STALL,* CTSPILL is set to 0. (If some elements are spilled, WAD will so note by setting CTSPILL equal to the number of elements spilled.) The calls on TIMEME (5, 6, 7) before and after the allocation cause the time spent during the actual allocation to be recorded in columns 6 and 7 of the TIMEME output print (number 23).

Before calling either allocation routine, however, the program must check the number of fixed weapon assignments. The limitation of weapons allocated to one target is 30 weapons on an undefended target and 30 weapon groups on a defended (i.e., terminal ballistic missile defenses) target. Usually, MULCON calls both STALL* and DEFALOC* on defended targets and chooses the best allocation. If there are more than 30 fixed weapon assignments, STALL* should not be called. If the number of fixed assignments is greater than 30, MULCON checks to see if it is a defended target. If not, an error message is printed and the excess assignments are ignored. Then STALL* is called. If it is a defended target, MULCON sets a dummy low profit (except for verification) and calls only DEFALOC.*

The additional details of the allocation required by later processors are then recorded in /DYNAMI/. PENX and TOARR are required by EVALALOC, while KORRX and RVALX are required by ALOCOUT, FOOTPRNT, and POSTALOC. Subroutine BOMPRM is then called to update the ASM allocation fraction array FASM.

The various running sums are then calculated. If DEFALOC has made the allocation, the KORRX array gives the number of missiles from each group allocated to the target. If KORRX is positive, it represents the corridor. If it is negative, it represents the number allocated. Then the profit and cost data are recorded.

The following quantities are of particular importance:

$DPROFIT = PROFIT - OPROFIT$

$SDPROFIT = \sum DPROFIT$

$DELTEFF = DPROFIT/VALWPNS$

$SDDLTEFF = SDPROFIT/VALWPNS$

---

* STALL and DEFALOC are called via computer system subroutine LLINK.

155

These quantities are computed and the last two printed out in the stand-
ard ALOC print number 2 to help the user evaluate the progress of the
allocation.* The quantity OPROFIT represents the profit of the old allo-
cation to the target evaluated in terms of the present values of the
Lagrange multipliers. DPROFIT is thus a measure of the improvement in
profit using the new allocation. Until PROGRESS = 1.0 this quantity
is summed over all targets (one complete pass only) to give SDPROFIT.
Thus, when the multipliers have been near the correct values for one
full pass the value of SDPROFIT should be small. To provide a standard
relative value for interpreting these quantities, they are divided by
the value of all the weapons VALWPNS,

$$VALWPNS = \sum NWPNS(G) * LAMEF(G)$$

to obtain DELTEFF and SDELTEFF which measure changes in profit as a
fraction of the total value of all weapons.

The quantity of SDELTEFF, therefore, provides an estimate of how effi-
cient the allocation would have been if the allocation had been termin-
ated one pass earlier. Presumably, the current efficiency is substan-
tially higher, but SDELTEFF does not, at this point, give any indication
of how much. It is nevertheless of value in developing experience on
how soon the PROGRESS .75 phase can be terminated. When PROGRESS is
equal to 1.00 the multipliers are frozen and this role of SDELTEFF
ceases to be relevant. The quantity is then reset to 0. Thereafter it
provides a measure of the effect on the profit of closing to the exact
stockpile. Usually during the closing phase SDELTEFF goes slightly
negative. However, since during this phase we continue to replace allo-
cations originally produced with slightly different values of the multi-
pliers, the value may go positive for a while until the closing forces
get large enough to force closure even at some loss of profit. Thus the
value of SDELTEFF at the end of the closing phase (PROGRESS = 1) measures
the loss in profit associated with closing. In the event that closing
requires more than one full pass, a test has been inserted which causes
SDELTEFF to continue to accumulate over more than one pass when PROGRESS
= 1.0.

Finally when PROGRESS = 2.0 the quantity is again set equal to 0. If a
verification pass is carried out, SDELTEFF then measures any increase in
profit in the verification pass relative to the final allocation. In
this role it defines an upper limit on the inefficiency of the actual
allocation.

Ordinarily after all these calculations are performed ASGOUT is called
to store the results. However, if PROGRESS = 2 this step is skipped
since the integrated data file already contains the final allocation.

---

* The column labelled (P-O)/VWPS in print number 2 contains these vari-
  ables: DELTEFF on the first line, SDELTEFF on the second.

At the end of each complete target, the target weight is adjusted. If it is also the end of a pass, the target weight is still adjusted by the same amount relative to other targets, but all target weights and the value of RUNSUM and WTSUM are renormalized.

After the target weights are adjusted, a test is made to see if it is time to recompute the multipliers. If so, control is transferred to point D of Part V.

Two other operations, however, require special comment. At the bottom of the diagram a test is made to see if sufficient progress has been made. If after three passes PROGRESS has not reached .75, it is assumed that a problem exists and the run is terminated. In the middle of the diagram, at the end of the first pass, the value of NOWPS(J) is reevaluated omitting any weapon groups that could not reach any targets. This allows the allocator to ignore such weapon groups thereafter, and avoids an endless and fruitless effort to allocate these weapons by reducing their Lagrange multipliers. The array TVALTOA provides a convenient test, since it is initialized to zero and will remain zero only for weapon groups that have never been within range of any target.

Part V: Multiplier Adjustment

Every fourth target or so, [*] when it is decided to recompute the multipliers, controls passes to this adjustment routine. The first step is to recompute all the allocation error estimates, ALERREST. At the same time SURPWP, the excess allocation estimate, is reevaluated based on the new value of ALERREST. Although SURPWP is continuously updated by the operating program it is useful, especially in the early phases of the program, to base it on the projected allocation rate estimates rather than the actual weapons allocated, which at that time could be very misleading. This provides a more rational basis for calculating the premiums at this early stage of the program.

If PROGRESS = 1.0, the change of local multipliers is omitted so that the same values of the multipliers are retained. Otherwise control passes over the local multipliers to the Do loop. Each multiplier is changed only if all the estimates of error rate have the same sign. In the early phases of the program (PROGRESS .LT. .75) better stability is achieved by requiring, in addition, that the average allocation rate to the last two to four targets (as computed from CURSUM) show the same sign. This limitation is later removed, since it clearly would not work well for weapon groups with very small numbers of weapons that might only be allocated 20 to 10 times during a pass over the target system.

An estimate is made of CORRATE, the rate at which it is desired to correct the allocation rate. If the allocation rate is corrected too rapidly there will be a tendency to over correct before the effects of the

---

[*] Every second target for PROGRESS equal to 0.0, 0.4, and 0.5; every fourth target for PROGRESS equal to .75 and 1.00.

correction become observable in the values of the allocation error estimates. This can produce oscillations. To estimate how rapidly to correct the error, an estimate is made of the number of targets that would have to be observed before an error of the observed size would be statistically significant. Even if the multipliers were exact and the average allocation rate was correct, statistical fluctuations would be observed in the allocation of each weapon group when the allocation rate was sampled for a small number of targets.

Let n equal the expected or average number of weapons from a group available per target; i.e., $n = NOWPS(J)/NTGTS$. Then in M targets the expected number of weapons allocated should be just $n(M)$. Suppose the actual number observed however is $n'(M)$. Then our estimate of the error in the allocation rate ALERREST would be

$$ALERREST = n' - n$$

Assuming a Poisson distribution, the statistically expected error in a number of expected value $n(M)$ is equal to $\sqrt{n(M)}$. That is,

$$(n'(M) - n(M))^2 = n(M)$$

$$(N' - n)^2 = n/M$$

Solving for the number of targets M we have:

$$M = n/(n' - n)^2$$

or

$$M = (NOWPS(J)/NTGTS)/(ALERREST(J))^2$$

as the number of targets we should expect to sample to get a statistical error estimate of size, ALERREST. If we wish to reduce the indicated error by 1 part in M per target, our fractional correction in the allocation rate per target should be:

$$1/M = ALERREST^2 / (NOWPS(J)/NTGTS)$$

This, multiplied by a sensitivity factor SNSTVTY, is the first term in the value of CORRATE. However, if the entire set of targets were observed, the estimate would not be a sample but would be exact. Therefore, even a very small value of ALERREST becomes statistically significant if it is based on a sample of size NTGTS. Therefore, errors should always be corrected at a rate at least equal to one part in NTGTS. This explains the second term in CORRATE which is just 1.0/NTGTS multiplied by a sensitivity factor FSNSTVTY (final sensitivity). This factor controls the sensitivity of corrections to the allocation rate in the final phase of the allocation where the errors are small. Thus the desired correction rate is just:

158

$$CORRATE = SNSTVTY * ALERREST^2 / (NOWPS(J)/NTGTS) + FSNSTVTY/NTGTS$$

This is multiplied by the number of targets processed between corrections MULSTEP to determine the fraction CORFAC of the error to correct. In addition, a safety limit of 1/2 is used to avoid ever making a correction larger than 1/2 the estimated error rate.

However, even when it is known what fraction of the error in the allocation rate we wish to correct, an estimate must be made of the relationship of the allocation rate to changes in the Lagrange multipliers before the size changes to make in the multiplier can be estimated. For this purpose it is useful to have a model of the dependence of the allocation rate on the value of the multipliers. We have assumed a dependence as follows:

$$Rate = K\lambda^{-n}$$

Consider now two rates, the current rate R_0 associated with a multiplier λ_0 and a predicted rate R_1 associated with a new multiplier λ_1. Thus, we find

$$R_1 \lambda_1{}^n = R_0 \lambda_0{}^n = k$$

or

$$R_1/R_0 = (\lambda_1/\lambda_0)^{-n}$$

so

$$\frac{(R_1/R_0)}{(\lambda_1/\lambda_0)} = -n$$

For small differences between λ_0 and λ_1 this implies:

$$\frac{R_1 - R_0}{R_0} = -n \frac{\lambda_1 - \lambda_0}{-\lambda_0}$$

Solving for the new value λ_1 of λ

$$\lambda_1 = \lambda_0 \left(1 + \frac{(R_1 - R_0)/(-n)}{R_0}\right)$$

If we now identify a new variable R_2 as the ultimately desired allocation rate, R_1 as the new rate we hope to obtain with λ_1, and R_0 as the current allocation rate -- then the above variables can be associated with information already available as follows:

$$R_1 - R_0 = CORFAC * (R_2 - R_0) = CORFAC * ALERREST$$

159

$$R_0 = \text{ALERREST} + (\text{NOWPS}/\text{NTGTS})$$

If we now associate the FORTRAN variable PARTIAL with n and the local multiplier LA with λ this gives rise to the following procedure for updating LA:

$$\text{LA}_1 = \text{LA}_0 * 1.0 + \left[\frac{\text{CORFAC} * \text{ALERREST}(J,\text{INTPRD})/(-\text{PARTIAL})}{\text{ALERREST}(J,\text{NTPRD}) + (\text{NOWPS}(J)/\text{NTGTS})} \right]$$

This formula is well behaved if ALERREST is large and positive, but if it is negative and as large as the expected rate (NOWPS(J)/NTGTS) (i.e., if the actual allocation rate is zero), then the denominator goes to 0. In this case an infinite correction would be indicated. To avoid this, the expected rate in the denominator is multiplied by 2 giving:

$$\text{LA}_1 = \text{LA}_0 * \left[1.0 + \frac{\text{CORFAC} * \text{ALERREST}(J,\text{INTPRD})/(-\text{PARTIAL})}{\text{ALERREST}(J,\text{INTPRD}) + 2 * (\text{NOWPS}(J)/\text{NTGTS})} \right]$$

In the present version of the program the value of PARTIAL(J) has been set equal to 1.0 for all the local multipliers LA(J). This choice is based on the effect of the premium on the sensitivity of the allocation rate to the value of LAMEF or λ. When the multipliers are almost correct, it is usually the case that most weapon groups are in·close competition with many other groups with very similar properties. Then a small change in the multiplier LAMEF will produce a very large change in the allocation rates, as the weapon group in question almost totally replaces, or is replaced by, its competitors.

However, such a large error in the allocation rate will not actually occur because as the error builds up the estimated value of the payoff will be automatically changed by the premium. Thus, for constant values of LAMEF, when an equilibrium allocation rate is reached, it must be approximately true that the error in LAMEF is compensated by the premium. That is, if λ_0 is the correct value for LAMEF then:

$$\text{LAMEF} - \text{PREMIUM} \cong \lambda_0.$$

Since

$$\text{PREMIUM} = \text{PRM} * \text{LAMEF} * \frac{\text{SURPWP} - .5 * \text{CTMULT}}{\text{NWPNS}}$$

we can define a relation between LAMEF and (SURPWP/NWPNS).

$$\text{LAMEF} * \left(1 - \text{PRM} * \frac{\text{SURPWP} - .6 * \text{CTMULT}}{\text{NWPNS}} \right) \cong \lambda_2$$

Since this relationship is the same for all groups it is reasonably simple to use the same value 1.0 of partial derivative for all local multipliers.

The values of LAMEF(G) are recomputed using the new values of the local multipliers LA(J). At the same time it is necessary to reevaluate the summation of the value of all the weapons VALWPNS $= \sum$LAMEF(G) $*$ NWPNS(G) and the summation of the value of the error in weapons allocated.

VALERR $= \sum$LAMEF(G) $*$ ABSF(SURPWP(G)) using the updated values of LAMEF.

The average number of targets over which allocation rates are averaged (the integration period) is determined by the rate at which the target weights are increased.

In estimating the rate with which to correct multipliers, it was computed on a statistical basis that even if the allocation rates were correct an estimated error of size ALERREST would be expected if the allocation rates were monitored only over a small sample of M targets where:

$$M = (NOWPS(J)/NTGTS)/(ALERREST(J))^2$$

Thus, if separate integration periods could be used for each local multiplier, M, as defined above, might provide a reasonable basis for determining the period. However, in fact, the same three periods (INTPRD = 1, 2, 3) must be used for all local multipliers LA(J). Consequently, the value of the integration period used might be based on an estimate of overall error rate. The corresponding relation is:

$$M = (\sum_G NOWPS(J)/NTGTS)/\sum_G (ALERREST(J))^2$$

where the summations are taken over all weapon groups. The quantity, $\sum_G$NOWPS(J), is identical with NOWPS(2) and so for efficiency the variable NOWPS(2) is used.[*] While the expected value of $(ALERREST(J))^2$ is the same as $\sum_G (ALERREST(J))^2$ the variance of the later version is much less and it is therefore preferable as an estimator of the expected integration period, EXPINTPD.

To allow the possibility of using integration periods either longer or shorter than the theoretical EXPINTPD, a desired longest integration period DESINTPD is defined:

$$DESINTPD = EXPINTPD * RATIONT$$

where RATIOINT is an adjustable input parameter.

If this period were used exactly in setting the rate of change of the target weight; i.e., WRATE = 1.0/DESINTPD, and WRATE would never become exactly 0 as is required for a constant target weight. Obviously when

---

[*] LA(2) is used for all weapon groups.

161

the change in the target weight becomes small over a full pass, the WRATE should be allowed to go to 0. Therefore,

$$WRATE = (1.0/DESINTPD) - (2.0/(NTGTS * RATIOINT))$$

the term $2.0/(NTGTS * RATIOINT)$ is subtracted and if the resulting WRATE is negative it is set to zero. To avoid a situation where large errors cause the integration period to become ridiculously small, a limit that WRATE $\leq$ .07 is set.

Moreover, after the allocation is well underway, PROGRESS $\geq$ .5 the value of WRATE is not allowed to increase. In the program WTRATE(INTPRD) is used as a multiplier of the target weight; therefore, we all 1.0 to WRATE to obtain a suitable multiplier for the longest period NINTPRD.

The values of the WRATE for the shorter periods are then derived from this value to give a ratio of integration periods (roughly equal to RINTPRD) another input parameter.

The evaluation of progress is shown in the final sheet for the subroutine MULCON. The procedure is very straightforward, and should be obvious from the flowchart. It may be worth noting, however, that when the allocation is finally complete, the index IFINTGT of the last target and the final pass IFINPASS, are recorded.
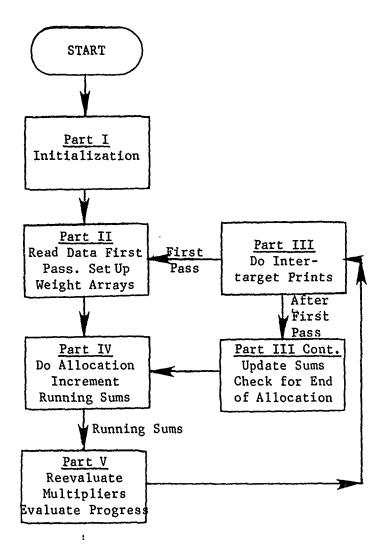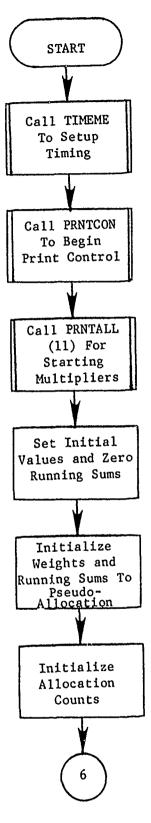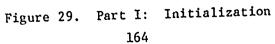
Figure 29. Subroutine MULCON Summary Flow

163

Figure 29. Part I: Initialization

164

Figure 29. Part II: First Pass Processing

Figure 29. Part III: Main Flow (After First Pass)
(Part 1 of 3)

166

Figure 29. Part III: (Part 2 of 3)
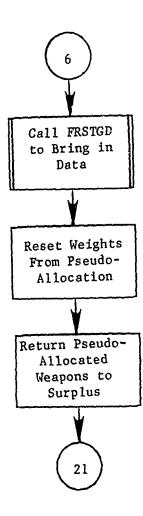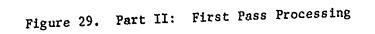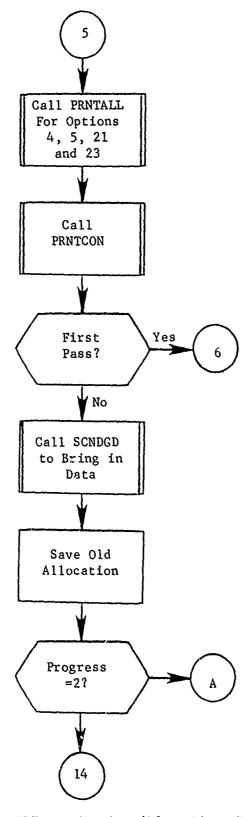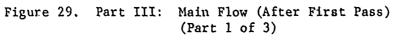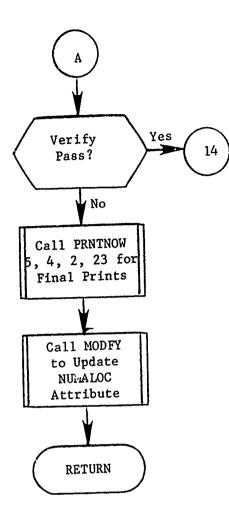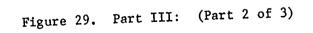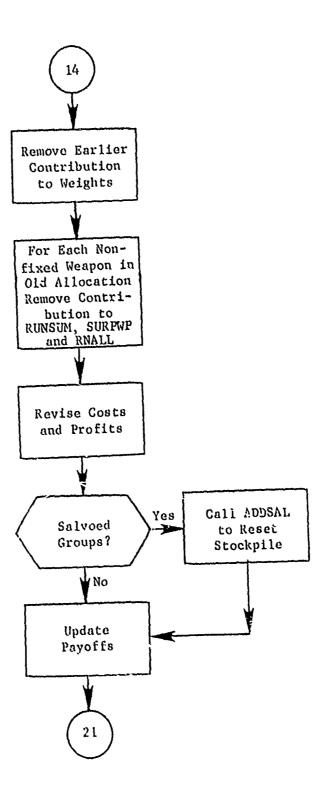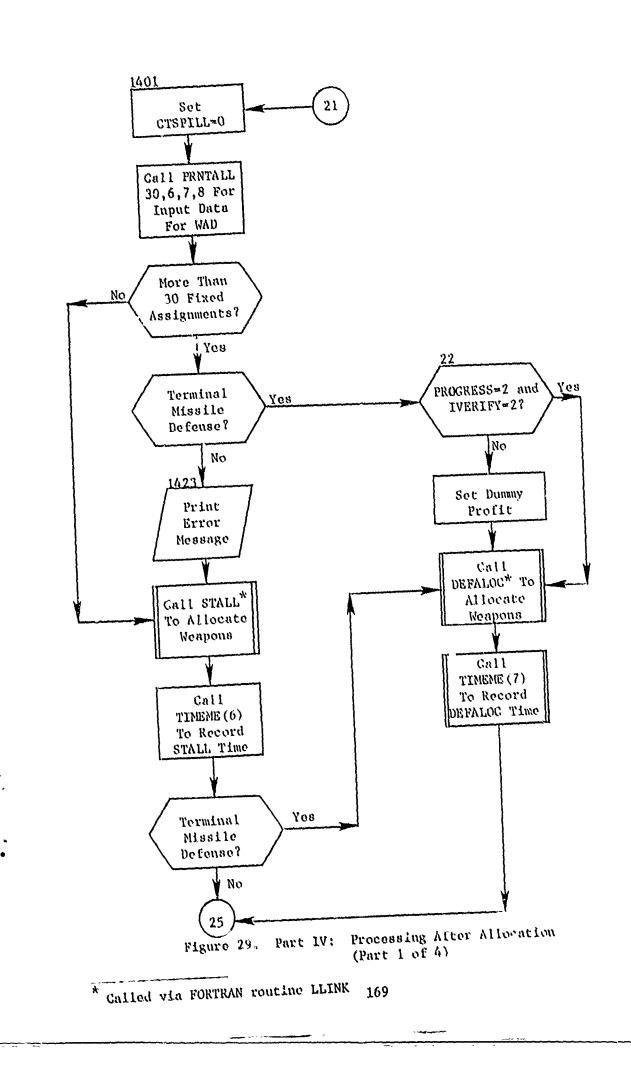
Figure 29, Part III: (Part 3 of 3)

Figure 29. Part IV: Processing After Allocation
(Part 1 of 4)

* Called via FORTRAN routine LLINK   169

```
                          ( 25 )
                             │
                             ▼
                  ┌────────────────────┐
                  │   For Weapons      │
                  │ Assigned, Record   │
                  │     Data In        │
                  │  DYNAMI Arrays     │
                  └────────────────────┘
                             │
                             ▼
                  ╔════════════════════╗
                  ║  Call BOMPRM To    ║
                  ║   Update ASM       ║
                  ║  Fraction Array    ║
                  ╚════════════════════╝
                             │
                             ▼
                  ┌────────────────────┐
                  │   Record Target    │
                  │    Weight In       │
                  │  DYNAMIC Array,    │
                  │  Augment WTSUM     │
                  └────────────────────┘
                             │
                             ▼
                ┌──────────────────────────┐
                │   For Each Nonfixed       │
                │ Weapon Assigned, Aug-     │
                │ ment NALL and RNALL And   │
                │ Associated Entries In     │
                │  RUNSUM and CURSUM        │
                │       Arrays              │
                └──────────────────────────┘
                             │
                             ▼
                  ┌────────────────────┐
                  │    Record And      │
                  │   Accumulate       │
                  │  Payoff, Cost,     │
                  │  And Profit Data   │
                  └────────────────────┘
                             │
                             ▼
                          (  B  )
```

Figure 29.   Part IV: (Part 2 of 4)

170

Figure 29. Part IV: (Part 3 of 4)

171

Figure 29. Part IV: (Part 4 of 4)
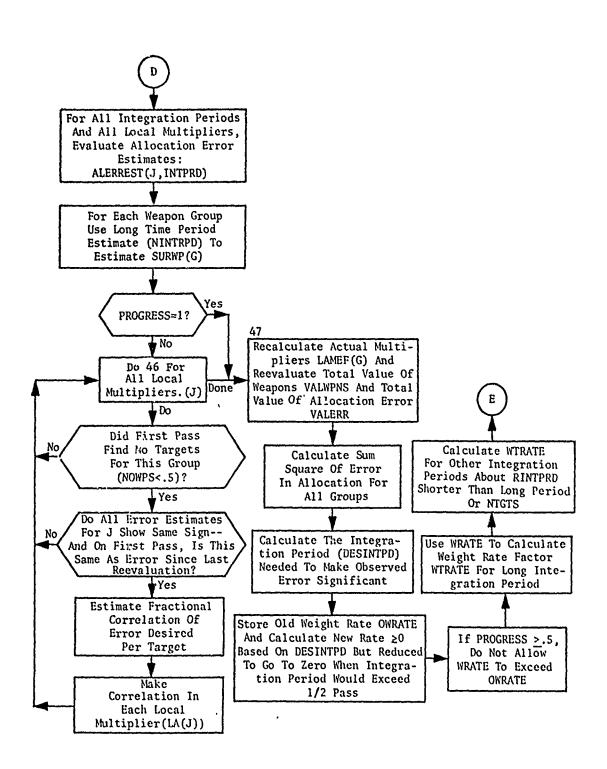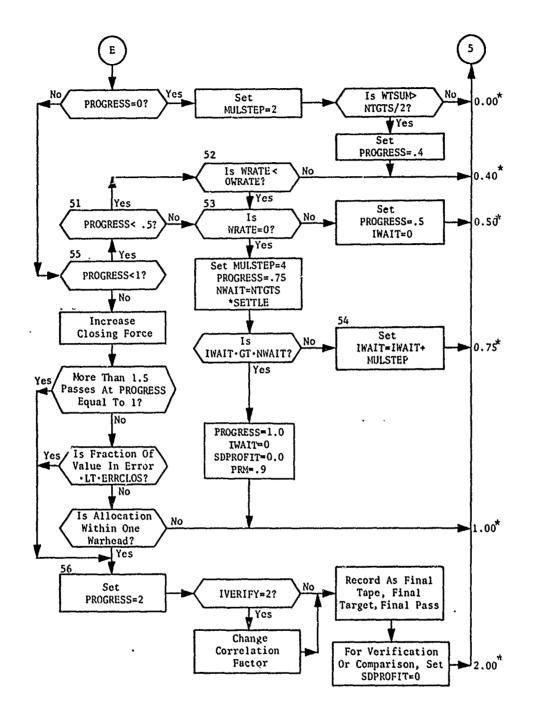
Figure 29. Part V: Multiplier Adjustment and
Progress Evaluation (Part 1 of 2)

173

3.8.1 Subroutine ADDSAL

PURPOSE: This routine updates the stockpile for salvoed weapons

ENTRY POINTS: ADDSAL

FORMAL PARAMETERS:
IGP   - group number
IOPT   - option code
NUR   - index to ISAL array
ISALIN - salvo number

COMMON BLOCKS: DYNAMI, MULTIP, SALVO

SUBROUTINES CALLED: None

CALLED BY: MULCON, STALL, WAD, DEFALOC

Method:

If the weapon is a nonsalvoed weapon but a bomber, ISAL is set to indicate whether weapon is a gravity bomb or ASM. Otherwise the variable IDIFF is set depending upon the option. From this the number in NSALAL (packed 4 per word) is either incremented or decremented.

Subroutine ADDSAL is illustrated in figure 30.
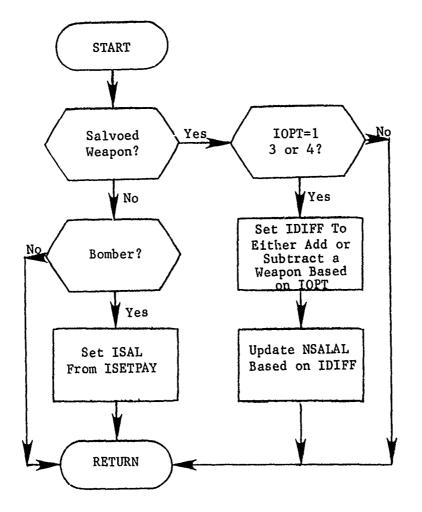
Figure 30. Subroutine ADDSAL

3.8.2 Subroutine ASGOUT
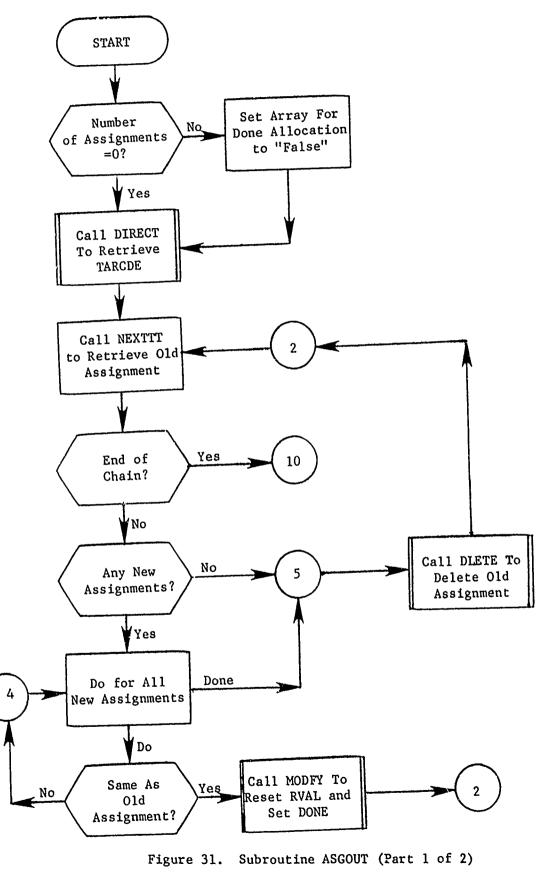
PURPOSE:	To update allocation assignment records in the integrated data base
ENTRY POINTS:	ASGOUT
FORMAL PARAMETERS:	None
COMMON BLOCKS:	C10, C30, DYNAMI, MULTIP, SPLITS, PAYSAV, TARREF, WEPSAV
SUBROUTINES CALLED:	DIRECT, DLETE, MODFY, MYAPOS, NEXTTT, STORE
CALLED BY:	MULCON

Method:

First a logical switch is set for each new weapon assignment to indicate it is unassigned. Next each old assignment is compared to the new assignments to see if all values, save RVAL, of the old assignment are equal to a new assignment. If it does, the RVAL attribute is deleted. Finally, an ASSIGN record is created for all the new assignments for which there is no match.

Subroutine ASGOUT is illustrated in figure 31.

Figure 31. Subroutine ASGOUT (Part 1 of 2)
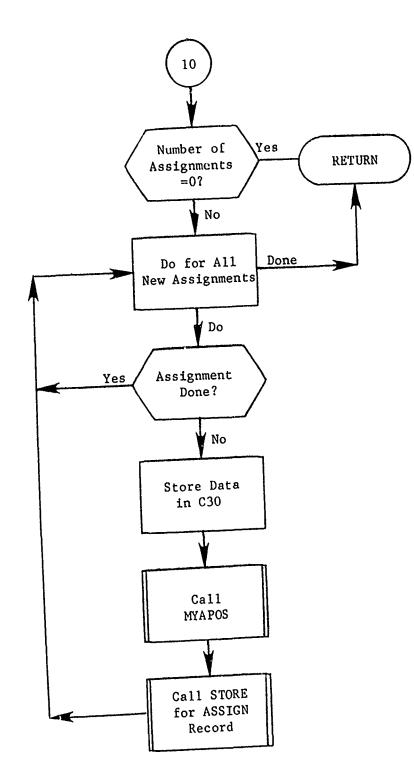
Figure 31. (Part 2 of 2)

3.8.3 Subroutine BOMPRM

PURPOSE: The purpose of this routine is to maintain the array containing the fraction of weapons all allocated from each group which are ASMs.

ENTRY POINT: BOMPRM

FORMAL PARAMETERS: IDIFF = -1 if weapons are being deleted
           +1 if weapons are being added

COMMON BLOCKS: C33, DYNAMI, MULTIP, NALLY, PAYSAV, SALVO, WEPSAV

SUBROUTINES CALLED: None

CALLED BY: MULCON, SCNDGD

Method:

This routine merely updates the ASM fraction array FASM in common block /SALVO/. The important local variables are:

  LXDONE(1) = a logical array set true if a weapon has already been processed to update FASM

  TOTW   = total number of weapons allocated from a group on the target

  TASM   = number of ASMs from a group allocated on the target

FASM(G) is the fraction of currently allocated weapons from group G which are ASMs.

The factor FASM is updated whenever the state of the allocation changes. These changes occur when allocations from a previous pass are removed and when the allocation from the present pass is output. Thus, BOMPRM is called from subroutine MULCON on each target, and by subroutine SCNDGD for each target after the first pass.

Subroutine BOMPRM has one formal parameter IDIFF. If weapons are being removed (previous pass's allocation), then the value of IDIFF is -1. If weapons are being added, then IDIFF is equal to +1. In subroutine SCNDGD, the call to BOMPRM with IDIFF equal to -1 is made after reading the last pass allocation, just prior to the update of the running allocation sums. The call from MULCON with IDIFF equal to +1 is made just prior to the running sum update.

Upon entry to subroutine BOMPRM, the routine checks variable NBLN in /C33/. If this variable is negative, the allocation in /DYNAMI/ was made by subroutine DEFALOC and contains no bomber weapons. In this case, the subroutine returns with no further processing. If the

180

allocation was made by subroutine STALL, the IG array of /DYNAMI/ is checked. For each allocation in this array, the variable KORRX of /DYNAMI/ is checked to determine if the weapon is a bomber. If not, the next entry in the IG array is checked. If the weapon is a bomber, then the value of GSEASM for the group is checked (in /WEPSAV/). If GSEASM is equal to zero or one, then FASM is set to GSEASM and processing continues with the next entry in the IG array. Otherwise, FASM is updated for the group.

The ISAL array of /DYNAMI/ contains the indicator of bomb or ASM allocation (for bomber groups only. This array is defined differently for missile groups). If the value is zero, a gravity bomb was allocated. A value of one signifies the use of an ASM.

The total number of weapons from the group which are currently allocated is kept in array RNALL of common /NALLY/. This array is updated twice for each target, just following the call on BOMPRM.

Using these variables, the value of FASM is updated as follows:
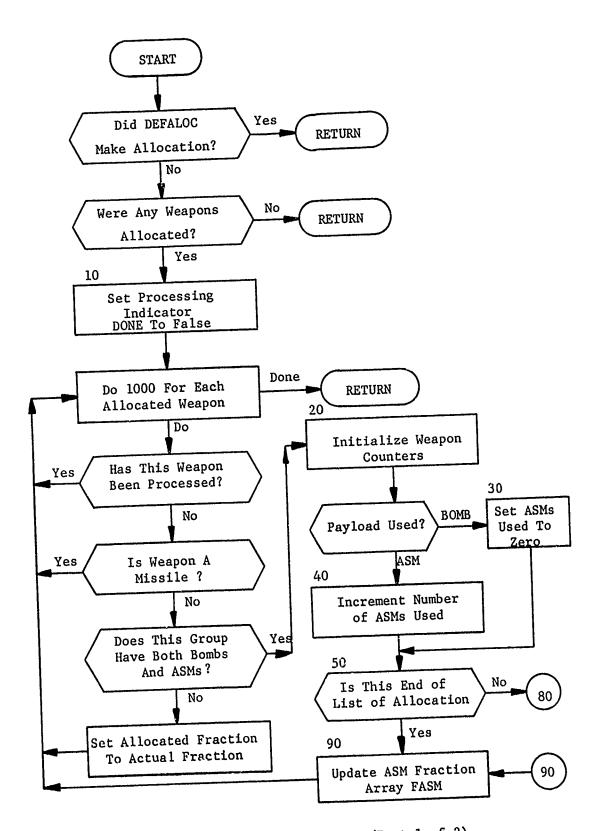
Define: TASM = number of ASMs allocated from group G (as determined from the ISAL array) on current target

TOTW = number of weapons allocated from group G on current target

Then,

$$FASM_{new} = \frac{(FASM_{old} * RNALL) + (TASM * IDIFF * CTMULT)}{RNALL + (TOTW * IDIFF * CTMULT)}$$

Note that the variables FASM and RNALL in the above equation are arrays indexed by the group number G. CTMULT, from common block /MULTIP/ is the current target multiplicity.

Figure 32 displays the logic of subroutine BOMPRM.
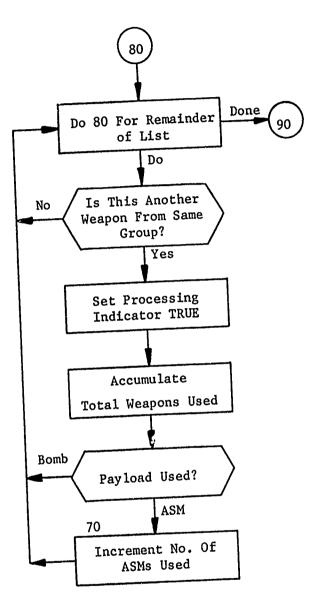
181

Figure 32. Subroutine BOMPRM (Part 1 of 2)

182

Figure 32. (Part 2 of 2)
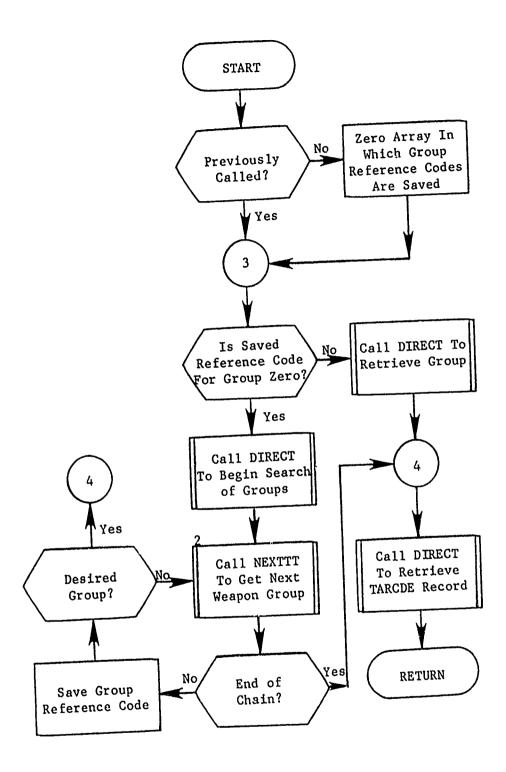
3.8.4 Subroutine MYAPOS

PURPOSE: To position records properly before storage of a
new ASSIGN record.

ENTRY POINTS: MYAPOS

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C30, GRPHDR, TARREF

SUBROUTINES CALLED: DIRECT, NEXTTT

CALLED BY: ASGOUT

Method:

On first call the array of group IDS reference codes is set to zero.
From then on, with each call the saved reference code of the desired
group is checked. If it is nonzero it is retrieved. If it is zero,
the group chain is cycled up to the desired group, the intervening
groups also have their reference codes saved. Finally, when the proper
group record has been retrieved, the target record is retrieved.

Subroutine MYAPOS is illustrated by figure 33.

184

START

Previously Called? —No→ Zero Array In Which Group Reference Codes Are Saved

Yes ↓

3

Is Saved Reference Code For Group Zero? —No→ Call DIRECT To Retrieve Group

Yes ↓

Call DIRECT To Begin Search of Groups

4

↓

Desired Group? —No→ Call NEXTTT To Get Next Weapon Group

Yes → 4

Save Group Reference Code ←No— End of Chain? —Yes→

Call DIRECT To Retrieve TARCDE Record

RETURN

Figure 33. Subroutine MYAPOS

185

3.8.5 Subroutine PRNTALL

PURPOSE: This routine provides a way of calling the print subroutine PRNTNOW that is conditional on the print control flags set by PRNTCON.

ENTRY POINTS: PRNTALL

FORMAL PARAMETERS: IOPT - Print option number

COMMON BLOCKS: C30, CONTRO, PRNTCN

SUBROUTINES CALLED: PRNTNOW, TIMEME

CALLED BY: MULCON, WAD, WADOUT, FRSTGD, RESVAL, DEFALOC, SETPAY

Method:
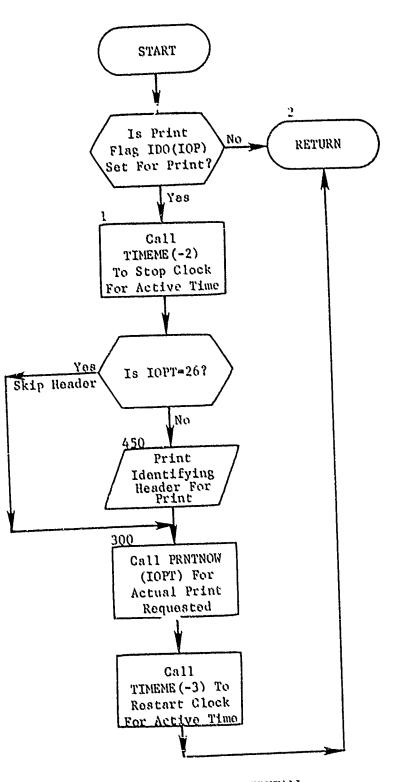
To provide convenient control over prints in program ALOC almost all print statements are contained in subroutine PRNTNOW. They are activated by calling PRNTNOW(IOPT) for the appropriate print option IOPT. If it is desired to place the print under data-input control so that the print will not appear unless a specific print request is included in the data deck, this can be accomplished by calling PRNTNOW via a call on PRNTALL. PRNTALL executes the request on PRNTNOW only if the print control subroutine PRNTCON has set the corresponding print control flag IDO(IOPT) active (i.e., $= 3$).

For each call PRNTALL first checks to see if the print has been set active by PRNTCON. If not, it immediately RETURNs (statement 2) to minimize time wasted on inoperative print calls.

If the particular print is active, PRNTALL immediately calls TIMEME (statement 1) to stop the clock which records active time in the program. This makes it possible to do a test run with an unusual number of prints and still obtain a valid estimate of what the running time would be without such prints. After the call on PRNTNOW, PRNTALL reactivates the clock before returning to the main program.

Before each print option (except 26), PRNTALL prints a heading identifying the optional print.

Subroutine PRNTALL is illustrated in figure 34.

START

Is Print Flag IDO(IOP) Set For Print? — No → RETURN

Yes

Call TIMEME(-2) To Stop Clock For Active Time

Is IOPT=26?

Yes — Skip Header

No

450

Print Identifying Header For Print

300

Call PRNTNOW (IOPT) For Actual Print Requested

Call TIMEME(-3) To Restart Clock For Active Time

Figure 34. Subroutine PRNTALL

187

3.8.6 Subroutine PRNTCON

PURPOSE:	This routine sets the print control flags which determine whether a given print request made through PRNTALL will be executed.
ENTRY POINTS:	PRNTCON
FORMAL PARAMETERS:	None
COMMON BLOCKS:	C30, CONTRO, PRNTCN
SUBROUTINES CALLED:	None
CALLED BY:	MULCON

Method:
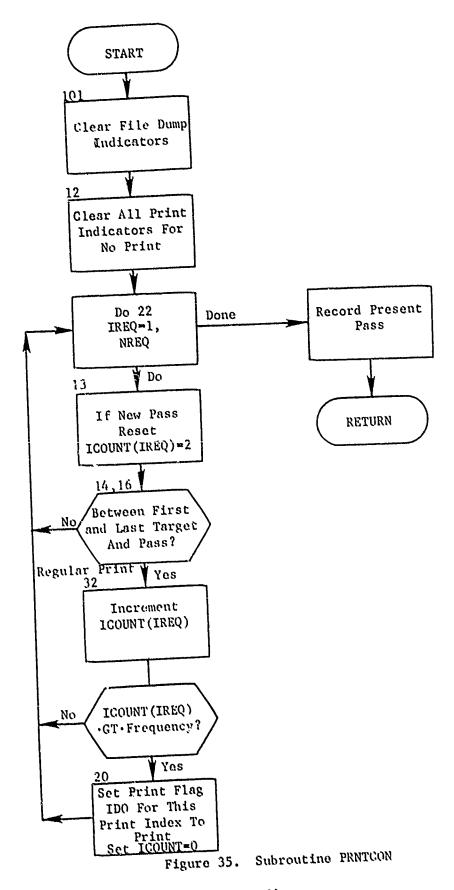
The input arrays for the print requests were read in subroutine RDALCRD. These arrays are:

INDEXPR	The index to the print requested
JPASS	The first pass (value of NPASS in /CONTRO/) on which the request is to operate
LPASS	The last pass on which the request is to operate
JTGT	The first target (value of TGTNUMB in /C30/) on which the request is to operate on each pass
LTGT	The last target on which the request is to operate
KTGTFREQ	The frequency with which the print is to operate (e.g., KTGTFREQ = 5 implies every fifth target).

PRNTCON is called by MULCON before proceeding to process each new target. PRNTCON first reinitializes all print control flags to a nonprint state (IDO = 1). It then examines the list of print requests to see if any are operative for this target on this pass. For each operative print the flags are set to print (IDO = 3, IFTPRNT = KTGTFREQ).

This arrangement makes it possible to request the same print at different targets or passes with separate independent print requests. If regular prints are requested with KTGTFREQ greater than 1, the first print will not occur at JTGT but at KTGTFREQ -1 targets later, and thereafter the print will occur every (KTGTFREQ)th target.

Figure 35 illustrates subroutine PRNTCON.

Figure 35. Subroutine PRNTCON

189

3.8.7 Subroutine PRNTNOW
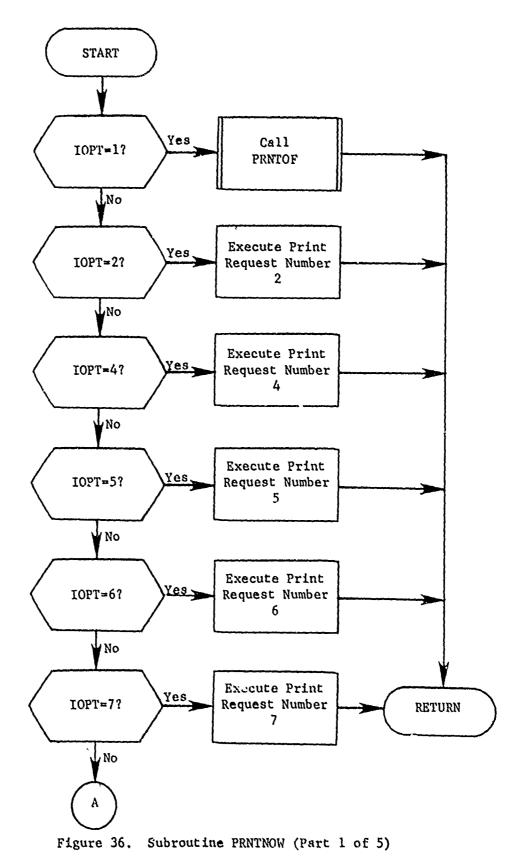
PURPOSE: To produce optional printed output.

ENTRY POINTS: PRNTNOW

FORMAL PARAMETERS: IOPT - Print option number

COMMON BLOCKS: ALERUN, C30, C33, DYNAMI, MULTIP, NALLY, DAYSAV,
                        PREMS, PRTMUL, SALVO, SURPW, TGTSAV, IACB, PAYOFF,
                        WADFIN, WADOTX, WADWPN, WEPSAV, WPFIX, WTS

SUBROUTINES CALLED: ABORT, PRNTOD, PRNTOF, PRNTOS, TIMEME

CALLED BY: PRNTALL, MULCON

Method·
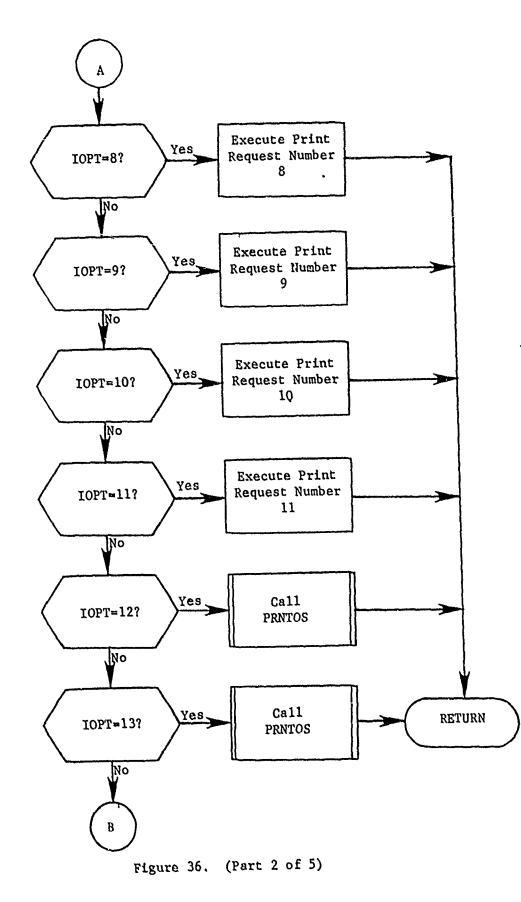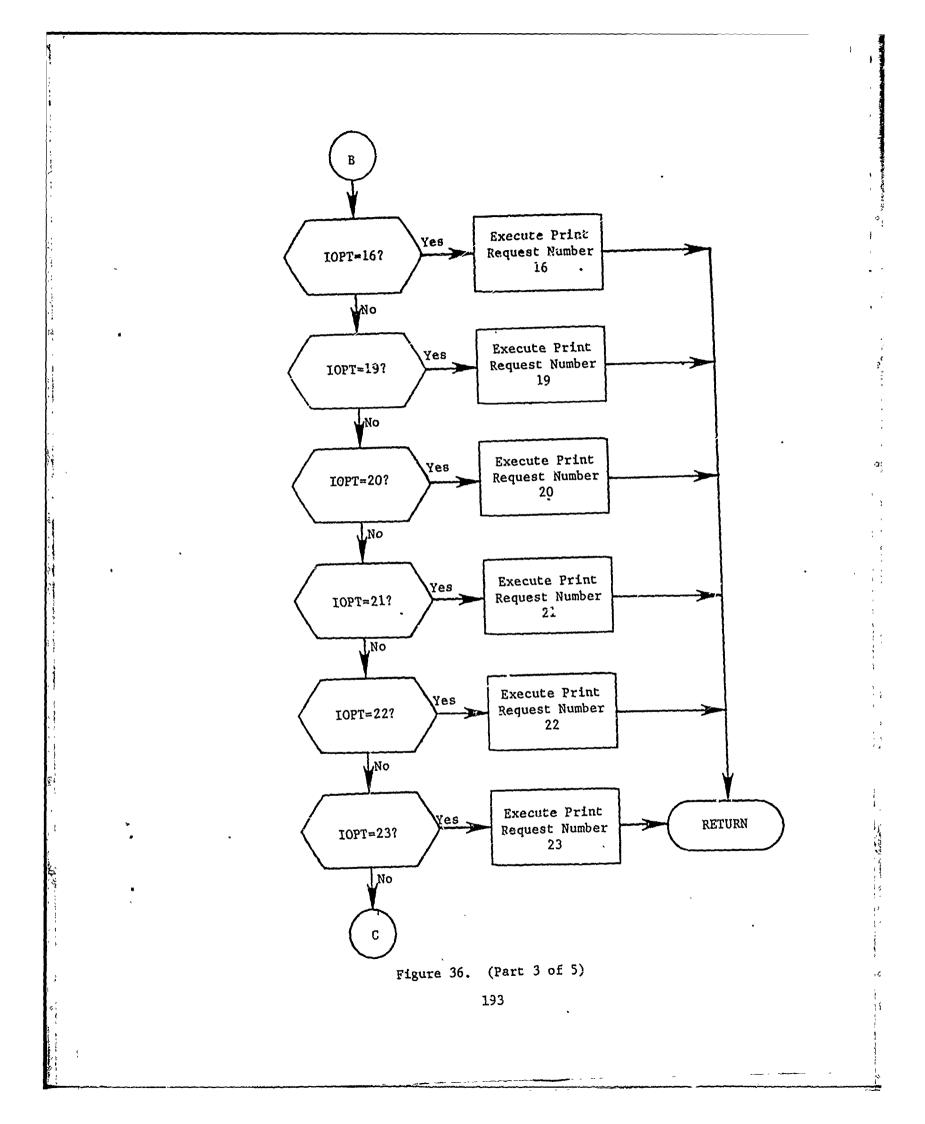
The formal parameter IOPT determines which print is produced. The re-
sult of the alternatives appear in the Users Manual, UM 9-77, Volume III.
Options 1, 12, 13, 26, 27, and 28 require a subroutine be called which
contains the print function.
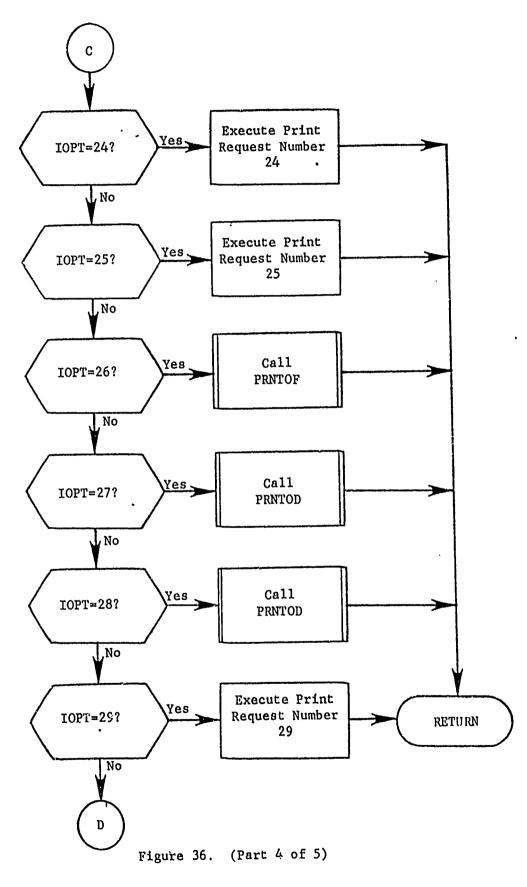
Subroutine PRNTNOW is illustrated in figure 36.

Figure 36. Subroutine PRNTNOW (Part 1 of 5)

191

Figure 36. (Part 2 of 5)

Figure 36. (Part 3 of 5)

193

Figure 36. (Part 4 of 5)
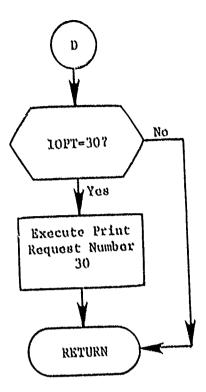
Figure 36. (Part 5 of 5)

3.8.8 Function TABLEMUP

PURPOSE: This function calculates weapon-target kill fac-
                        tors for either the exponential or square root
                        damage laws as a function of an input single shot
                        survival probability.

ENTRY POINTS: TABLEMUP

FORMAL PARAMETERS: S -- a single shot survival probability

COMMON BLOCKS: TABLE, WADWPN

SUBROUTINES CALLED: None

CALLED BY: RESVAL, RECON
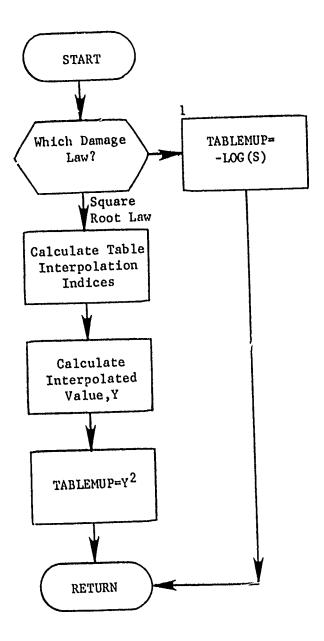
Method:

This function computes weapon kill factors by computation for the expo-
nential damage law and by a table lookup for the square root law. The
function uses the square root law only if the option is selected by the
user and the target has a radius greater than 0.

If the square root law is used on a target, the variable ILAW in common
/WADWPN/ is set to 100 by subroutine RECON. This variable is checked by
TABLEMUP to determine which damage law is used. The exponential law use
causes ILAW to be set to 0.

The input formal parameter is a single shot survival probability, S. If
the exponential damage law is selected, the function returns the value
-LOG(S). If the square root law is selected, the function performs a
table lookup technique on the array TABLE in common /TABLE/. This array
was preset by subroutine SETABLE. The function performs linear interpo-
lation between the entries of the table. It returns the square of the
interpolated value. (The method of determining the kill factors used in
subroutine SETABLE is too slow for use in TABLEMUP.)

Function TABLEMUP is illustrated in figure 37.

Figure 37. Function TABLEMUP

197
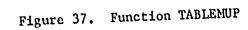
3.9 Subroutine FRSTGD[*]

PURPOSE: Assemble allocation data on the first pass.

ENTRY POINTS: FRSTGD

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C15, C30, C33, DYNAMI, FIL21, FILL, FIRST,
                         GRPHDR, GRFSTF, INITSW, MULTIP, PAYSAV, TARREF,
                         TGTSAV, WADWPN, WEPSAV, WPFIX, XFPX

SUBROUTINES CALLED: CRDCAL, DIRECT, FLGCHK, HDFND, HEAD, INICRD, MODFY,
                         NEXTTT, NXSPLT, PKCALC, PRNTAL, RECON, RETRV,
                         TGTCRD, TIMEME

CALLED BY: MULCON

Method:
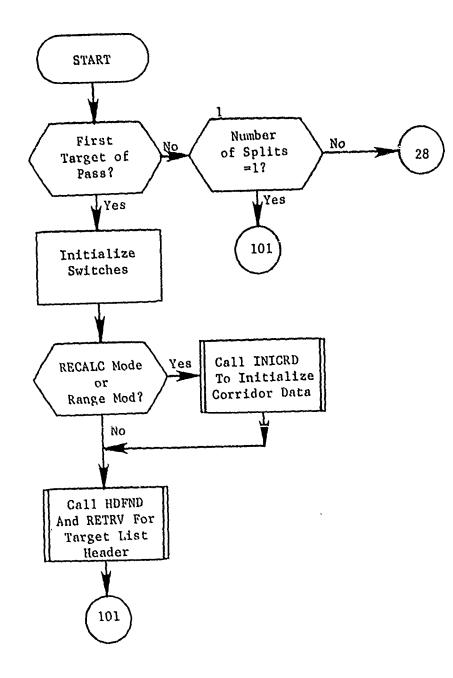
This routine processes each target in target number order. Each call
causes the next target to be retrieved. Since each record on the tar-
get list points to either a target or a complex record, the next step
in the process is to retrieve the remainder of the target data. Next
the weapon data is acquired. This process depends to a great extent
upon whether the user has saved file 15 from a previous run of ALOC.
If so, this file is read in and unpacked. If not, the file is created
by cycling through the weapon groups and calculating the various needed
quantities. Much of the group data needed for this process is contained
on file 25 where it was stored by DATGRP. If the user has specified
range modifications, any information which differs from that on file
15 is written on file 22 in the same format.

During this process, the INACTIVE array is set. This array has an entry
for each group and is either set to 0 or 100. 0 implies that the group
is available for allocation to the target. 100 indicates that the group
is unavailable for one of several reasons: target out of range, time
decay requirements, and flag location and MIRV restriction. This array
is written onto unit 21.
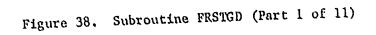
The final step is to read in any fixed assignments to the target and up-
date the assignment records.

Subroutine FRSTGD is illustrated in figure 38.
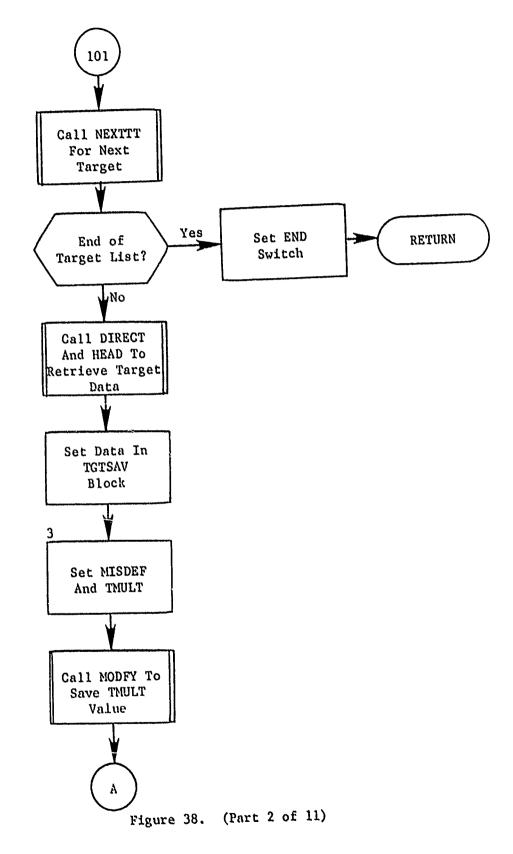
---

[*] First subroutine of segment FGD.
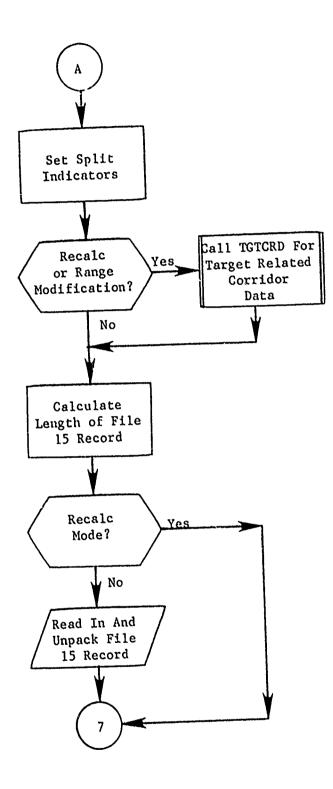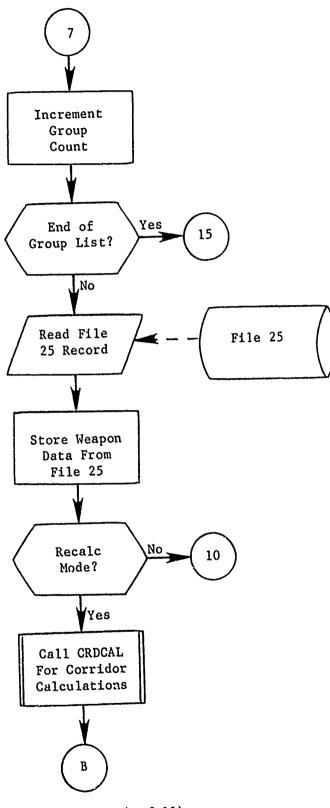
198

Figure 38. Subroutine FRSTGD (Part 1 of 11)

Figure 38. (Part 2 of 11)

Figure 38. (Part 3 of 11)

Figure 38. (Part 4 of 11)

Figure 38.   (Part 5 of 11)

Figure 38. (Part 6 of 11)

Figure 38.   (Part 7 of 11)

Figure 38. (Part 8 of 11)

Figure 38. (Part 9 of 11)

Figure 38. (Part 10 of 11)
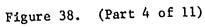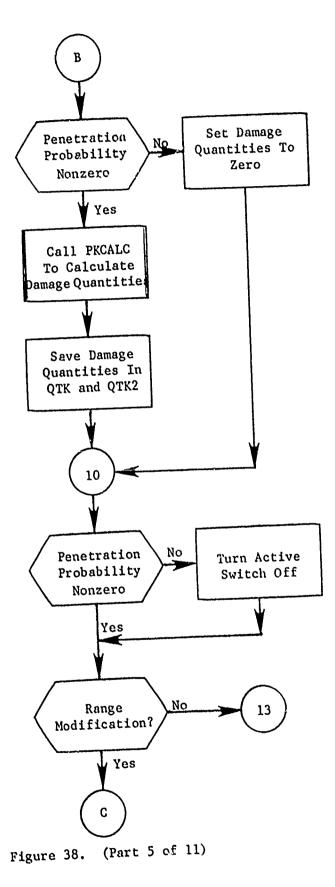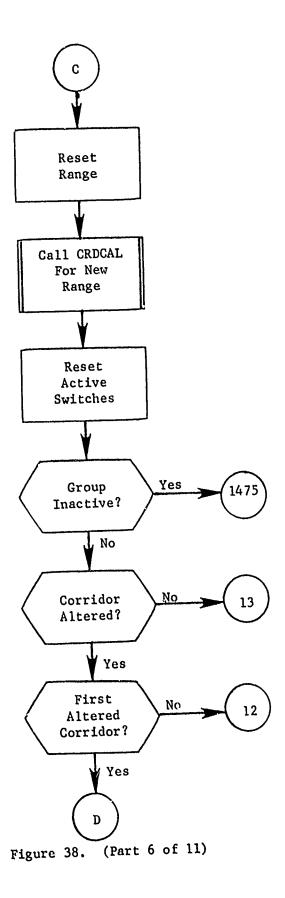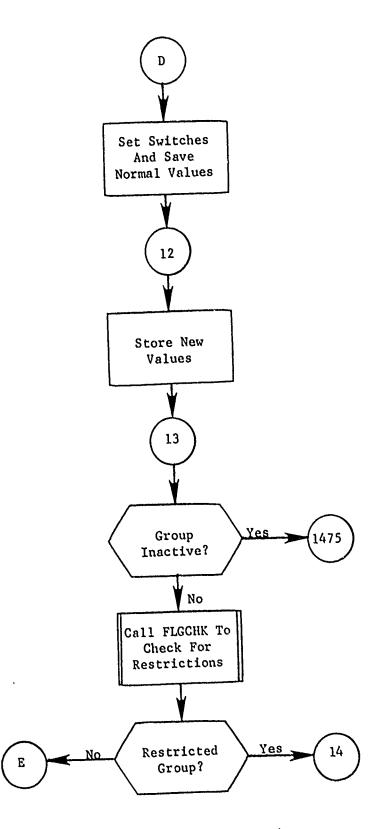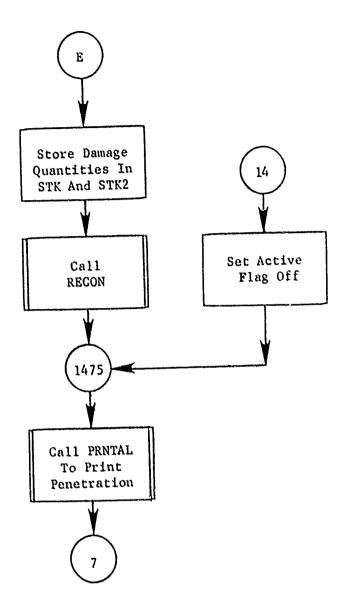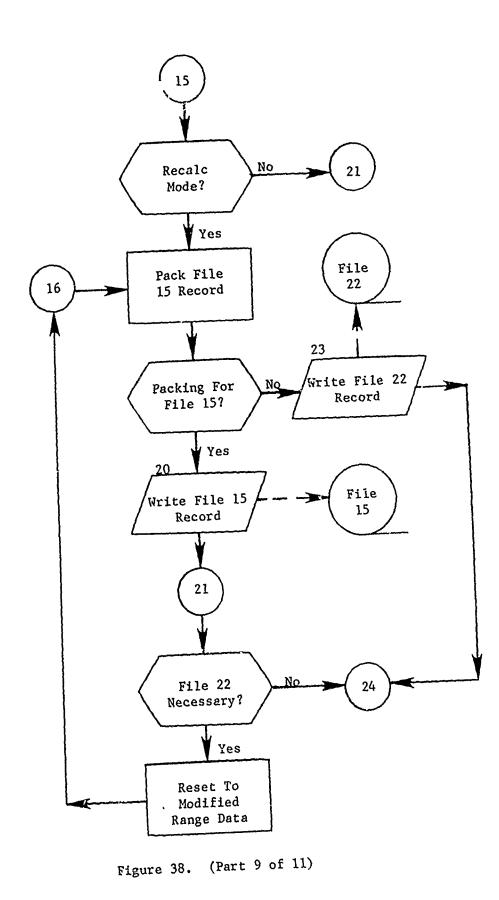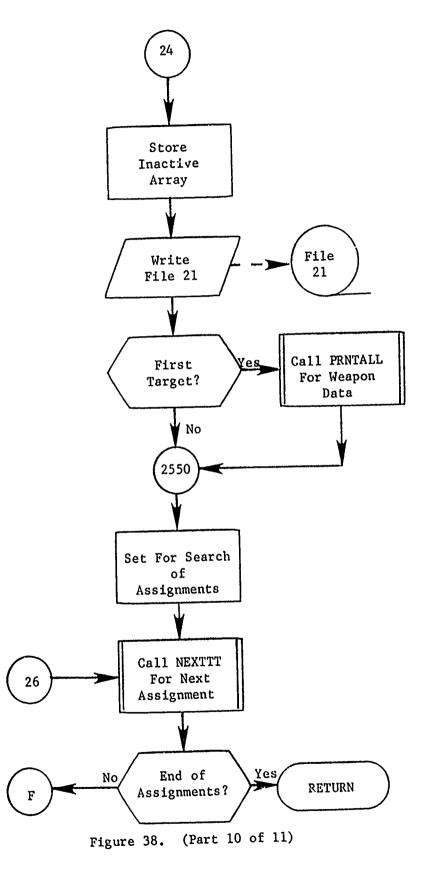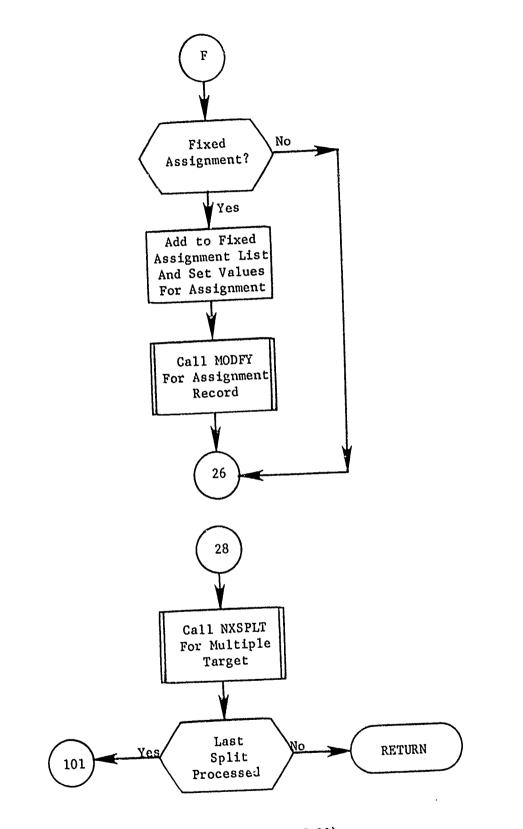
Figure 38. (Part 11 of 11)

3.9.1 Subroutine CRDCAL

PURPOSE: Calculate penetration probability and pick bomber
                     corridor

ENTRY POINTS: CRDCAL

FORMAL PARAMETERS: JORR   - Corridor picked
                     XEN   - Penetration probability
                     ZOA   - Time of arrival
                     ICLSS - Class index of group

COMMON BLOCKS: C30, CORSTF, PAYSAV, REFPNT, TGTSAV, WADWPN,
                     WEPSAV, XFPX

SUBROUTINES CALLED: DISTF, EXP, TOFM

CALLED BY: FRSTGD

Method:

First a check is made for match-ups of naval weapons to naval targets.
Next, if the group is a missile, its corridor is set to zero and its
penetration probability and time of arrival are calculated.

For bomber groups, first the PKNAV and IPENMO are checked to see if
group is restricted to corridors 2 and 1 respectively. Otherwise, each
corridor in turn is examined to see if it would provide the best route
to the target. Corridors which would make the distance too great are
rejected. For the remainder, the penetration probability is calculated
with the bomber using low altitude flight as much as possible. The
highest penetration corridor is chosen.

Subroutine CRDCAL is illustrated in figure 39.
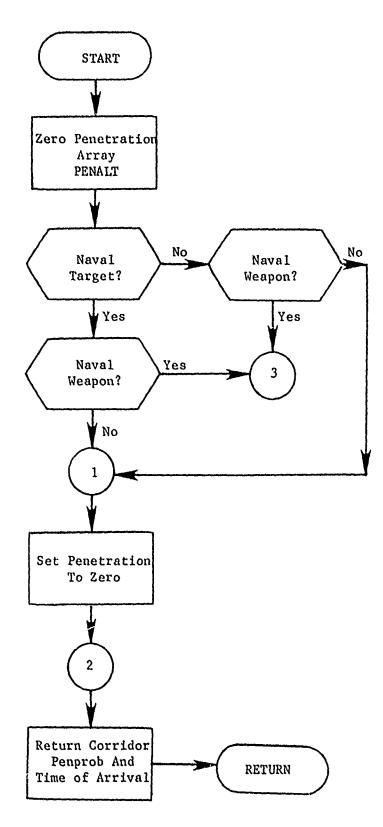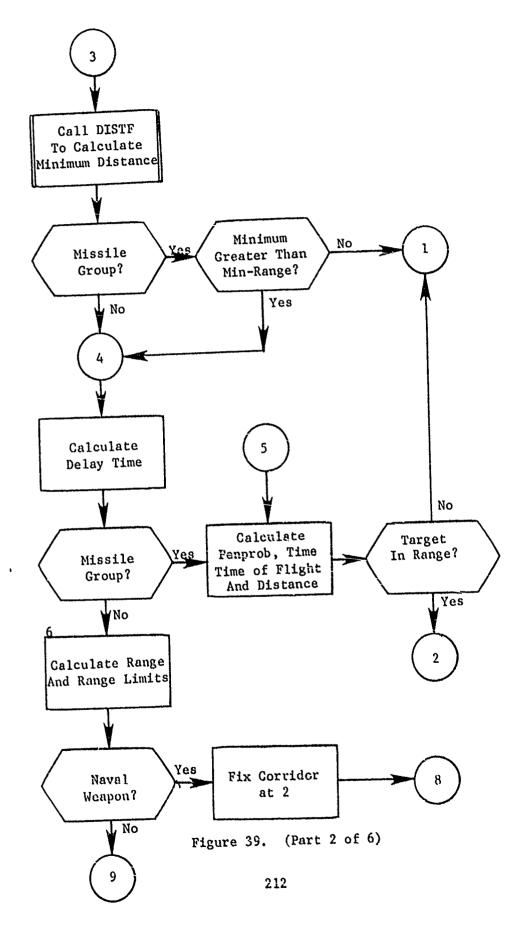
Figure 39. Subroutine CRDCAL (Part 1 of 6)

211

Figure 39.   (Part 2 of 6)

212
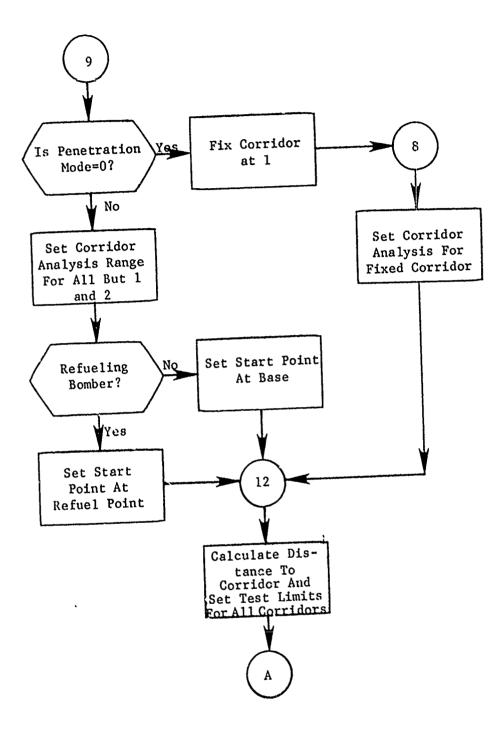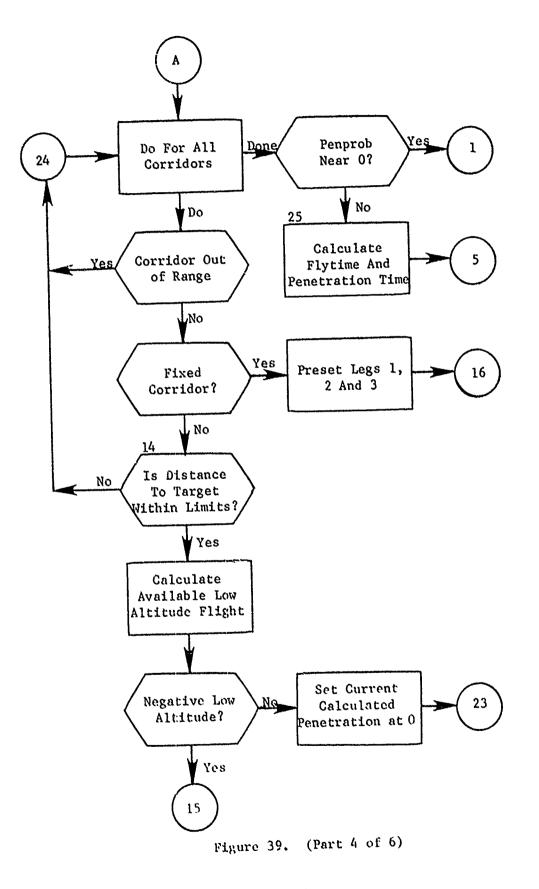
Figure 39. (Part 3 of 6)

Figure 39. (Part 4 of 6)
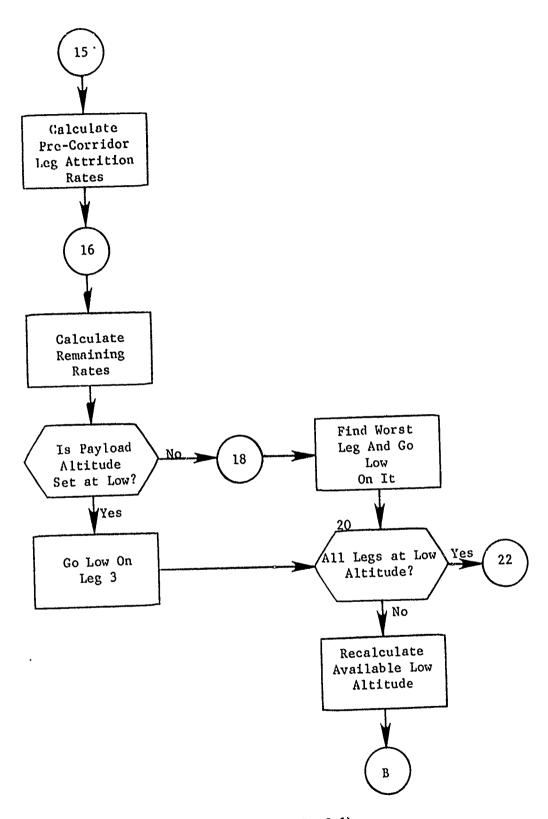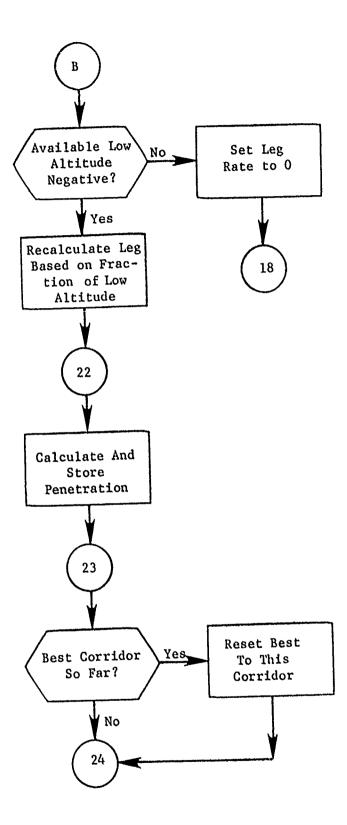
Figure 39. (Part 5 of 6)

215

Figure 39.   (Part 6 of 6)

3.9.2 Subroutine FLGCHK

PURPOSE: To check flag, location and MIRV restrictions.

ENTRY POINTS: FLGCHK

FORMAL PARAMETERS: FIND - True if group not restricted
            False if group restricted

COMMON BLOCKS: C30, CNCLS, GRPSTF, INITSW, MULTIP, TGTSAV

SUBROUTINES CALLED: None

CALLED BY: FRSTGD
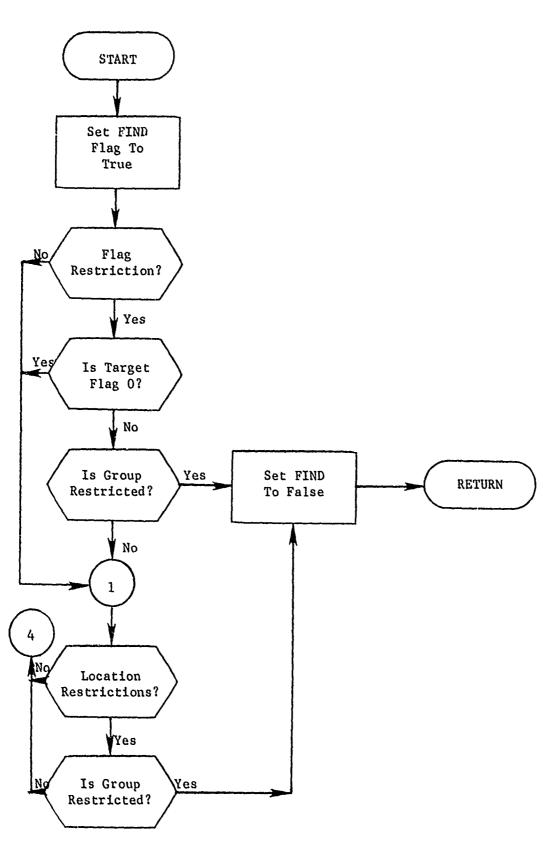
Method:
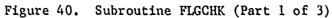
This routine uses the logical arrays stored on file 25 by DATGRP. Each type of restriction is checked for correlation between the group and the current target.

Subroutine FLGCHK is illustrated by figure 40.

Figure 40. Subroutine FLGCHK (Part 1 of 3)

Figure 40. (Part 2 of 3)

219
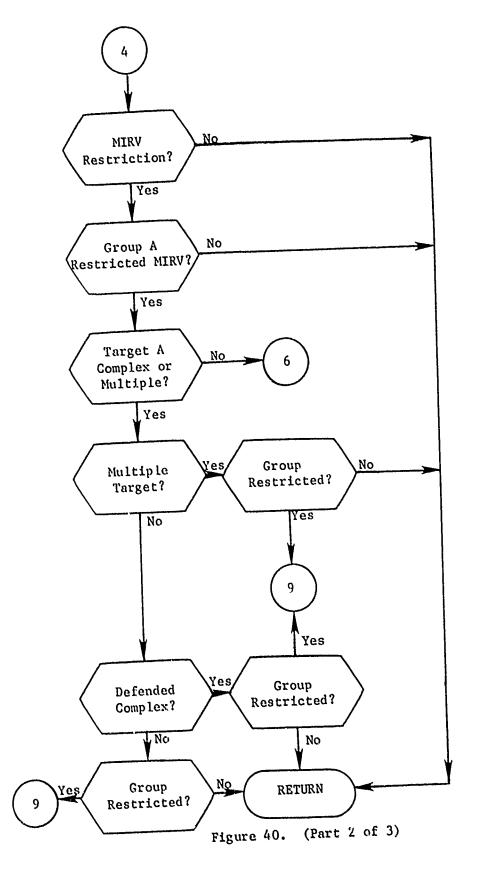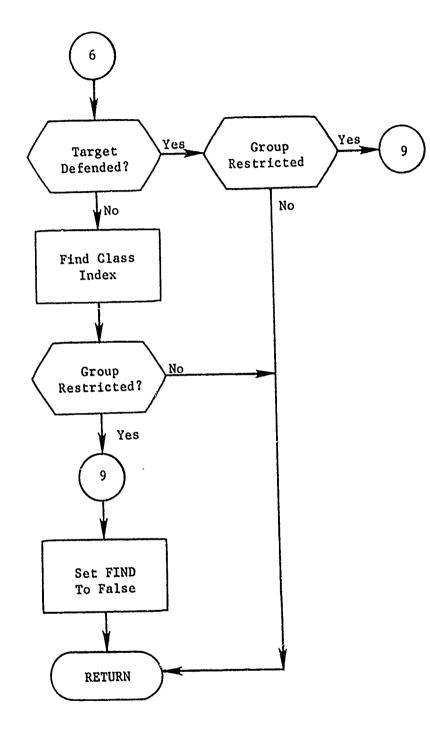
Figure 40. (Part 3 of 3)

3.9.3 Subroutine INICRD

PURPOSE:	To make preliminary geography calculations, first for all corridors then for each target.
ENTRY POINTS:	INICRD, TGTCRD
FORMAL PARAMETERS:	None
COMMON BLOCKS:	C10, C15, C30, CORSTF, REFPNT
SUBROUTINES CALLED:	DELLON, DISTF, HDFND, HEAD, NEXTTT, RETRV, SORT, TIMEME
CALLED BY:	FRSTGD

Method:

Entry INICRD

First each penetration corridor has its data stored in common block CORSTF. As the legs of the corridor are queried, the length is calculated. Also, for each corridor various quantities required for computation of crossdistance are computed.
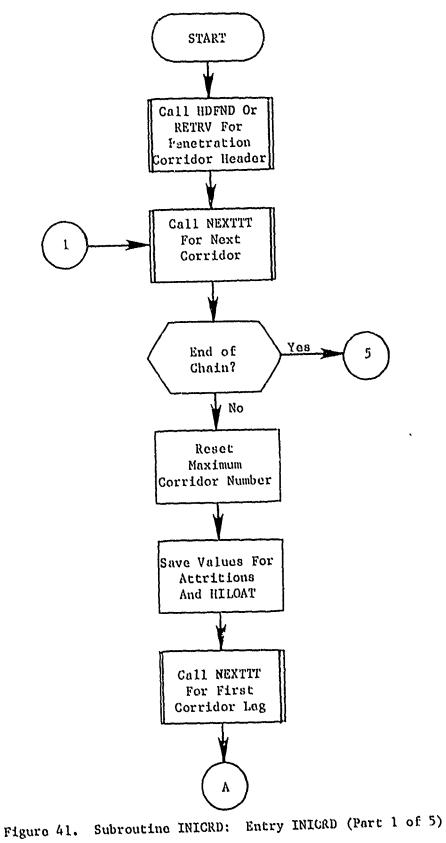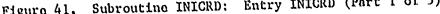
After all penetration corridors are processed, the refuel points are saved in block REFPNT.
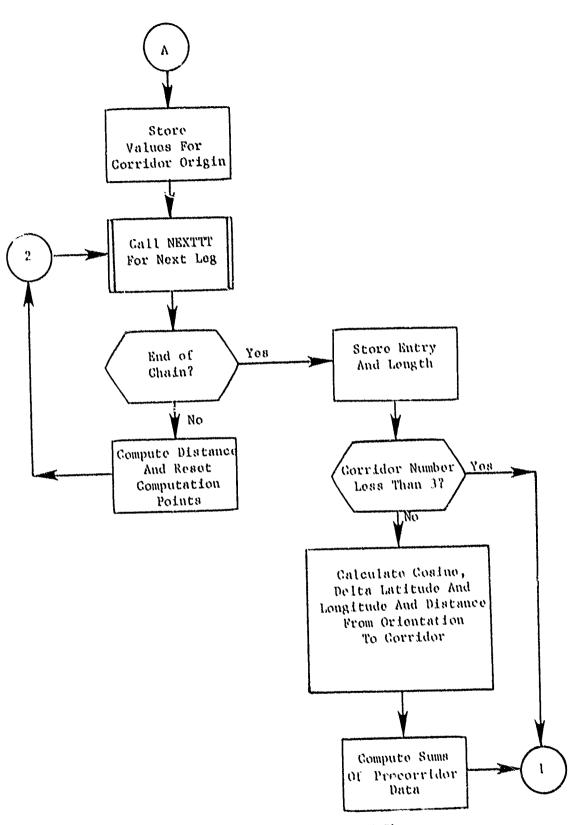
Entry TGTCRD
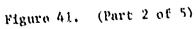
First the chains connecting target to depenetration corridor are accessed to find the shortest recovery distance. Next all penetration corridors are accessed and attrition factors stored and cross distances calculated.
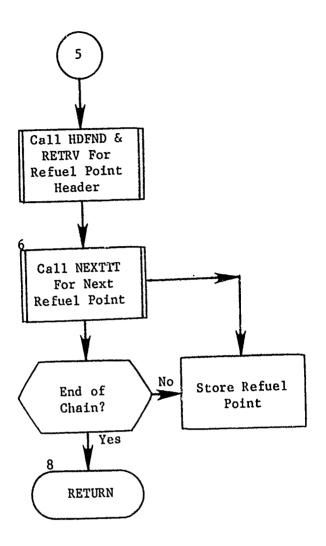
Subroutine INICRD is illustrated in figure 41.

Figure 41. Subroutine INICRD: Entry INICRD (Part 1 of 5)

Figure 41. (Part 2 of 5)

Figure 41.   (Part 3 of 5)

224

Figure 41. Entry TGTCRD (Part 4 of 5)

225
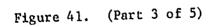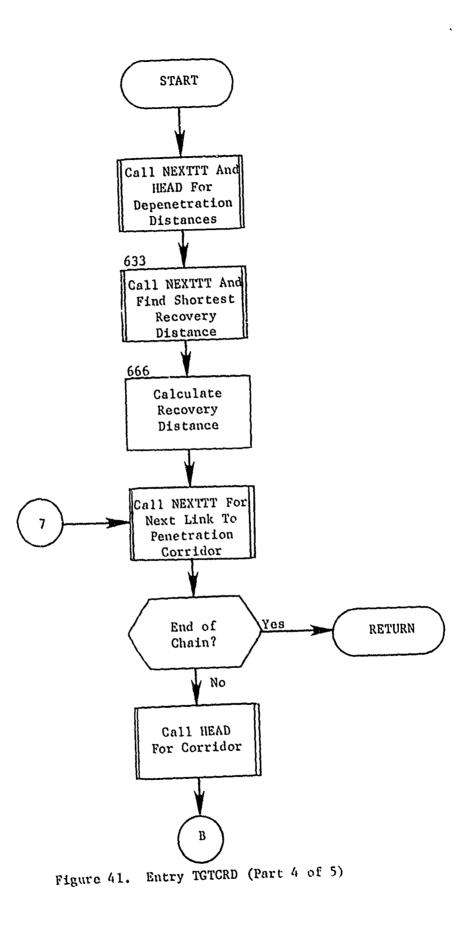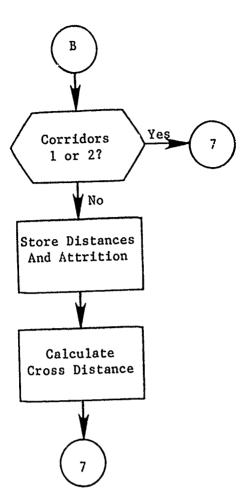
Figure 41.   (Part 5 of 5)

3.9.4 Subroutine NXSPLT

PURPOSE: To process input for split multiple targets.

ENTRY POINTS: FRSPLT, NXSPLT

FORMAL PARAMETERS: IBR - Returned from NXSPLT
        1 = splits active
        2 = splits ended

COMMON BLOCKS: C10, C30, C33, DYNAMI, MULTIP, SPLITS, TARREF

SUBROUTINES CALLED: DIRECT, MODFY, NEXTTT

CALLED BY: FRSTGD, SCNDGD

Method:

When a multiple target is split during the allocation process (see sub-
routine SPLIT). The normal contents of block C33 (TARCDE record) are
replaced by split information and the old contents saved on file 25.
This routine is designed to process input for multiple targets which
have already been split.

Entry NXSPLT
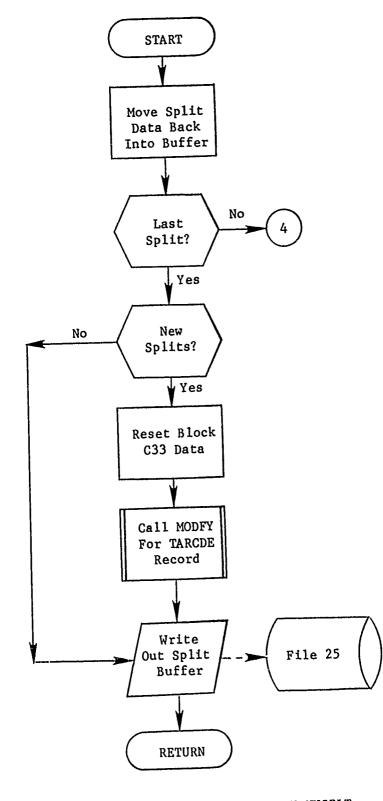
This entry processes the next split (i.e., second or greater) of a multi-
ple target. The first step is to see if all splits have been processed.
If they have, the split information for block C33 is updated if necessary
and IBR set to 2 to indicate the end of the splits to the calling pro-
gram.

Otherwise, the next set of C33 data is obtained from the buffer and
KLMULT is set. Finally, the current assignments (fixed assignments on
pass one) are read in, checking the FLMULT attribute to assure that the
assignment applies to the current split.

Entry FRSPLT

This entry processes the first split of a multiple target split on a pre-
vious pass. The split data is saved from block C33 into block SPLITS.
Then the process passes into the entry NXSPLT code to set KLMULT and
read the current assignments.

Subroutine NXSPLT is illustrated in figure 42.

Figure 42. Subroutine NXSPLT: ENTRY NXSPLT
(Part 1 of 4)
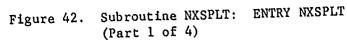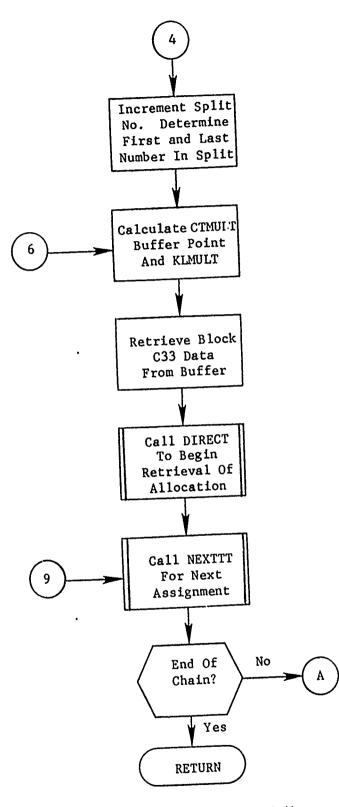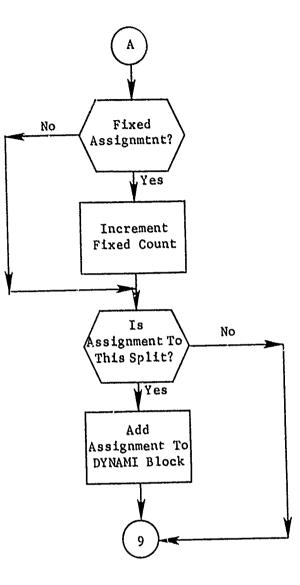
Figure 42. (Part 2 of 4)
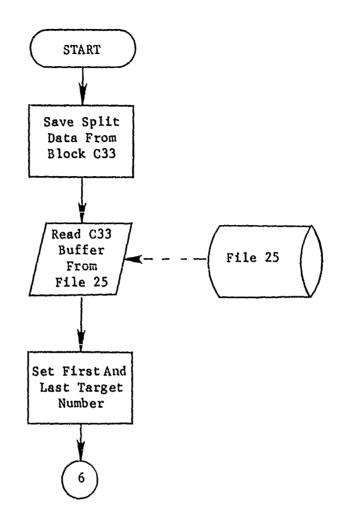
Figure 42. (Part 3 of 4)

Figure 42. ENTRY FRSPLT (Part 4 of 4)
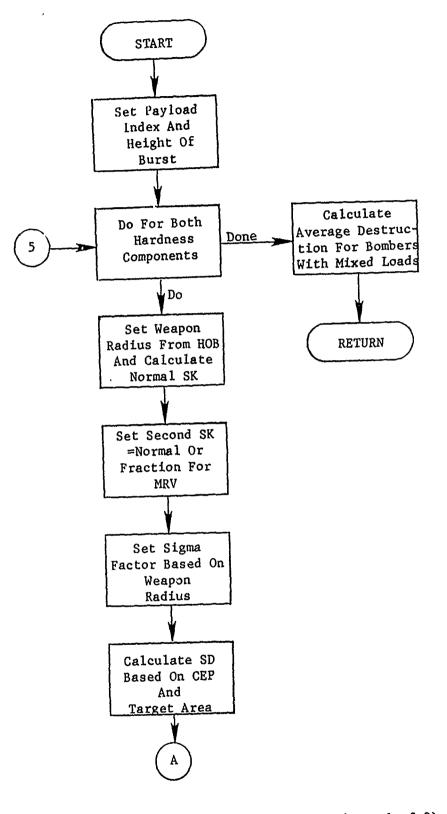
3.9.5 Subroutine PKCALC

PURPOSE:	To calculate kill probabilities
ENTRY POINTS:	PKCALC
FORMAL PARAMETERS:	STA - Array for normal PK
	STB - Array for second PK
	ICLSS - Class index of weapon group
COMMON BLOCKS:	C30, PAYSAV, SALVO, TGTSAV, WEPSAV
SUBROUTINES CALLED:	None
CALLED BY:	FRSTGD

Method:

This routine calculates the kill probabilities (PK) of the weapon group against the target using a standard approximation to the circular coverage function. Two PK's are calculated for each hardness component of the target. The first PK is based upon the yield of the group as input and the group's CEP. The second PK varies according to the class of the weapon. For missiles it is the first PK adjusted by the number of warheads which make up the group yield (for any types but MRV, the first and second PK's are equal). For bombers the second PK reflect the PK of any ASMs onboard. Naval weapon's have their PK's input in the attribute GPKNAV. Average Destruction (AVDE) is calculated for bombers.

Subroutine PKCALC is illustrated in figure 43.
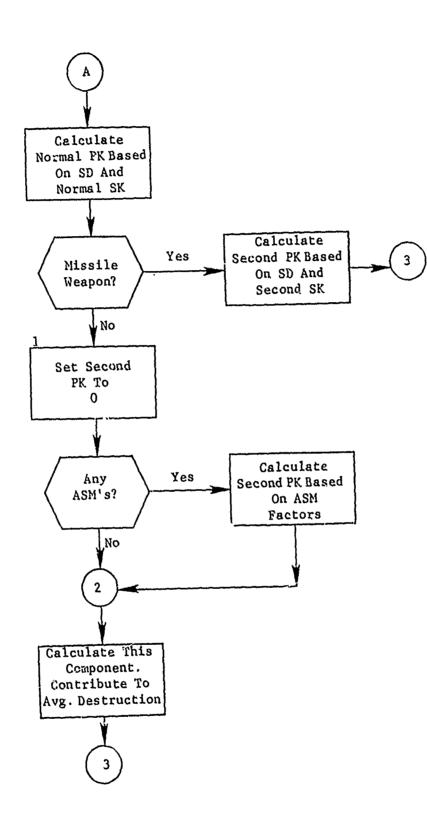
Figure 43.   Subroutine PKCALC (Part 1 of 3)

233

Figure 43. (Part 2 of 3)

234
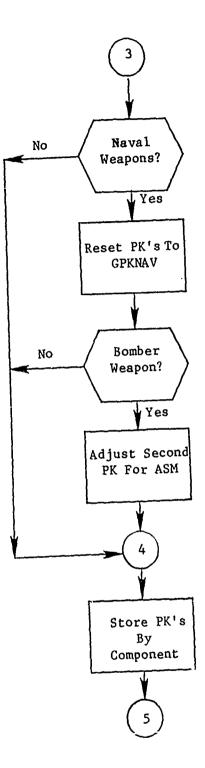
Figure 43. (Part 3 of 3)

3.9.6 Subroutine PRNTOF

PURPOSE: To produce optional prints for overlay FGD (options
                   1 and 26),

ENTRY POINTS: PRNTOF

FORMAL PARAMETERS: IOPT - print option number

COMMON BLOCKS: C30, PAYSAV, SALVO, WADWPN, WEPSAV, WPFIX, XFPX
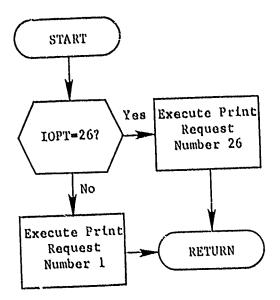
SUBROUTINES CALLED: None

CALLED BY: PRNTNOW

Method:

The formal parameter IOPT determines whether option 1 or 26 appears.
The result of these options appears in the Users Manual, UM 9-77, Volume
III.

Subroutine PRNTOF is illustrated in figure 44.

Figure 44. Subroutine PRNTOF

3.9.7 Subroutine RECON

PURPOSE: To reconstruct data for the WADWPN block.

ENTRY POINTS: RECON

FORMAL PARAMETERS: ICLSS - class index of group

COMMON BLOCKS: C30, C45, FIL21, PAYSAV, SALVO, SMATAD, TGTSAV,
                     WADWPN, WEPSAV

SUBROUTINES CALLED: ALOG, SETPAY, TABLEMUP, VALTAR

CALLED BY: FRSTGD, SCNDGD

Method:

Subroutine SETPAY is called to select use of gravity bombs or ASMs from
the bomber groups. If ASMs are selected probability STK 2 is used in the
calculation of the survival probability STK and the MUP array. The MUP
XMUP, SSIG, and RISK arrays are constructed according to the required
formulae.

The SMAT array is loaded depending upon whether the group is a MIRV.

Subroutine RECON is illustrated in figure 45.
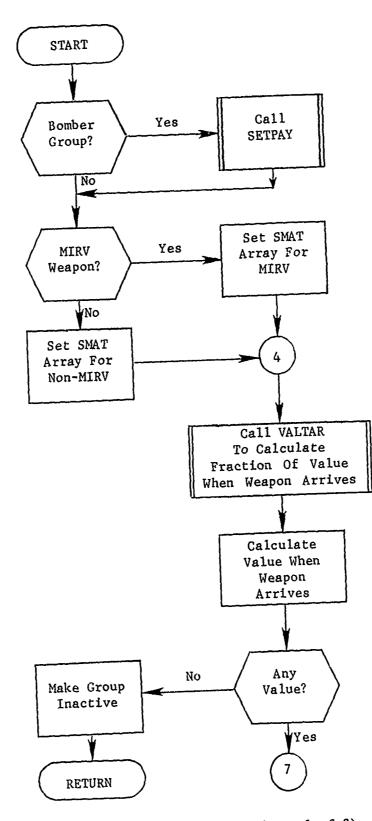
Figure 45. Subroutine RECON (Part 1 of 2)
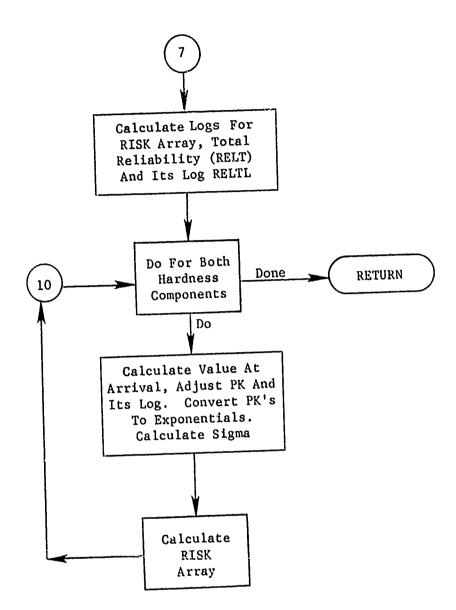
Figure 45. (Part 2 of 2)

3.9.8 Subroutine SETPAY

PURPOSE: This routine sets the bomber payload indicators to
                          specify use of gravity bombs or ASMs.

ENTRY POINT: SETPAY

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, CONTRO, DYNAMI, PAYSAV, SALVO, TGTSAV, WADWPN,
                          WEPSAV

SUBROUTINES CALLED: None

CALLED BY: RECON

Method:

This subroutine sets the bomber payload indicators (array ISETPAY in com-
mon block /SALVO/). The setting calculation considers the allocation
rate of ASMs (array FASM in /SALVO/), the actual fraction of ASMs in each
group array GSEASM in /PAYSAV/, the damage difference between the weapons,
the average damage difference (array AVDE or /SALVO/), and the state of
allocation progress (variable PROGRESS in /CONTRO/).

Five local variables are of some significance to the calculation. The
variable IPREF is used for temporary storage of the payload indicators.
A zero value specifies use of a bomb; a value of one specifies use of an
ASM. The variable DEA contains the expected damage (ignoring the same
vehicle planning factors) if a bomb were used. Local variable CONPAY is
used to weigh the difference in allocation rates relative to the differ-
ence in allocation rates relative to the difference in damage expectan-
cies. Local variable PAYSENS is used to calculate CONPAY.

The variable CONPAY is used to reflect the importance of the allocation
rate difference relative to the damage difference. As CONPAY decreases,
the allocation rate difference increases in importance relative to the
damage difference and vice versa. CONPAY is defined as follows:

$$CONPAY = \frac{PAYSENS}{(CLOSE + 1.5) * PROGRESS}$$

Thus, PAYSENS is merely a multiplicative factor, set internally by SETPAY
to a value of 0.1 which provides a base value for CONPAY. The denomina-
tor of the right-hand term of the equation represents the effects of allo-
cation progress. As the allocation proceeds, CONPAY decreases to put more
weight on the allocation rate differences. Variable CLOSE is a user input
parameter which determines the size of the closing forces. When PROGRESS
equals one, CLOSE increases to increase the closing forces. The table
below displays the values of the denominator for different values of
PROGRESS. CLOSE is set to its default value, 1.05, and CLOSER (another

241

input parameter) is set to its default value, 4.

Table CONPAY Denominator vs. PROGRESS

Denominator	PROGRESS
1.02	.4
1.275	.5
1.913	.75
2.55	1.00 (initial)
6.55	1.00 (after one pass)
8.55	1.00 (highest value of denominator)

After PROGRESS equals on, the value of CLOSE increases by an amount CLOSER for each pass at PROGRESS equal one. The maximum number of passes with this value of PROGRESS is 1.5.

When PROGRESS is zero or in a verification pass, the weapon with the greater damage is selected for allocation. Also, the selection equations are bypassed if a group has only bombs or ASMs.

Figure 46 displays the logic or subroutine SETPAY.
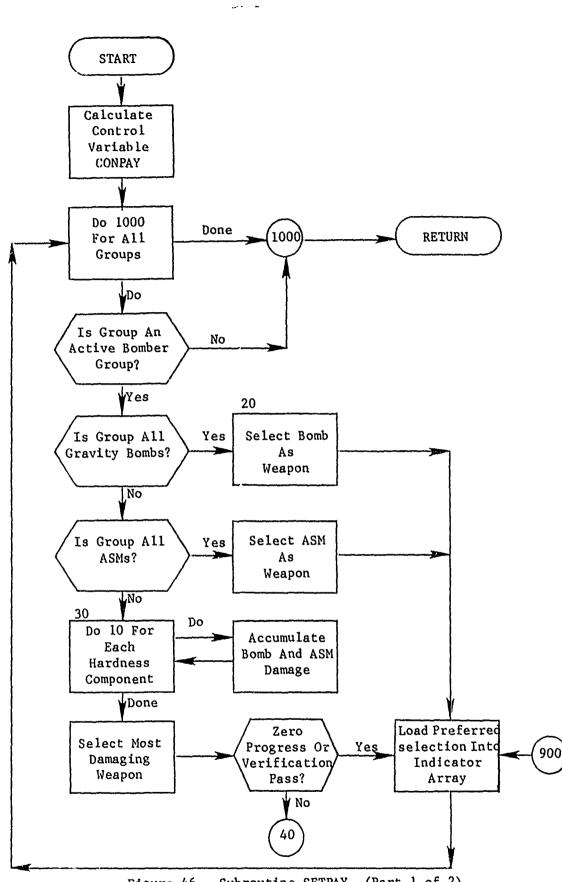
Figure 46.   Subroutine SETPAY   (Part 1 of 2)
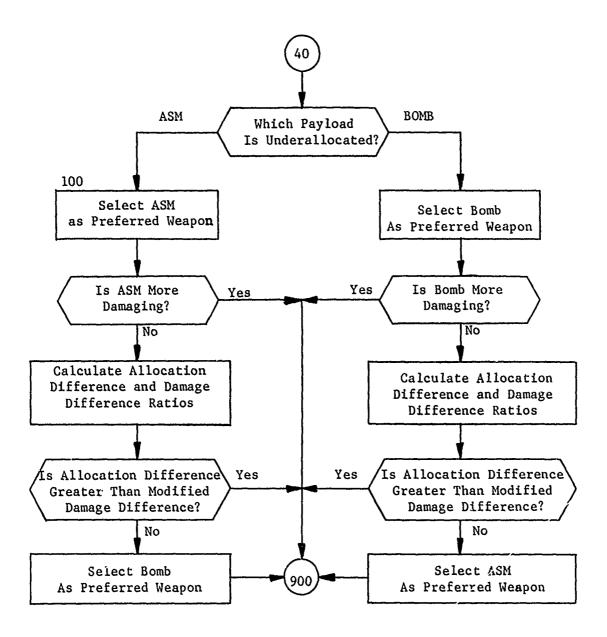
Figure 46. (Part 2 of 2)

3.10 Subroutine SCNDGD[*]

PURPOSE: To read in data on second and later ALOC passes
                         over the target list.

ENTRY POINTS: SCNDGD

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C30, C33, DYNAMI, FIL21, FILL, FIRST, GRPHDR,
                         MULTIP, TARREF, TGTSAV, WADWPN, WEPSAV, WPFIX

SUBROUTINES CALLED: BOMPRM, DIRECT, FRSPLT, HEAD, NEXTTT, NXSPLT,
                         RECON

CALLED BY: MULCON

Method:

First, if this is the first target of the pass, units 15 and 21 and per-
haps 22 are rewound. Then the next target is read and all its data
accessed. Next files 21 and 15 are read in. If file 22 is in use for
this target it is read with its contents replacing those of file 15. The
INACTIVE array is now loaded and the contents of file 15 (22) unpacked
into the WADWPN block. The previous assignments are now read and stored
in the DYNAMI block. Finally, BOMPRM is called to remove the assignments
effects from ASM totals and RECON is called for all groups.

The processing for split multiple targets is somewhat different.

Subroutine SCNDGD is illustrated in figure 47.

---

[*] First subroutine in segment SGD -- all other routines appear in section
3.9.
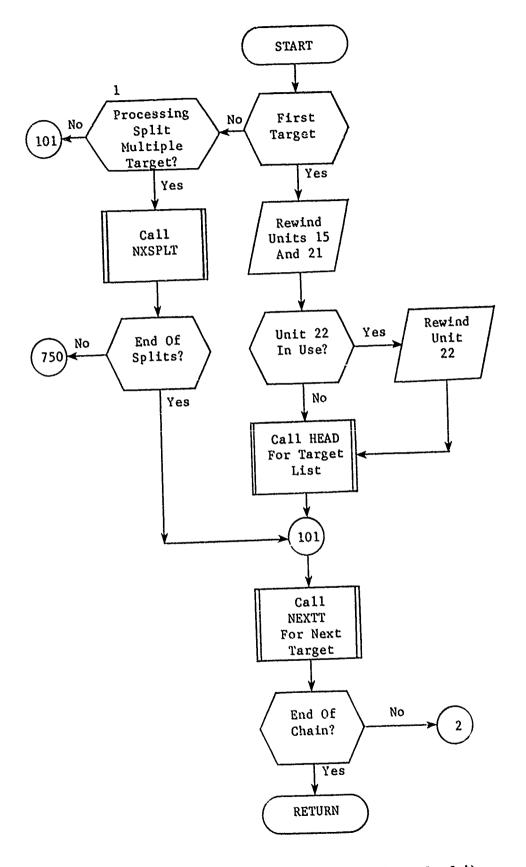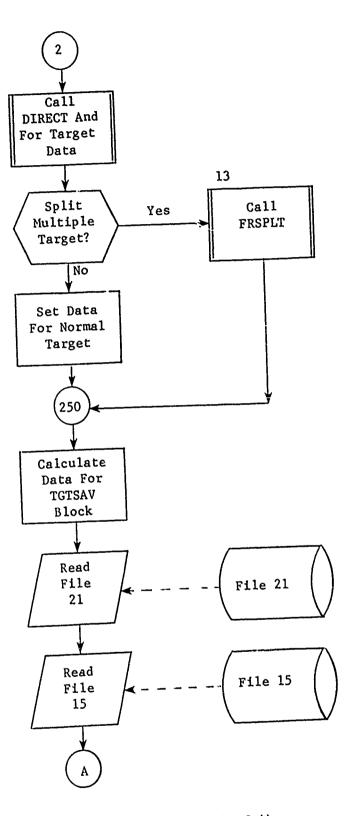
Figure 47.   Subroutine SCNDGD (Part 1 of 4)
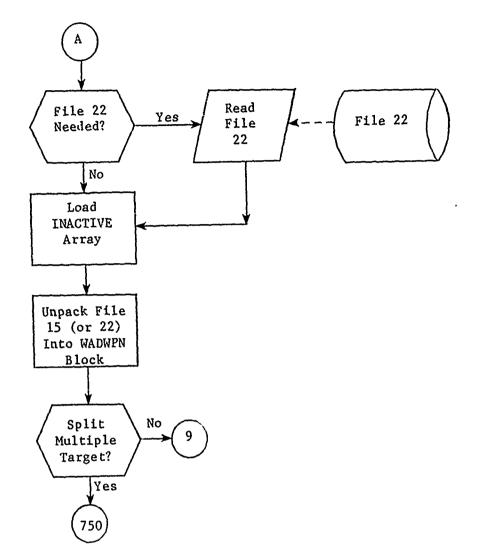
Figure 47. (Part 2 of 4)

Figure 47. (Part 3 of 4)

```
              ┌─────┐
              │  9  │
              └──┬──┘
                 ↓
          ┌───────────────┐
          │    Set To     │
          │   Read In     │
          │  Assignments  │
          └───────┬───────┘
   10             ↓
          ╔═══════════════╗
          ║     Call      ║
          ║  NEXTTT For   ║←─────────────────────┐
          ║     Next      ║                       │
          ║  Assignment   ║                       │
          ╚═══════╤═══════╝                       │
                  ↓                               │
           ╱────────────╲                 ┌───────────────┐
          ╱              ╲      No         │    Store      │
         ╱   End Of       ╲───────────────→│  Assignment   │
         ╲   Chain?       ╱                │  In DYNAMI    │
          ╲              ╱                 │    Block      │
           ╲────────────╱                  └───────────────┘
                  │ Yes
                  ↓
              ┌──────┐
              │ 750  │
              └──┬───┘
                 ↓
          ╔═══════════════╗
          ║     Call      ║
          ║  BOMPRM To    ║
          ║   Remove      ║
          ║  Assignment   ║
          ╚═══════╤═══════╝
                  ↓
          ┌───────────────┐
          │   Increment   │←──────────────┬──────────────────┐
          │  Weapon Group │               │                  │
          │   Counter     │               │                  │
          └───────┬───────┘               │                  │
                  ↓                        No                 │
           ╱────────────╲          ╱────────────╲     ╔═════════════╗
          ╱   Last       ╲   No    ╱   Group      ╲Yes ║    Call     ║
          ╲   Group      ╱────────→╲   Active?     ╱───→║   RECON     ║
          ╲   Done?      ╱          ╲             ╱     ╚═════════════╝
           ╲────────────╱            ╲───────────╱
                  │ Yes
                  ↓
            ╭───────────╮
            │  RETURN   │
            ╰───────────╯
```

Figure 47. (Part 4 of 4)

249
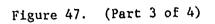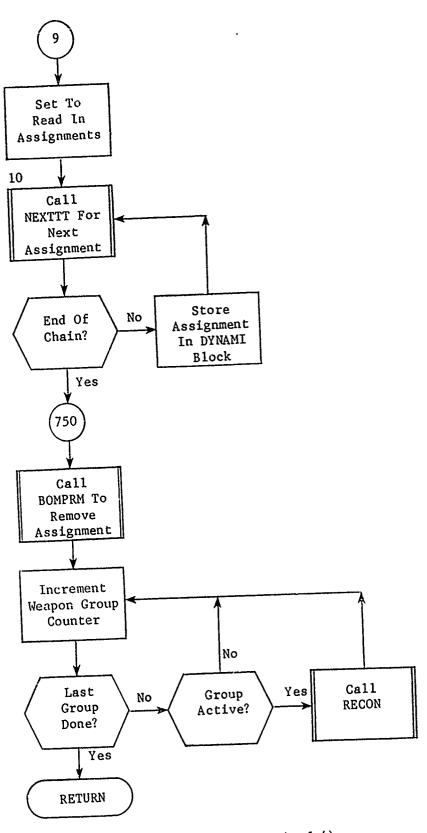
3.11 Subroutine STALL[*]

PURPOSE: This routine[*] determines the sequence of weapon additions and deletions required to achieve a near optimum allocation of weapons to a target.

ENTRY POINTS: STALL

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, C33, CONTRO, DYNAMI, MULTIP, SALVO, SURPW, WADFIN, WADOTX, WADWPN

SUBROUTINES CALLED: ADDSAL, INITSAL, RESTORE, WAD

CALLED BY: MULCON

Method:

STALL controls WAD's addition and deletion of weapons by setting the values of the following three variables:

WADOP   in /CONTRO/

G   in /WADOTX/

NW   in /WADOTX/

WADOP has the following options:

WADOP = 1   Initialize allocation

WADOP = 2   Finalize allocation (optional print of results)

WADOP = 3   Add weapon from group (G)

WADOP = 4   Delete (NW)th weapon now on target

To facilitate monitoring the operation of STALL, the variable STALPRIN is also set to provide a unique indicator of the position in STALL where WAD is called. This variable is printed under the print option 22.

The input data for STALL consists of the following six variables supplied by WADOUT in /WADOTX/:

PPMX and IPPMX - the maximum potential increase in effective profit for any weapon, and the index G to that weapon group, respectively.

---

[*] First subroutine of segment STAL.

PVRMX a·d IPVRMX – the maximum effective efficiency for any weapon, and the index to that weapon group, respectively.

DPMN and IDPMN – the minimum marginal effective profit for any weapon now assigned, and the index to that weapon in the list of weapons assigned, respectively.

The flowchart for STALL is in four parts. The first part contains the setup and single weapon allocation phase. This phase provides a prompt exit from STALL if the indicated allocation consists of one weapon or less. This part also includes a dummy version of STALL which is used to reproduce the prior allocation independent of current payoff data. This option is used in place of the usual verification phase (when PROGRESS = 2) if IVERFY = 2. This mode of operation checks the effects of an alternate level of interweapon correlation, CORR2.

Before going into the normal allocation phase, the initial value of the time-of-arrival error allowance DELTVAL is saved, so that it can be restored if it is necessary to change it. This quantity determines the maximum fractional difference in target value at the time-of-arrival of weapons that are allowed to use the same time-of-arrival bin in the calculations by WAD. If the indicated allocation would result in an overflow of the available time-of-arrival bins, this quantity is increased by STALL and the allocation is reinitialized for another attempt.

Before performing any operations, STALL first calls subroutine INITSAL. This routine initializes the arrays in common block /SALVO/. At the end of the routine, subroutine RESTORE restores the multipliers for the salvoed groups.

The second part of STALL processes the fixed assignment data. It puts the weapons down on target. After initializing WAD, the routine checks the pass number. On the first pass, the fixed assignments were placed in array IG by FRSTGD. In later passes, they are in the IGO array. The statements after 126 determine the number of weapons the assignment represents. If DEFALOC has made an allocation on the previous pass, the number of missiles allocated from each group is shown as a negative number in the KORRX array. If the KORRX entry is positive, there is only one weapon assigned from the group. STALL then checks the INACTIVE flag to see if the weapon can reach the target. If not, an error message is printed and the assignment request is ignored. If statement 443, WAD is called to actually put the weapon on target. If the weapon is a salvoed missile, subroutine ADDSAL is called to modify the salvoed weapon stockpile.

For the fixed weapons, no change to the variable SURPWP is made as the weapon is allocated, since th variable controls the allocation of only those weapons used by the methematical allocator.

If the fixed weapon cannot be allocated because of its active flag (INACTIVE (G)) prevents allocation, the error message notifying the user

251

of this fact lists the reason for the inactivity. The reason is contained in array MORRX for all inactive groups.
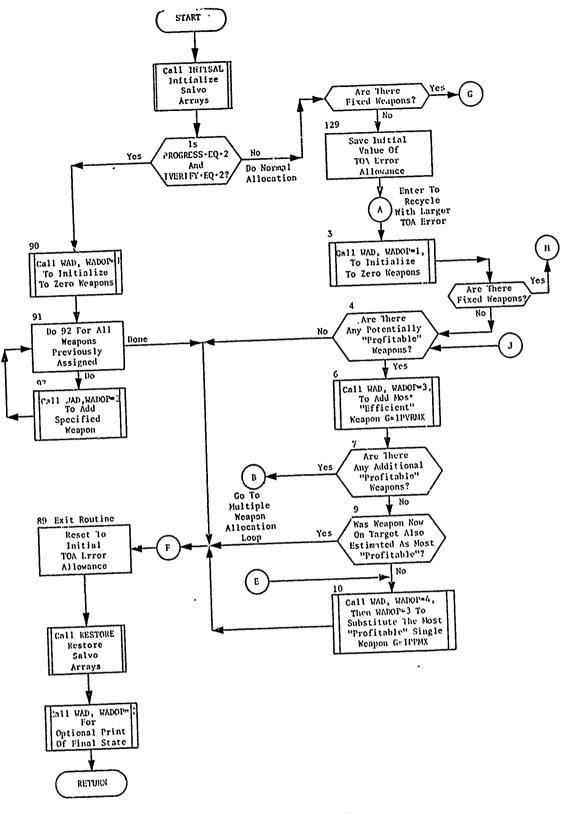
The third part of the subroutine provides an initial laydown of weapons if multiple weapons are indicated against the target. As this laydown progresses two types of array overflows could occur. The overflow of available time-of-arrival bins has just been discussed. It results in simply restarting the allocation. The other possibility is that the total number of weapons assigned could exceed the maximum number (30) permitted. In this event, control is passed to the refinement loop just as it would be in the normal exit at the bottom of Part III.

Part IV independently checks for such a potential overflow (near statement 59) and if it is threatened, a cycle of operation is generated through the connector ⓓ that results in removal of the least profitable weapons (statement 52) and replacement by the most profitable available weapons (statement 56). This sequence is terminated either by entering the normal refinement process when the residual target value is reduced so that no potential weapons remain profitable (statement 54), or by using exit Ⓕ after statement 59 if no combination of 30 weapons can be found which will reduce the target value sufficiently.

The operation of the normal refinement loop, which cycles through the branch at statement 71 until the tests at statement 66 are satisfied, is discussed in detail in appendix A and will not be repeated here.

Figure 48 illustrates segment STALL.

Figure 48. Segment STALL.
Part I: Setup and First Weapon
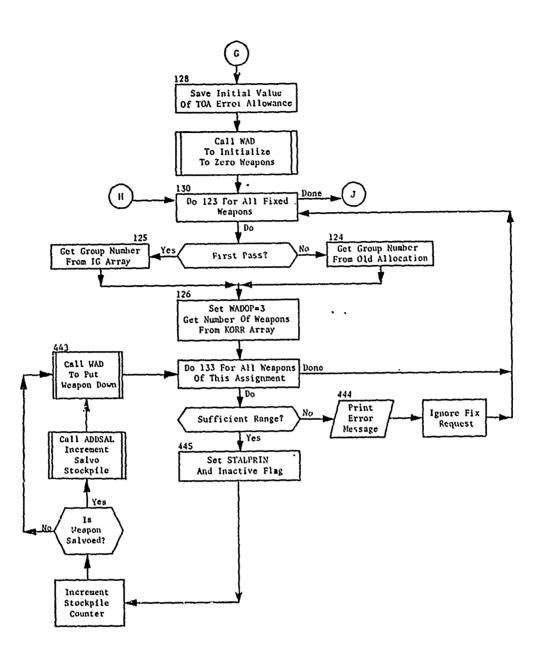
Figure 48. Part II: Fixed Weapon
Assignment Processing

254
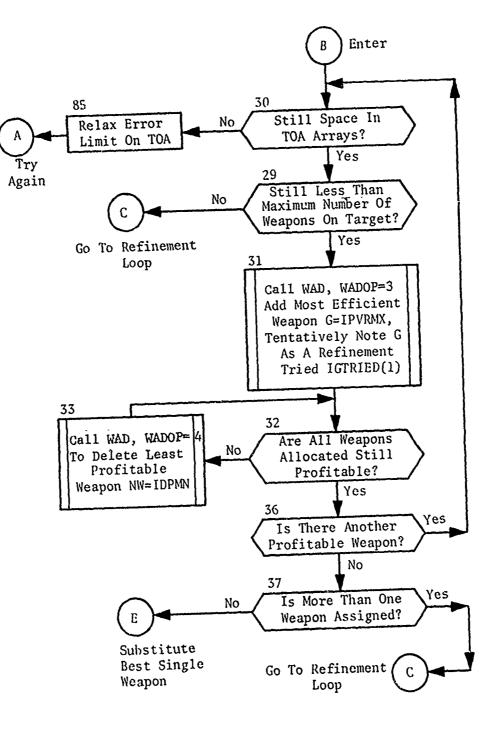
Figure 48. Part III: Multiple
Weapon Laydown
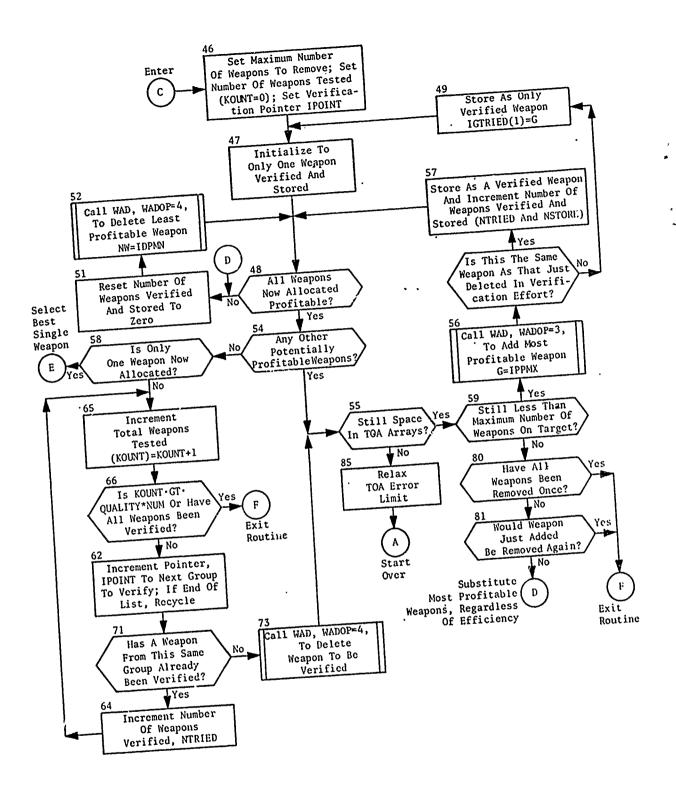
Figure 48. Part IV: Multiple Weapon Refinement Loop

3.11.1 Subroutine FORMATS

PURPOSE: This subroutine determines the best 10-column BCD
                        format for a variable.

ENTRY POINTS: FORMATS

FORMAL PARAMETERS: None

COMMON BLOCKS: FORMTT

SUBROUTINES CALLED: None

CALLED BY: PRNTOS

Method:

The variable to be formatted is INWORD, the first word in common /FORMTT/.
(This word is equivalenced to WORDIN.) The best 10-column format is re-
turned in NFORMAT, the second word of common /FORMAT/. The resulting
value of NFORMAT can be used in a FORTRAN output statement such as:
PRINT NFORMAT, INWORD.

The names WORDIN and INWORD are equivalenced to allow correct specifica-
tion (real or integer) for any possible input. The name NFORMT is in-
ternally equivalenced to the variable name NFORMAT for convenience. In-
ternal to FORMATS, the absolute value of the input variable is kept in
INABS, equivalenced to ABSIN for type specification purposes.
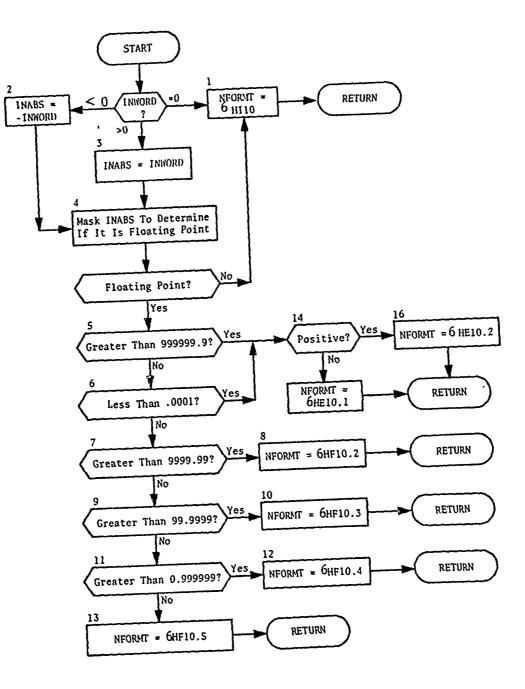
To determine if a number is fixed or floating point, the first 12 bits
of the absolute value are tested. If a bit is set, the number is assumed
to be floating point, since all normalized floating point numbers have at
least one bit set in this range. Thus, if an integer quantity greater
than 16,777,216 is input, it will be treated as if it were a floating
point variable. However, no variable input from PRNTNOW can have a value
of that magnitude, so this restriction is never a handicap. Table 6
presents the returned formats for each range of input values.
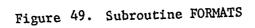
Subroutine FORMATS is illustrated in figure 49.

Table 6. Calculated Formats for Variables

RAN_E OF ABSOLUTE VALUE OF VARIABLE, X	FORMAT OF NEGATIVE	FORMAT OF POSITIVE
0 or Integer Variable	110	110
$0 < X < 0.0001$	E10.1	E10.2
$0.0001 \leq X \leq 0.999999$	F10.5	F10.5
$0.999999 < X \leq 99.9999$	F10.4	F10.4
$99.9999 < X \leq 9999.99$	F10.3	F10.3
$9999.99 < X \leq 999999.9$	F10.2	F10.2
$999999.9 < X$	F10.1	F10.2

Figure 49. Subroutine FORMATS

259

3.11.2 Function FMUP

PURPOSE: This function computes a survival probability
                             given a sum of kill factors for all weapons
                             allocated to the target.

ENTRY POINTS: FMUP

FORMAL PARAMETERS: S - A sum of kill factors

COMMON BLOCKS: WADWPN

SUBROUTINES CALLED: None

CALLED BY: WAD, RESVAL

Method:

This function is a straightforward application of the two damage laws
used in the system. The law used on the current target is determined
by the variable ILAW in common /WADWPN/. If this variable was set
positive in subroutine RECON, then the square root damage law is used.
Otherwise, the exponential law is used.

The input formal parameter S is a sum of the kill factors (each prepared
by function TABLEMUP) for all weapons assigned to the target.

The value returned by the function is defined as follows:

Exponential Law

$$FMUP = exp\ (-S) \quad \text{(statement 1)}$$

Square Root Law

$$FMUP = (1 + \sqrt{S}\ ) * exp\ (-\ \sqrt{S}\ ) \quad \text{statement 2}$$

Figure 50 illustrates function FMUP.
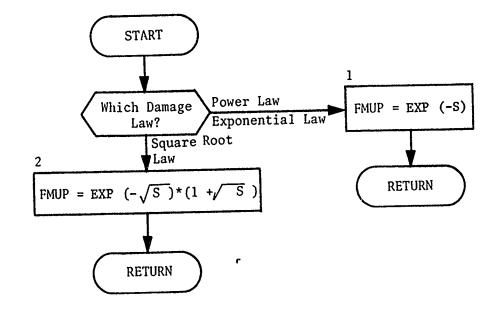
Figure 50. Function FMUP

3.11.3 Function LAMGET

PURPOSE:

This real valued function calculates the Lagrange multiplier for salvoed missile groups.

ENTRY POINTS:

LAMGET

FORMAT PARAMETERS:

LAM – Initial multiplier
P   – Salvo balance variable
ISAL – Salvo number

COMMON BLOCKS:

None

SUBROUTINES CALLED:

None

CALLED BY:

DEFALOC, SALVAL (entry INITSAL)

Method:
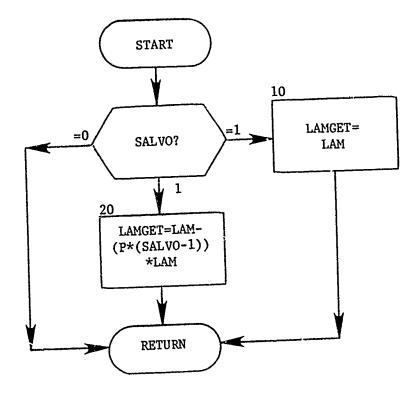
Note that this function is a real valued function. Its correct usage requires that a REAL LAMGET specification be present in the calling program.

The formal parameters specify the original or first salvo multiplier (LAM), the salvo balance variable maintained by PUPDT(P), and the salvo number (ISAL).

The returned value LAMGET is computed as follows:

$$LAMGET = LAM - P * (ISAL -1) * LAM$$

Function LAMGET is illustrated in figure 51.

Figure 51. Function LAMGET
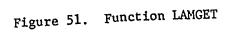
3.11.4 Subroutine PREMIUMS

PURPOSE:	This routine calculates the premium used by WADOUT in evaluating the benefit of using or not using weapons from specific groups.
ENTRY POINTS:	PREMIUMS
FORMAL PARAMETERS:	G - An integer group number
COMMON BLOCKS:	C30, CONTRO, MULTIP, PREMS, SURPW, WPFIX
SUBROUTINES CALLED:	None
CALLED BY:	WAD, DEFALOC, MULCON, SPLIT

Method:

The formal parameter G specifies for which weapon group the premium is to be recomputed. Three modes are provided for the computation of the premiums, depending on the value of PROGRESS.

<p style="margin-left:2em;">PROGRESS < 1.0   A normal linear premium is computed which keeps the allocator from producing allocations with unnecessarily large deviations from the desired allocation rate (statement 12).</p>

<p style="margin-left:2em;">PROGRESS = 1.0   The normal premium is augmented by a step function which strongly motivates the allocator to exactly match the stockpile statement 1).</p>

<p style="margin-left:2em;">PROGRESS > 1.0   The subroutine exists with a zero premium for use in verification allocations (statement 10).</p>
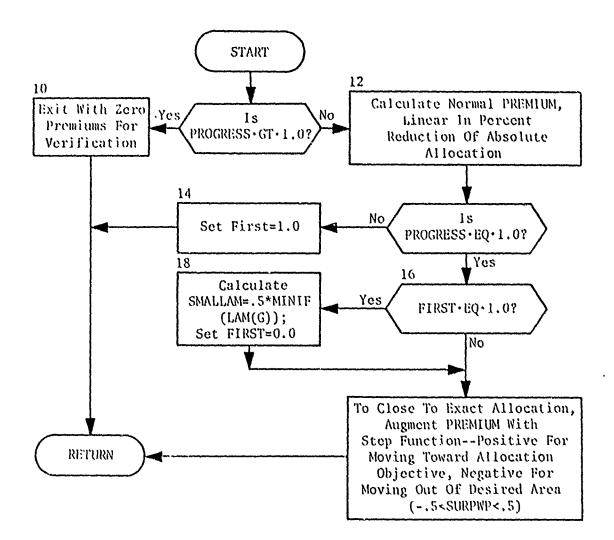
Since the calculation of the step function premium requires the quantity SMALLAM = .5 * LAMEF(G) for the lowest value of LAMEF, this quantity is evaluated only on the first call of PREMIUMS after PROGRESS is 1.0 (statement 18). Thereafter (since the values of LAMEF are frozen while PROGRESS = 1.0) there is no need to recompute this quantity.

PREMIUM is subtracted from the cost of adding a weapon. DPREMIUM is subtracted from the cost of deleting a weapon.

Where a weapon surplus exists (i.e., SURPWP(G) > 0), PREMIUM(G) > 0.0 and DPREMIUM(G) ≤ 0.0. Where a weapon deficiency exists (i.e., SURPWP(G) < 0), the reverse is true.

Figure 52 illustrates subroutine PREMIUMS.

Figure 52. Subroutine PREMIUMS

Note

--Where a weapon surplus exists, (PREMIUM) is positive, and the premium for deleting (DPREMIUM) is negative.

--Where a deficiency exists (SURPWP<0), the reverse is true.

3.11.5 Subroutine PRNTOS

PURPOSE: To produce optional prints for overlay STAL (op-
                     tions 12 and 13)

ENTRY POINTS: PRNTOS

FORMAL PARAMETERS: IOPT - Print option number
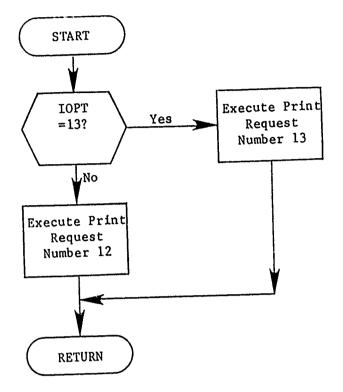
COMMON BLOCKS: C30, C33, CONTRO, DYNAMI, FORMTT, PREMS, WADFW,
                     WADLOC, WADOTX

SUBROUTINES CALLED: None

CALLED BY: PRNTNOW

Method:

The formal parameter IOPT determines whether option 12 of 13 appears.
The result of these options appears in the Users Manual, UM 9-77, Vol-
ume III.

Figure 53. Subroutine PRNTOS

267
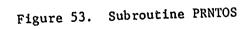
3.11.6 Subroutine SALVAL

PURPOSE: This routine selects the preferred salvo for each salvoed missile group, saves, and restores the appropriate Lagrange multipliers.

ENTRY POINTS: SALVAL, INITSAL, NEWSAL, RESTORE

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, CONTRO, MULTIP, PAYSAV, SALVO, TGTSAV, WADOTX, WADWPN, WEPSAV, WPFIX

SUBROUTINES CALLED: GLOG, LAMGET, VALTAR

CALLED BY: STALL, DEFALOC, WAD

Method:

There are three entry points, INITSAL, RESTORE, and NEWSAL. Entry INITSAL is called at the beginning of automatic weapon allocation in STALL or DEFALOC. First, it saves the multipliers LAM for each salvoed group in array SAVLAM. Then, for each group and each salvo, INITSAL determines the salvo number and cost of the best salvo. The profit of each salvo is defined as

$$PR_i = VTAR_i - TLAM_i$$

where $VTAR_i$ is the value of the target at arrival time of salvo i and $TLAM_i$ is the value of the multiplier for salvo i as determined by function LAMGET. The salvo with the highest PR_i is selected as the best salvo. The salvo number is entered in array MYSAL in common block /SALVO/ and the cost L_i replaces the multiplier for the group LAM. Entry RESTORE merely restores the original values of the multipliers (SAVLAM) back to the multiplier array LAM. Entry NEWSAL checks the current allocation and running sum for the weapon just added or deleted. If the salvo has been completely allocated, NEWSAL flags the salvo as unavailable. Entry SALVAL is never used.

Entry INITSAL

This entry is the most complex of SALVAL. The local variable IAM is set to "INITSL" to flag the exit points after statement 420 and at statement 500. The first processing (DO loop to statement 10) saves the Lagrange multipliers (LAM) in the SAVLAM array in /SALVO/. The major processing in INITSAL occurs in the DO loop to statement 100 over all the salvoed groups. Within this loop, the DO loop to statement 100 investigates each salvo to determine its worth. Array IHAVE in /SALVO/ is used to exclude salves from consideration. If IHAVE (I, J) is false, then salvo I in group J does not have any available weapons and is ignored. If weapons were available, a jump is made to statement 420 in entry

NEWSAL. That section of code, described later, determines if the salvo to be considered is overallocated beyond its limit. If so, the salvo is not further considered. If the salvo is available, function VALTAR is used to obtain VTAR the target value for the salvo. Function LAMGET is used to calculate TLAM, the multiplier for the salvo. The best salvo is the one with the highest positive difference between VTAR and TLAM. When this maximum is selected, MYSAL in /SALVO/ is set to the best salvo number. LAM in /WPFIX/ is set to the appropriate multiplier. The arrays TOA, TVALTOA, and VTOA (in /WADWPN/) are set to correspond to the arrival time of the best salvo.

Entry INITSAL is illustrated in part 1 of figure 54.

Entry RESTORE

This simple entry point uses a DO loop to statement 1000 to restore the original values of the Lagrange multipliers (SAVLAM) to the multiplier array (LAM).

This entry is illustrated in part 3 of figure 54.

Entry NEWSAL
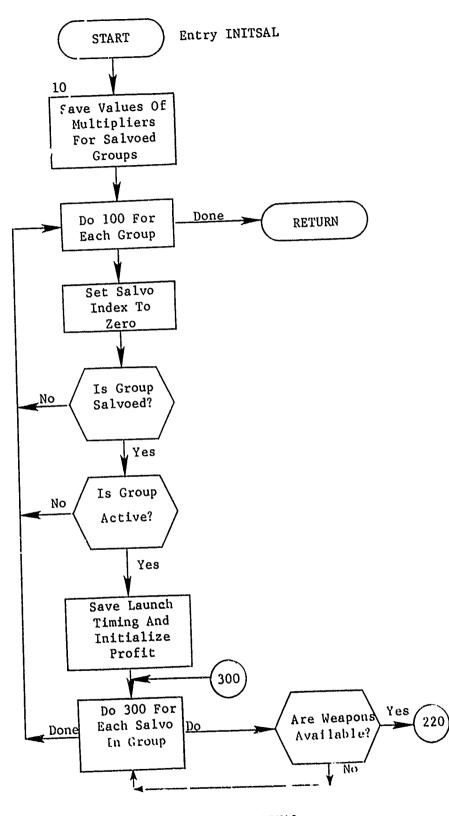
This entry is used after each allocation of salvoed weapons to determine if the salvo is still available. Each salvo has a maximum limit of overallocation. Before PROGRESS equals one, this limit is +225 which is the largest number which can be stored in the NSALAL array. (A description of the structure of the NSALAL array is contained in the Method section of subroutine ADDSAL, in this chapter.) When PROGRESS equals one, a more severe limit is imposed in order to accelerate closure to the stockpile. In this case the limit is zero. That is, a salvo is available only if it is underallocated.

To exit from this entry, local variable IAM is checked. If it is "INITSL," then the original call was to INITSAL and control passes to statement 220 in entry INITSAL. If it is "NEWSAL" the routine exits.

Note that if a salvo is unavailable because of limit on a call to NEWSAL, the routine attempts to find the closest lower available salvo. If none can be found, then MYSAL is set negative to flag that no salvo is available for this group.

Entry NEWSAL is illustrated in part 4 of figure 54.

---
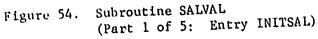
* In line function IHAVE extracts information from logical array LXIHAVE.

Figure 54. Subroutine SALVAL
(Part 1 of 5: Entry INITSAL)

270

Figure 54.   (Part 2 of 5)

```
                    ┌─────────┐
                    │  START  │
                    └─────────┘
                         │
                         ▼
              ┌─────────────────────┐
              │   Do 1000 For       │   Done    ┌──────────┐
      ┌──────▶│   All Salvoed       │─────────▶│  RETURN  │
      │       │   Missile           │           └──────────┘
      │       │   Groups            │
      │       └─────────────────────┘
      │                │ Do
      │                ▼
      │       ┌─────────────────────┐
      │       │   Restore           │
      └───────│   Saved             │
              │   Multiplier        │
              └─────────────────────┘
```
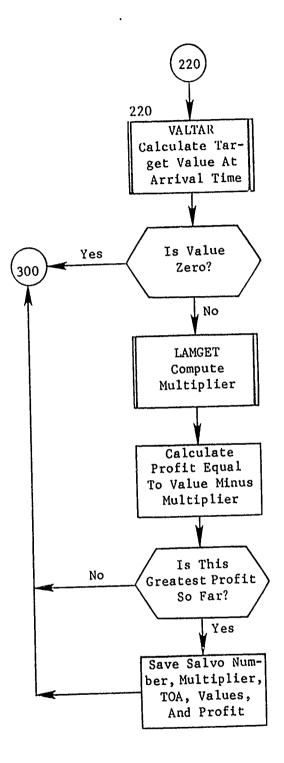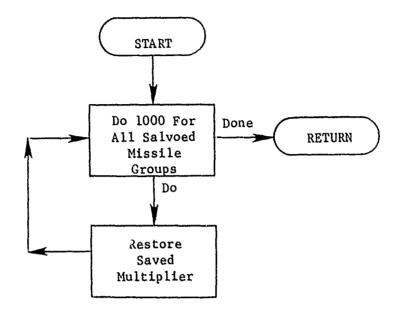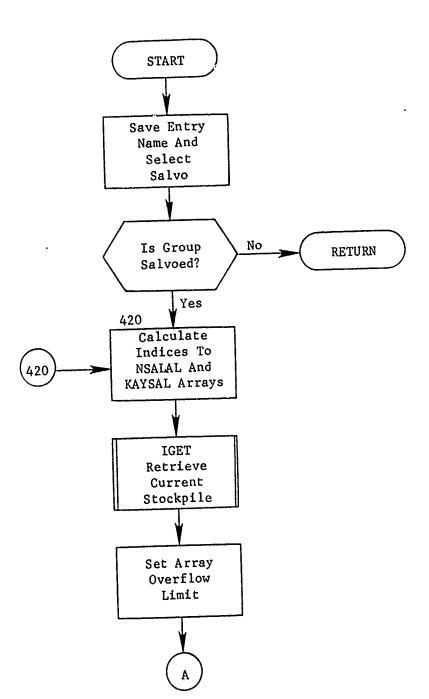
Figure 54.   (Part 3 of 5:   Entry RESTORE)

272
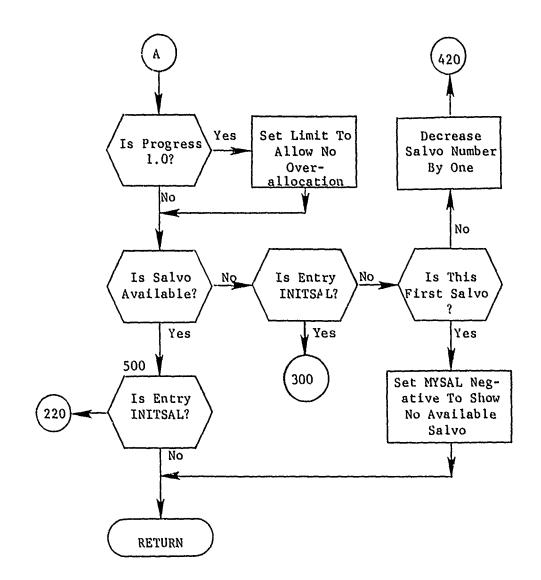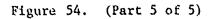
Figure 54. (Part 4 of 5: Entry NEWSAL)

Figure 54. (Part 5 of 5)

3.11.7 Subroutine SPLIT

PURPOSE:	To split multiple targets for allocation purposes.
ENTRY POINTS:	SPLIT
FORMAL PARAMETERS:	NEWSP - Number of targets to be split off
COMMON BLOCKS:	C30, C33, DYNAMI, MULTIP, SPLITS, SURPW, WADFIN, WADWPN
SUBROUTINES CALLED:	PREMIUMS
CALLED BY:	DEFALOC, WAD

Method:

If target has not been split before, the index[*] and offset[*] for its file 25 record are calculated and the record read in. The C33 data is stored in the file 25 buffer and the split data set into block C33. If target has been split before the new split is set up using the data from the old split and rearranging the file 25 buffer. In either case, the weapon surpluses are adjusted and the premiums recomputed.

Subroutine SPLIT is illustrated in figure 55.

---

[*] C33 data is stored on file 25 after any weapon group records. There are five C33 sets possible per multiple target and three target data sets per file 25 record.
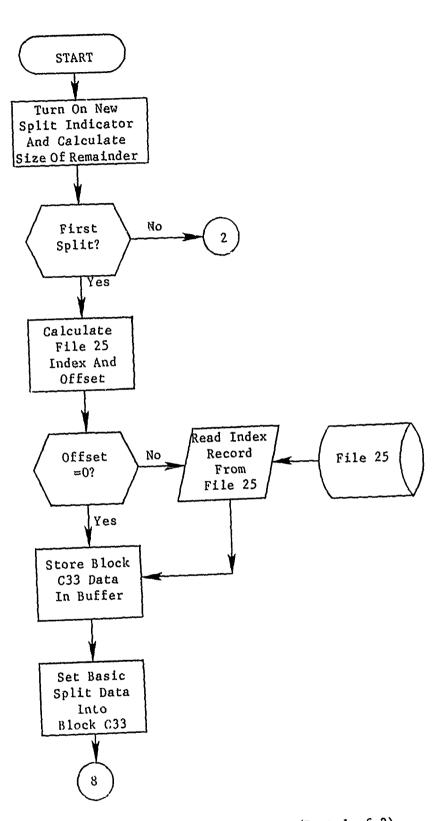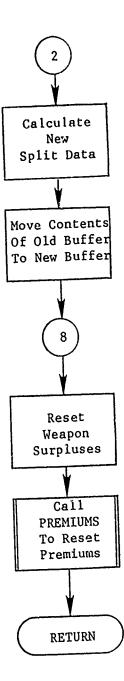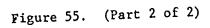
Figure 55.   Subroutine SPLIT (Part 1 of 2)

```
        ┌───┐
        │ 2 │
        └─┬─┘
          │
          ▼
   ┌──────────────┐
   │  Calculate   │
   │     New      │
   │  Split Data  │
   └──────┬───────┘
          │
          ▼
   ┌──────────────┐
   │Move Contents │
   │Of Old Buffer │
   │To New Buffer │
   └──────┬───────┘
          │
          ▼
        ┌───┐
        │ 8 │
        └─┬─┘
          │
          ▼
   ┌──────────────┐
   │    Reset     │
   │   Weapon     │
   │  Surpluses   │
   └──────┬───────┘
          │
          ▼
   � ┌────────────┐ �│
   │ │    Call    │ │
   │ │  PREMIUMS  │ │
   │ │  To Reset  │ │
   │ │  Premiums  │ │
   ╵ └────────────┘ ╵
          │
          ▼
     ╭──────────╮
     │  RETURN  │
     ╰──────────╯
```

Figure 55.   (Part 2 of 2)

3.11.8 Subroutine WAD

PURPOSE: This routine carries out the addition and deletion
                            of weapons as specified by STALL. After each
                            change in the allocation to a target WAD computes
                            the surviving target value VT; for each potential
                            weapon G, the potential surviving value VTP(G) if
                            a weapon for that group were added; and for each
                            weapon currently on the target, the potential sur-
                            viving value VTD(NW) if a weapon from that group
                            were deleted.

ENTRY POINTS: WAD

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, C33, CONTRO, DYNAMI, MULTIP, PREMS, SALVO,
                            SURPW, TGTSAV, WADFIN, WADLOC, WADOTX, WADWPN,
                            WEPSAV, WPFIX

SUBROUTINES CALLED: ADDSAL, FMUP, NEWSAL, PREMIUMS, PRNTALL, SPLIT,
                            WADOUT

CALLED BY: STALL

Method:

The surviving target value VT is given by:

$$VT = \sum_{J=1}^{J=M} \sum_{N=0}^{N=NN} \left[V(N,J) - V(N + 1,J) \right] * S(N,J)$$

where

$$S(N,J) = FMUP \left[(MU(N,J) ** 2)/(MU(N,J) + SIG(N,J)) \right]$$

The function FMUP(X) is defined as follows:

Exponential damage law:

$$S(N,J) = FMUP(X) = \exp (-X).$$

Square root damage law:

$$S(N,J) = FMUP(X) = (1 + \sqrt{X}) \ (\exp (-\sqrt{X})).$$

The index J is over the hardness components, and the index N is over the
time of arrival bins. V(N,J) is the unattrited value of the Jth hard-
ness component at the time corresponding to the Nth time-of-arrival bin.

278

MU(N,J) is the summation of MUP(G,J) for all weapons arriving at the target through the Nth time-of-arrival bin.

SIG(N,J) is the summation of all the cross terms (relative to the Jth hardness component) between all the weapons on the target by the Nth time-of-arrival bin.

The calculations indicated above are carried out by WAD in sets of scratch arrays and computation arrays that make it possible to retain all intermediate values in the calculations. The resulting scratch pad is then referred to and used wherever possible in computing the modified value of VT,VTP(G) if a weapon G is added, and the modified value of VT,VTD(NW) in the (NW)th weapon is deleted.

Table 7 will help to visualize the scratch pad results that are stored. This illustrates the calculation at a time when there are three weapons on the target (NUM = 3), which use two time-of-arrival bins (NTOA = 2), and there are two hardness components (M = 2).

The first time-of-arrival bin always corresponds to 0 time and reflects the maximum target value, VAL(1) = 100, in the example. Bin number 2 corresponds to the first actual time of arrival and it contains two weapons NWP(2) = 2, but the unattrited target value at that time is lower, VAL(2) = 60. Thus, 40 units of value are assumed to escape before the first two weapons arrive. Similarly, an additional 20 units are assumed to escape before the last weapon arrives. The total value, 100 units is assumed to be distributed 80-20 between the two hardness components (VO(1) = 80, VO(2) = 20). The calculation is carried out in parallel for the two hardness components JH = 1, JH = 2. The total surviving and escaping value for each component, VSN(4,J), is added to obtain the total surviving value:

$$VT = 57.6 = 48.0 + 9.6$$

The intermediate computation values not previously defined, VS and VSN, are defined as follows:

$$VS(N,J) = \left[V(N,J) - V(N + 1,J) \right] * S(N,J).$$

$$VSN(N + 1,J) = \sum_{NI=1}^{NI=N} VS(NI,J)$$

These calculations which constitute the core of the calculations of WAD are carried out in statements 400 through 406 of WAD.

It is useful to visualize how this computation (as illustrated in table 7 would be revised if a new weapon were added. The new weapon might have a time of arrival between bins 2 and 3. In this case a new column 3 would have to be created for the weapon and the contents of columns 3 and 4 would have to be moved over. MU, SIG, and S for column 2 would be unaffected by a weapon arriving at a later time and would remain unchanged.

279

Table 7. Illustrating Calculation of Actual Payoff on Target

TIME OF ARRIVAL CELL		1	2	3	4	5
	NWP(TOA)	0.0	2.0	1.0	0.0	0.0
	VAL(TOA)	100.0	60.0	40.0	0.0	0.0

Hardness Element

		1	2	3	4	
JH=1	V(TOA,J)	80.0	48.0	32.0	0.0	
VO(1)=80	MU(TOA,J)	0.0	.90	1.70		
	SIG(TOA,J)	0.0	.274	.385		
	S(TOA,J)	1.0	.50	.25		
	VS(TOA,J)	32.0	8.00	8.00		
	VSN(TOA,J)	0.0	32.00	40.00	48.0	

Hardness Element

		1	2	3	4	
JH=2	V(TOA,J)	20.0	12.0	8.0	0.0	
VO(2)=20	MU(TOA,J)	0.0	2.0	3.0		
	SIG(TOA,J)	0.0	.48	.915		
	S(TOA,J)	1.0	.20	.10		
	VS(TOA,J)	8.0	0.80	.80		
	VSN(TOA,J)	0.0	8.00	8.80	9.60	

$$VT = 57.6$$
$$NUM = 3$$
$$NTOA = 2$$
$$M = 2$$

However, the value of VS would be changed since the value of $V(N + 1, J)$ should now reflect the target value for the new column 3 and would be higher than the 32.0 now shown for column 3, JN   1. The value of MU and SIG in the new column 3 would be the same as that in column 2 except that MU would be augmented by MUP(G,J) for the new weapon and SIG would be augmented by the cross term between the new weapon and all other weapons on the target at that time. The same rule of course applies for all succeeding time-of-arrival bins. In each following column (including old column 3, now 4) the previous value of MU is increased by MUP for the weapon added, and the value of SIG is increased by the cross terms between all weapons previously on the target and the weapon added.

Of course, the new weapon might fit into one of the existing time-of-arrival bins. In this case it would be unnecessary to make a new column, and it would be unnecessary to recompute VS for the previous column. The value of MU and SIG would simply be augmented in the corresponding column and in all succeeding columns as before. Naturally, after the values of MU and SIG are revised, the value of S must be recomputed and the VS and VSN must be revised in the columns affected.

We now recall that WAD is required to provide potential weapon added target value VTP(G) for every weapon group each time a weapon is added or deleted. Obviously the calculation every time of all the above cross terms could be very time consuming. Moreover, precalculation of all individual cross terms for 250 weapon groups would be equally impractical. The technique adopted, therefore, was to calculate cross terms for each weapon group, but only with the weapons already on the target. Since the process of augmenting the values of SIG must be carried out individually for each time-of-arrival bin, the resulting data are stored by time-of-arrival bins. That is, for each weapon group G and each time-of-arrival column, N, data are stored which indicates the amount by which SIG would be increased in that column if the weapon G were added. These data are stored in the array SIGP(G,J,N). Using these data, the effective augmentation of SIG to calculate VTP(G) for each weapon group G can be accomplished simply by adding SIGP for the appropriate column, and the augmentation of MU is accomplished simply be adding MUP for each weapon G.

Table 8-A may help to visualize how these data are used. Each potential group G is tagged with the index ITOA(G) of the time of arrival column it would occupy if it were added. In addition, it is also noted whether the weapon would generate a new column (IADDTOA=1) or share the column with the weapons already there (IADDTOA=0). The situation illustrated here in table 8 corresponds to the same one illustrated in table 7. Notice that for each weapon group G the array SIGP(G,J,NI) contains a significant (usually nonzero) data for NI ITOA(G) -IADDTOA. The extra term in the column NI = ITOA(G) -1 for rows 1 and 5, where the weapon group would require a new column (IADDTOA=1), is to provide a term required for the new columns if this weapon were added. Since addition of this weapon would move all columns (including its own) one position to the right, the resulting term would be in the proper position after moving, even though it is in an incorrect column at present.

281

Table 8. Illustrating Quantities Calculated for Potential Weapon Added and Deleted Payoffs

A. Data on All Potential Weapons

G	VTP (G)	ITOA (G)	IADDTOA (G)	J	DSIG, 3	NI=1	NI=2	NI=3	NI=4
1	2	0	1	.021	0.0	.042	.063	
			2	.103	0.0	.206	.659	
2	3	0	1	.052	0.0	0.0	.45	
			2	.660	0.0	0.0	4.23	
3	2	0	1	.274	0.0	.548	1.653	
			2	.780	0.0	1.560	1.880	
4	3	0	1	.005	0.0	0.0	.010	
			2	.020	0.0	0.0	.040	
5	4	1	1	.003	0.0	0.0	.007	
			2	.009	0.0	0.0	.018	

B. Data on Weapons Now on Target (As Candidates for Possible Deletion)

NW	IG (NW)	VTD (NW)	J	SIGD (NW,J,NI) NI=1	NI=2	NI=3	NI=4
1	2		1	0.0	0.0	- .105	
			2	0.0	0.0	-1.32	
2	3		1	0.0	- .274	- .052	
			2	0.0	-1.560	- .660	
3	3		1	0.0	- .274	- .052	
			2	0.0	-1.560	- .660	

Just as the information in table 8-A is used to provide values of SIG for computing VTP(G), the array SIGD in table 8-B is used to provide values of SIG for the computation of VTD(NW). This table contains an entry for each weapon currently on the target. The array IG(NW) in this case indicates that three weapons have been assigned, first from group 2, then group 3, finally another from group 3. The role of SIGD exactly parallels SIGP; that is, to obtain the potential value of SIG if a weapon were deleted SIGD is added to SIG in each column. Since SIGD is negative, this has the effect of cancelling out the cross terms for the weapon that would be removed. Of course, if removal of the weapon would reduce the number of weapons in a time-of-arrival column to 0, the following columns would be spaced back to avoid unnecessary columns.

In summary, table 7 contains the scratch pad data used to calculate the actual payoff. Table 8-A contains the corrections SIGP for SIG needed to calculate the corresponding weapon-added estimate VTP(G) for each weapon group G. Table 8-B contains the corrections SIGD for SIG needed to calculate the weapon-deleted estimate VTD(NW) for each weapon NW now on the target.

These arrays SIGP and SIGD are kept continuously up-to-date as weapons are added and deleted. For example, as illustrated in table 8, the last weapon added was from group 3. Thus the last set of cross terms computed would have been the cross terms between group 3 and every other group. These cross terms are shown in table 8-A in the array DSIG(G,J). When the last weapon from group 3 was added, these terms were computed, and for each weapon G they were added into the array SIGP in all time of arrival columns where both the weapon G and the weapon 3 would be present. In the array SIGD the same quantity:

    DSIG (IG(NW),J)

is subtracted out for each column where both weapons are present, thus removing the contribution of the weapon, IG(NW), to SIG in the calculation of VTD(NW).

Whenever it is decided to add or delete a weapon from a group, KG, the local subroutine CALSG is called to calculate the array DSIG(G,J) to obtain the cross terms between KG and all potential weapon groups G. CALSG is contained in statements 100 through 108 of WAD.

Table 9 (only partly filled out) illustrates some of the input data required by WAD. The array RISK(IAT,G,J) is used in calculating the cross terms DSIG. However, those elements contribute where the particular attribute (class, type, etc.) is shared.

The shared attributes between weapon groups G1 and G2 are determined by checking whether JATTRIB(IAT,G1) = JATTRIB(IAT,G2).

The flowchart for WAD consists of 14 parts. Part I shows the overall flow of the subroutine. The main processing by the subroutine is

283

Table 9. Illustrating Quantities Pre-Calculated for Each Potential Weapon Before WAD is Called

G	INACTIVE (G)	TOA (G)	TVAL-TOA (G)	PEX (G)	MORR (G)	J	VTOA (G,J)	MUP (G,J)	SSIG (G,J)	LAT=1 ALL	LAT=2 GROUP	LAT=3 REGION	LAT=4 CLASS	LAT=5 TYPE	LAT=6 ALERT
1	0	.38	50			1	40.0								
						2	10.0								
2	0	.54	40.0			1	32.0								
						2	8.0								
3	0	.23	60.0			1	48.0								
						2	12.0								
4	0	.55	39.7			1	31.8								
						2	7.9								
5	2	1.5	10.0			1	8.0								
						2	2.0								

controlled by one of the control programs depending on the WADOP option chosen (initialize, add, or delete). The following three parts each illustrate the operation of one of these control programs. Each control routine utilizes a number of other local subroutines, as well as external routines (see figure 56).

The Add Weapon Control routine (Part III) will be used as a vehicle to illustrate the operation of the program. Once the operations of this routine are understood, the corresponding operations in the other routines should be obvious.

The routine first checks to be sure that the number of weapon additions and weapon deletion operations, IOP, on this target does not exceed 100. If it does, it is assumed that STAIL and WAD are caught in an endless loop probably repeatedly adding and deleting the same weapon, so the processing of the target is terminated. In principle, such looping should not occur. However, it has been found that errors in reading file data for one target, or a random machine malfunction, or inconsistencies in the data supplied to the program can sometimes result in such a situation. This makes it possible for the program to proceed to the next target rather than aborting the entire run. However, when this happens, the LOOP flag is set nonzero. This causes the print in statement 41 to appear during the initialization of every succeeding target, so that the user is sure to notice that the difficulty occurred.

However, assuming that no such loop has occurred, the routine adds the Lagrange multiplier for the weapon added to the COST of the allocation and also updates SUMPREM, the sum of the premiums for the target. (This last variable SUMPREM, as well as the variables TBENEFIT and TPMX near statement 14, are computed only to provide a consistency check on the treatment of premiums. These variables are not essential to the operation of the program.) Notice that these variables and almost all other variables used by WAD such as VT, VTP, VTD, PAYOFF, PROFIT, etc. are computed (even for multiple targets) as if the program were dealing only with a single simple target. It is necessary to take target multiplicity into account only when dealing with variables which accumulate the total cost, total payoff, or total consumption of specific weapon groups over the whole target system.

The PREMIUMs used by the allocator, however, depend on just such a variable -- namely SURPWP(G), the available surplus (positive or negative) of unused weapons in each group G. Consequently, when a weapon G is added, the PREMIUM for that weapon group must be recalculated. Before it is recalculated, SURPWP(G) must be revised as it is in statement 7 to reflect the multiplicity of the target; i.e.:

$$SURPWP(G) = SURPWP(G) - CTMULT$$

The variable CTMULT, current target multiplicity, is used rather than the initial multiplicity TGTMULT, because when PROGRESS is equal to 1.0 a multiple target can be split into several parts of reduced multiplicity.

The local subroutine SPLIT (Part VI) which begins with statement 9 is responsible for determining if such a split is needed, and for carrying out the adjustment of bookkeeping on the arrays SURPWP and PREMIUM if it is. To avoid unnecessary computation by WAD, SPLIT is designed to minimize the number of times multiple targets are split up. The intent is to avoid separating multiple targets, unless retaining the full multiplicity of the target during the allocation of the weapon indicated would cause a weapon surplus > .5 weapons. Such a change in SURPWP should cause the step premium for the groups to change from positive to negative. Obviously it would be a mistake to keep allocating as if the same premium would apply. Therefore, when this happens the target is split into two parts. One part, CTMULT, containing the largest multiplicity that could be allocated a weapon from one group without causing the premium to go negative, the other part, CTSPILL, containing the remainder.

After this is done, it is necessary to correct the value of SURPWP and recompute the premiums. To understand what is required to do this, we must recall that when MULCON began the allocation to the target on this pass, it removed the weapons previously assigned to all the elements of the multiple target and thus increased SURPWP by the multiplicity for each weapon previously assigned. If a decision is now to be made that only part of the target is to be dealt with, the old allocation should in effect be restored for the remainder of the target elements. Thus, in statement 911, SURPWP is decreased by the change in multiplicity for each weapon previously assigned. Conversely, while the present allocation was proceeding, SURPWP was being decremented by the multiplicity for each weapon assigned. If we now intend to interpret the allocation as applying only to a part of the total multiplicity, then the value of SURPWP must be increased as as in statement 908 by the change in multiplicity for each weapon already assigned. Finally, since the value of PREMIUM(G) depends not only on SURPWP(G) but also on CTMULT for the target all premiums are recomputed (statement 930). This completes the bookkeeping corrections made by SPLIT. A slightly different version of SPLIT beginning at statement 21 is used by the deletion control routine. In this case the question is whether deletion of the weapon group for the full multiplicity would cause a deficit > .5. Aside from this obvious change in sign, however, the operation is essentially identical.

In both versions of SPLIT, subroutine ADDSAL is used to modify the stockpile for salvoed missile weapons.

When SPLIT has completed its work, the program proceeds as usual to update SURPWP for the weapon now being added and PREMIUMS(G) is called as usual.

Both the add and delete weapon processes exit WAD through statement 12. At this statement, WAD calls: 1) ADDSAL to update the salvoed weapon stockpile; 2) NEWSAL to determine if the preferred salvo is still available, and 3) WADOUT to calculate the decision variables for STALL. After these calls, WAD updates the payoff, cost, profit and benefit variables.

286

The main functioning of WAD, however, proceeds after the stockpiles have been set and premiums calculated for each weapon group. This function consists of updating all the arrays in tables 7 and 8 to reflect addition of the weapon G. The process is accomplished by calling a series of local subroutines.

First CALSG (Part VII) is called to compute the cross term array, DSIG, in table 8-A. Then ADDSIG (Part VIII) is called to update the arrays SIGP and SIGD. For maximum efficiency, the revision of the arrays, SIGP and SIGD is done one column at a time, working from right to left and dealing only with those columns affected by the weapon G. If the weapon G adds a new column, the column NI = NWRT where the augmented results are written will be displaced one column to the right of NI = NRD where the original data were read. However, before actually updating these two arrays in each column, the data currently in SIGP in that column are used to update SIG in table 7. Since this value of SIGP was required to add the weapon G, the negative of it would be required to delete it. Consequently, at the same time SIGP is simply negated and stored in SIGD to produce a new row (NW = 4) for the weapon G in table 8-B. All of these operations required to update SIG, SIGP, and SIGD are done in series for the same read and write columns, one column at a time until the updating is complete.

Next, ADDIND (Part IX) is called to update all the indices to reflect the addition of the new weapon. The indices which must be updated are in table 7 NUM, NTOA, NUWP, VAL, in table 8 ITOA, IADDTOA.

When this has been done the payoff computations begin. CALPAY (Part XII) is called first to calculate the actual payoff (table 7). Since this involves on y one calculation it is done in a very straightforward way and is completely recalculated for all columns. Thus, the subroutine CALPAY is very straightforward and easy to understand.

The updated information in table 7 is then used as a basis for calculating the weapon added payoffs (CALPOT, Part XIII) and the weapon deleted payoffs (CALDEL, Part XIV). To avoid any need for additional arrays to store intermediate results these calculations are done (both by CALPOT and CALDEL) one weapon group and one hardness component at a time working straight down the lists in table 8. For each group and each hardness component the calculations then work from left to right dealing only with columns affected by the new weapon. For the columns in table 7 that would be affected, revised values S1 of S and VSN1 and VSN are computed working toward the right to obtain the final values which can be added to obtain the revised value VTD for the weapon deleted calculations.

When all payoffs have been calculated, WADOUT is called summarizing the results for STALL, and finally the actual PROFIT, and PAYOFF, BENEFIT, etc., are computed and stored so that they are available to be printed if such information is requested. WAD then returns control to STALL.

The control routine for weapon deletion exactly parallels the above procedures except that SUBSIG and SUBIND replace ADDSIG and ADDIND. SUBSIG parallels ADDSIG almost exactly except that the order of processing columns is reversed to avoid writing over essential data as columns are spaced back. SUBIND differs from ADDIND mainly in that indices are decremented instead of incremented.

The initialize control routine (part II), and the local subroutine INITIALIZE (part V), are concerned with establishing the starting state for the allocation to a target. After the discussion of the add and delete routine, the flow diagrams should be self-explanatory. However, a couple of comments may be appropriate. For computing efficiency the initialization is limited to the minimum required to provide the starting state. Many cells in SIGP and SIGD are not initialized and irrelevant data will remain in many cells. Thus on targets after the first target during the allocation, many cells shown in table 8 with irrelevant zeros, may in fact contain irrelevant data that will not be referenced. Particular attention is called to rows containing inactive weapons. These rows will not be reinitialized at all and thus will contain much irrelevant data for prior targets.

On the initialization call, the only nontrivial payoffs that need to be computed are the potential weapon payoffs VTP(G). Since at this stage these all involve only one weapon, the correlation cross terms are irrelevant. Consequently the simple formula,

$$S = FMUP\ (+ MUP(G))$$

can be used.
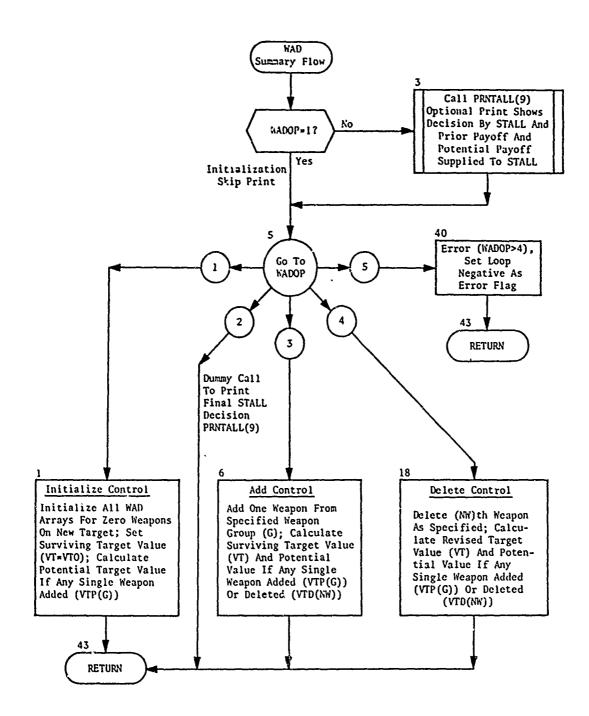
Figure 56.   Subroutine WAD
Part I: Summary Flow

289
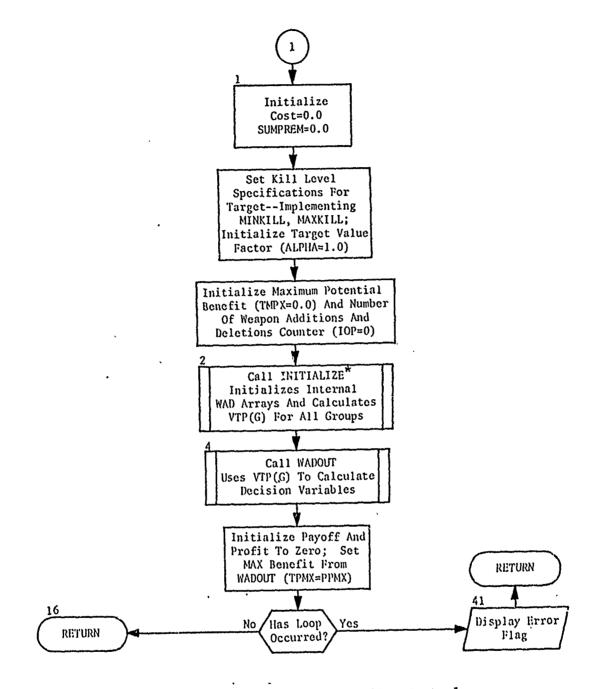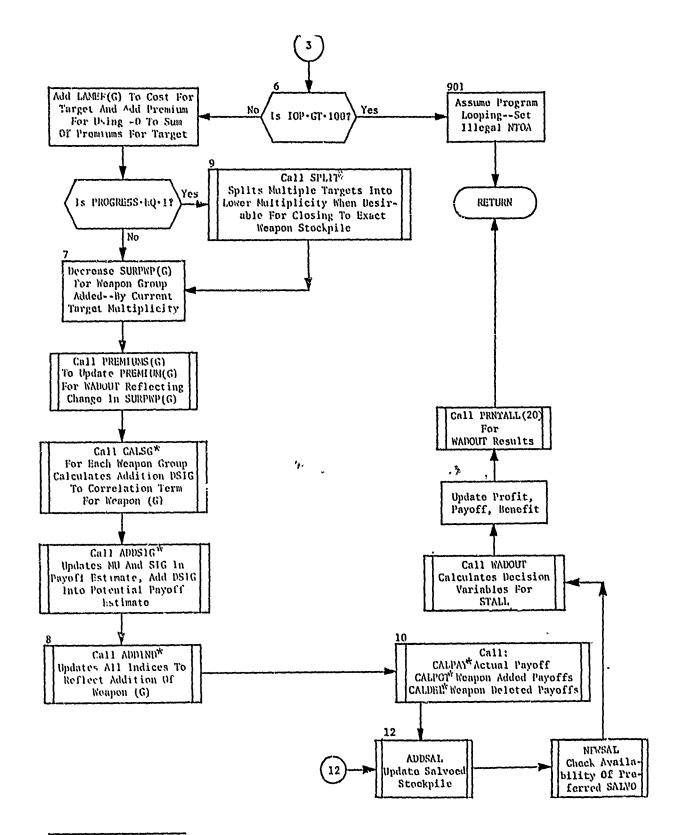
Figure 56. Part II: Initialize Control

---
*Local Subroutine
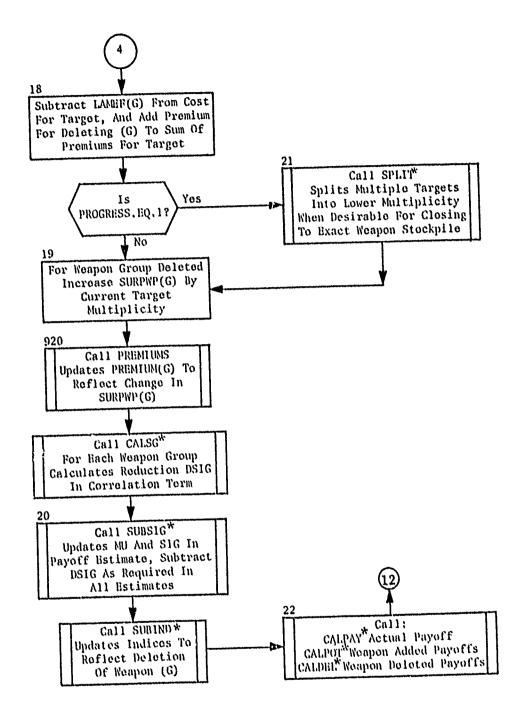
Figure 56. Part III: Add Weapon Control

291

④

18
Subtract LAMIF(G) From Cost
For Target, And Add Premium
For Deleting (G) To Sum Of
Premiums For Target

Is
PROGRESS.EQ.1?   **Yes** →

21
Call SPLIT*
Splits Multiple Targets
Into Lower Multiplicity
When Desirable For Closing
To Exact Weapon Stockpile

No

19
For Weapon Group Deleted
Increase SURPWP(G) By
Current Target
Multiplicity

920
Call PREMIUMS
Updates PREMIUM(G) To
Reflect Change In
SURPWP(G)

Call CALSG*
For Each Weapon Group
Calculates Reduction DSIG
In Correlation Term

20
Call SUBSIG*
Updates MU And SIG In
Payoff Estimate, Subtract
DSIG As Required In
All Estimates

⑫

22
Call:
CALPAY* Actual Payoff
CALPOT* Weapon Added Payoffs
CALDEL* Weapon Deleted Payoffs

Call SUBIND*
Updates Indices To
Reflect Deletion
Of Weapon (G)

---

*Local subroutine

Figure 56.   Part IV: Delete Weapon Control
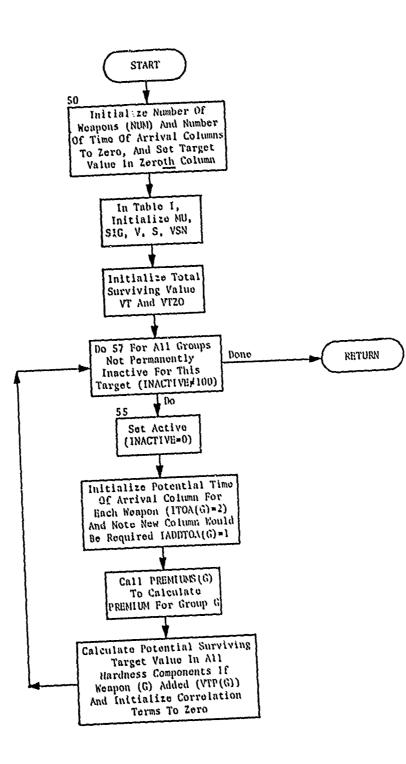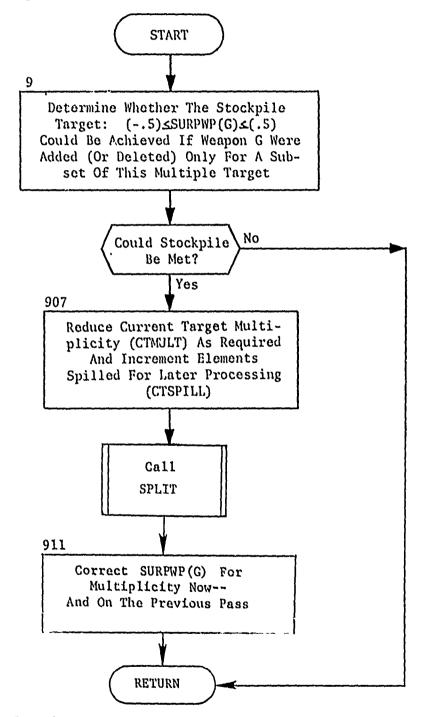
Figure 56. Part V: Local Subroutine INITIALIZE

293

[Called in Closing Phase of Allocation, Before Adding (or Deleting*)
a Weapon--Splits Multiple Targets into Lower Multiplicity When It
Will Help to Meet Exact Stockpile]

```
                           ┌─────────────┐
                           │    START    │
                           └─────────────┘
                                  │
  9                               ▼
      ┌──────────────────────────────────────────────┐
      │   Determine Whether The Stockpile             │
      │    Target:  (-.5)≤SURPWP(G)≤(.5)              │
      │   Could Be Achieved If Weapon G Were          │
      │   Added (Or Deleted) Only For A Sub-          │
      │      set Of This Multiple Target              │
      └──────────────────────────────────────────────┘
                                  │
                                  ▼
                         ╱─────────────────╲   No
                        ⟨  Could Stockpile   ⟩────────────────┐
                         ╲    Be Met?       ╱                 │
                          ╲───────────────╱                   │
                                  │ Yes                       │
  907                             ▼                           │
      ┌──────────────────────────────────────────────┐       │
      │    Reduce Current Target Multi-               │       │
      │   plicity (CTMULT) As Required                │       │
      │      And Increment Elements                   │       │
      │    Spilled For Later Processing               │       │
      │            (CTSPILL)                          │       │
      └──────────────────────────────────────────────┘       │
                                  │                           │
                                  ▼                           │
                     ╔═══════════════════════╗                │
                     ║        Call           ║                │
                     ║        SPLIT          ║                │
                     ╚═══════════════════════╝                │
                                  │                           │
  911                             ▼                           │
      ┌──────────────────────────────────────────────┐       │
      │    Correct  SURPWP(G)  For                    │       │
      │      Multiplicity Now--                       │       │
      │    And On The Previous Pass                   │       │
      └──────────────────────────────────────────────┘       │
                                  │                           │
                                  ▼                           │
                           ┌─────────────┐                    │
                           │   RETURN    │◄───────────────────┘
                           └─────────────┘
```

*
Separate Copy in Program with Change of Signs for Weapon
Deletion.

Figure 56.   Part VI: Local Subroutine SPLIT
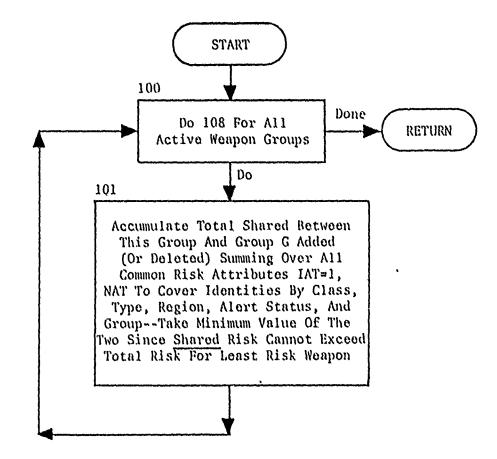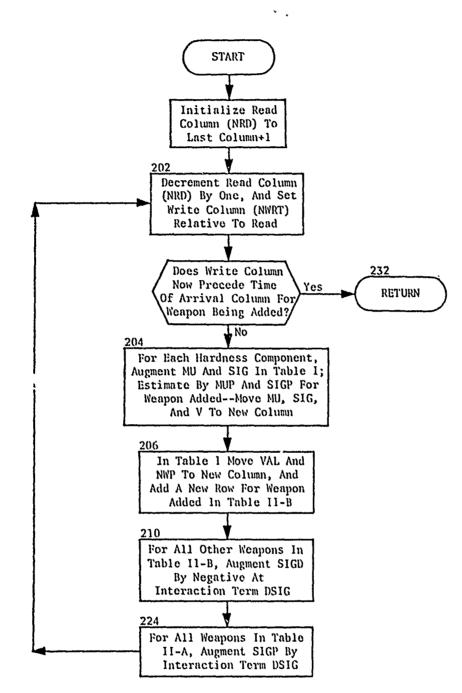
Figure 56. Part VII: Local Subroutine CALSG

295

```
                          ┌──────────┐
                          │  START   │
                          └────┬─────┘
                               │
                               ▼
                    ┌──────────────────────┐
                    │  Initialize Read      │
                    │  Column (NRD) To      │
                    │  Last Column+1        │
                    └──────────┬───────────┘
              202              │
                    ┌──────────▼───────────┐
                    │  Decrement Read Column│
                    │  (NRD) By One, And Set│
           ┌───────▶│  Write Column (NWRT)  │
           │        │  Relative To Read     │
           │        └──────────┬───────────┘
           │                   │
           │                   ▼
           │            ╱──────────────╲            232
           │          ╱  Does Write Column ╲      ┌──────────┐
           │         │   Now Precede Time    │ Yes │          │
           │         │  Of Arrival Column For├────▶│  RETURN  │
           │          ╲ Weapon Being Added? ╱      └──────────┘
           │            ╲──────┬───────╱
           │       204        │ No
           │        ┌──────────▼───────────┐
           │        │ For Each Hardness Component,│
           │        │ Augment MU And SIG In Table I;│
           │        │ Estimate By MUP And SIGP For │
           │        │ Weapon Added--Move MU, SIG,  │
           │        │ And V To New Column          │
           │        └──────────┬───────────┘
           │       206         │
           │        ┌──────────▼───────────┐
           │        │ In Table 1 Move VAL And│
           │        │ NWP To New Column, And │
           │        │ Add A New Row For Weapon│
           │        │ Added In Table II-B    │
           │        └──────────┬───────────┘
           │       210         │
           │        ┌──────────▼───────────┐
           │        │ For All Other Weapons In│
           │        │ Table II-B, Augment SIGD│
           │        │ By Negative At         │
           │        │ Interaction Term DSIG  │
           │        └──────────┬───────────┘
           │       224         │
           │        ┌──────────▼───────────┐
           │        │ For All Weapons In Table│
           └────────┤ II-A, Augment SIGP By  │
                    │ Interaction Term DSIG  │
                    └────────────────────────┘
```

This subroutine updates values for MU and SIG in all Time of Arrival
columns affected by new weapon G. If weapon G adds a new column, it
also moves the data one column to the right by writing in a column
with an index one larger than the index of the column being read.

Figure 56.   Part VIII: Local
                Subroutine ADDSIG

296

Figure 56. Part IX: Local
Subroutine ADDIND

297

START

300
Initialize Read (NRD)
To Time Of Arrival
Column (ITOA) Of
Weapon Group Deleted

302
If Weapon Deleted Is
Only One In Its Column
(ITOA), Set Write Column
NWRT=NRD-1 To Delete

Does NWRT Precede
ITOA Of Group
Deleted?

No →

305
For Each Hardness Component
Decrement MU And SIG In Table 14
For Weapon Deleted--Space Back
In TOA Column If Necessary

Yes ↓

1306
If This Group Had Same
ITOA As Group Deleted, Then
Update SIGP In Tables 15-A
And 15-B And Store In Prior
Column To Preserve Data On
Lost ITOA Column

In Table 14, Move
.VAL And NWP Back One
Column If Necessary

310
In Table 15-A, Decrement SIGP
By DSIG For All Columns After
And Including Affected Column--
Space Back If Column Removed

314
In Table 15-B, Decrement SIGD
By DSIG For All Columns After
And Including Affected Column--
Space Back If Column Removed

322
Reading Past
Last Column?

No ←

Yes → RETURN

304
Increment Read
Column (NRD)

318
In Table 15-B, Keep Table Com-
pact By Spacing IG And SIGD
Data Below Row For Deleted
Weapon Up One Row

Figure 56. Part X: Local
Subroutine SUBSIG

298

Figure 56. Part XI: Local
Subroutine SUBIND

```
                          ┌─────────┐
                          │  START  │
                          └────┬────┘
                               │
                       400     ▼
┌──────────────┐   Done  ┌──────────────┐
│ Store Total  │◄────────│  Do 406 For  │◄──────────────────┐
│ Surviving    │         │ Each Hardness│                   │
│ Value (VT)   │         │ Component (J)│                   │
└──────┬───────┘         └──────┬───────┘                   │
       │                        │ Do                        │
       ▼                        ▼                 406        │
┌──────────────┐         ┌──────────────┐   Done  ┌──────────────┐
│    RETURN    │    ┌───►│   Do 405     │────────►│  Accumulate  │
└──────────────┘    │    │ From First   │         │ Total Over   │
                    │    │ To Last TOA  │         │Components (J) │
                    │    │  Column (N)  │         └──────────────┘
                    │    └──────┬───────┘
                    │           │ Do
                    │           ▼
                    │  ┌─────────────────────────┐
                    │  │ Calculate Survival Probability│
                    │  │   Through This Time Of  │
                    │  │ Arrival (S(N,J));  Apply This│
                    │  │ Survival Probability Only To│
                    │  │ That Value Available Now But│
                    │  │ Not At Next TOA Interval And│
                    │  │ Accumulate Surviving Values│
                    │  └────────────┬────────────┘
                    │               │
                    │               ▼
                    │       ┌──────────────┐
                    └───────│ Calculate Zeroth│
                            │ Order Survival│
                            │  Value VTZO  │
                            └──────────────┘
```

Calculates Actual Surviving Target Value, VT:

$$VT = \sum_{\substack{J= \\ (1,M)}} \sum_{\substack{N= \\ (1,NTOA)}} S(N,J) * [V(N,J) - V(N+1,J)]$$

and Zeroth Order Surviving Value, VTZO;

$$VTZO = \sum_{J} S(NTOA,J) * VO(J)$$

Figure 56.   Part XII: Local
             Subroutine CALPAY

300

START

501

Do 512 For
All Active
Weapon Groups

Done

RETURN

Do

502

Initialize Potential
Target Value
VTP=0

Accumulate Potential
Value Surviving

$VTP = \sum VSN1$

Done

Do 510 For
Each Hardness
Component

Do

503

If This Weapon Would Add New
TOA Column, Calculate
Revised Value VS And VSN
For Preceding Column

504

For Each TOA Column After And
Including That For This Group G,
Calculate Revised Probability
(S1); If This Weapon Added
(Increment MU And SIG Of CALPAY
By MUP And SIGP For This Weapon)
--Accumulate Revised Survival
Value VSN1 For This Component

Figure 56.   Part XIII: Local
Subroutine CALPOT

301

Figure 56. Part XIV: Local
Subroutine CALDEL

302

3.11.9 Subroutine WADOUT

PURPOSE: This routine summarizes decision alternatives for STALL by combining payoff data produced by WAD with the weapon cost data (Lagrange multipliers minus premiums). It also contributes to the efficiency of WAD by making inappropriate weapons inactive.

ENTRY POINTS: WADOUT

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, C33, CONTRO, DYNAMI, PREMS, SALVO, WADFIN, WADOTX, WADWPN, WPFIX

SUBROUTINES CALLED: PRNTALL

CALLED BY: WAD

Method:

The output data produced by WAD consist of the following parameters for STALL that are recorded in /WADOTX/:

> PPMX and IPPMX - the maximum potential increase in effective profit for any single weapon; and the index G to that weapon group, respectively.

> PVRMX and IPVRMX - the maximum effective efficiency of any potential weapon; and the index G to that weapon group, respectively.

> DPMN and IDPMN - the minimum marginal effective profit for any weapon now assigned; and the index NW to that weapon in the list of weapons assigned, respectively.

It also produces the array INACTIVE(G) in /WADWPN/ which is used by WAD to determine which weapons groups need not be processed.

The input data from WAD consist of:

> VT - the surviving target value in /C33/

> VTD(N) - the potential weapon deleted surviving target value (also in /DYNAMI/ (equivalenced to RVAL)

> VTP(G) - the potential weapon-added surviving target value in /WADFIN/

The input data on weapon costs consist of LAMEF(G), PREMIUM(G), and DPREMIUM(G).

WADOUT also initializes VTMAX and VTMIN of /WADOTX/, and MAXCOST which reflect the MINKILL, MAXKILL specifications for each target.

The quantities ALPHA and VTEF of /WADOTX/, are essentially local variables for WADOUT. They are included in this common block for use by PRNTNOW, and in the case of ALPHA, to allow WAD to reinitialize it to 1.0 for each new target.

The flow diagram, figure 57, is in three parts. In part I, the user-input parameter IMATCH is used to determine the method of computing MINKILL and MAXKILL. If IMATCH is 0, then the damage calculations used to determine residual target value for purposes of MINKILL and MAXKILL. use time dependence of target value. If IMATCH is nonzero, then MINKILL and MAXKILL are computed relative to the original target value. In addition, if IMATCH is 100, then the routine prints the WADOUT variables, VTO, VT, VTZO (original target value), FLGMN, FLGMX, SVTMIN, SVTMAX, VTMIN, VTMAX, and ALPHA.

The Do loop ending at statement 18 is used to flag all groups with fixed weapons on this target as active. This prevents their being removed during processing and maintains the validity of the damage calculations.

In Part II, the processing begins by evaluating the actual effective surviving target value VTEF. It then scans all weapons currently assigned to calculate the output quantities DPMN and IDPMN. If any weapon now on target fails to destroy a fraction of the original value greater than MINDAMAG, the weapon is flagged for immediate removal (statement 15). At the same time, the groups already assigned are flagged with INACTIVE = -100 to eliminate any possibility that they would be erroneously set inactive. (WADOUT never exits with INACTIVE set negative. A weapon group flagged with a -100 is always reset to INACTIVE = 0 before the routine exits.) (Do 14 loop)

Basically, the processing in part III is concerned with scanning all potential weapons to calculate the output quantities PPMX, IPPMX, PVRMX, and IPVRMX. Any weapon which would fail to destroy a fraction of the original value greater than MINDAMAG will be ignored in these calculations. Thus, it could never be allocated to the target.

In addition, if a weapon is a salvoed missile but no salvo is available (i.e., MYSAL(G) less than or equal to zero), or if the group has no weapons, then it is not considered as a potential weapon.

At the same time, however, the values for the array INACTIVE(G) are established. The INACTIVE array for each target is permanently stored on the WPNTGT files, as it was originally computed by GETDTA, with only two values -- zero for weapons in range of the target, and 100 for

304

weapons out of range. Consequently, when these data are read on successive passes for each target, these initial values are automatically restored.

External to WADOUT the INACTIVE array is treated as if it has only two values - zero for active weapon groups, and nonzero for inactive groups. WADOUT makes weapons temporarily inactive relative to a specific target by setting INACTIVE equal to either 2,000 or 30,000.

If WADOUT exits with either value, 2,000 or 30,000, the weapon is treated as temporarily inactive in exactly the same way. The difference between 2,000 and 30,000 is relevant only if WADOUT recycles without exiting.

WADOUT will recycle if, after all potential weapons are examined, it is found (upon return to Part 2) that there are no potential weapons that appear profitable and, moreover, that the required kill probability, MINKILL, has not been achieved. In this case the value of ALPHA is increased to make the target seem more valuable and the evaluation is repeated. When this occurs, weapons tentatively set to INACTIVE = 30,000 are reset to 0 and the decision to inactivate them is reexamined.

If WADOUT exits with INACTIVE = 30,000, it is always set to 2,000 if WADOUT is called again.

The operation of Part III of WADOUT can now be summarized as follows. In the execution of this Do loop inactive weapons, INACTIVE = 100, 2,000 and 30,000 (30,000 except on a recycling pass), are skipped. All active weapons, INACTIVE = 0, -100 (or 30,000 on a recycling pass), are evaluated. Those for which INACTIVE is no, negative are then considered to determine whether they should be made inactive.

This consideration (lower half of the flowchart) is as follows:

a. Any weapon that still shows a positive potential profit remains active.

b. Any weapon which does not show a positive profit against the full target value (in the absence of other weapons) is made inactive.

c. If there are other weapons on the target the weapon remains active unless its efficiency is less than .1. This reduces the chance that an inactive weapon could become profitable if some other weapon currently on the target were removed.

d. If the efficiency is less than .1, it is made inactive unless there are already unprofitable weapons assigned. In this case the decision to make it inactive is postponed until these unprofitable weapons are removed.

305
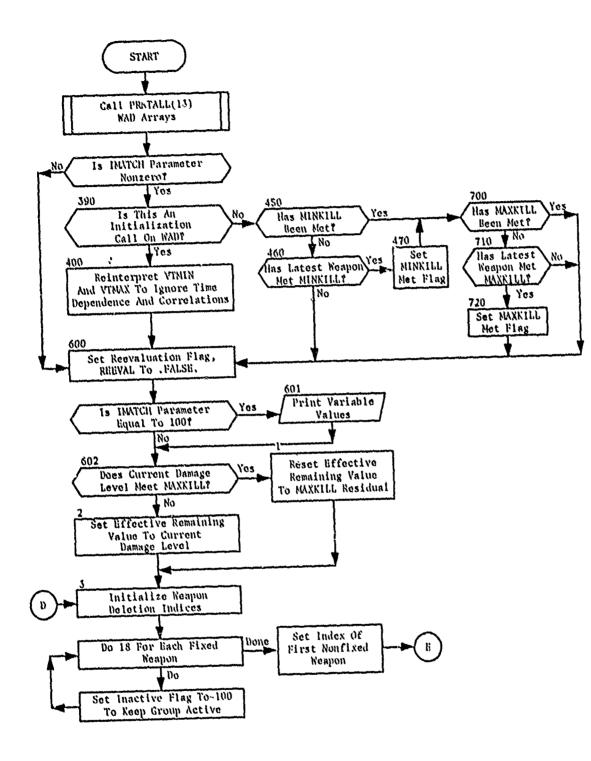
Figure 57. Subroutine WADOUT
(Part 1 of 5)

```
                                    ┌───┐
                                    │ E │
                                    └─┬─┘
                                      │
                                      ▼
        ┌─────────────────────────────┐  Done   ┌───┐
        │ Do 14  For  All Remaining   ├────────►│ A │
   ┌───►│ Weapon Groups Assigned      │         └───┘
   │    └─────────────┬───────────────┘        To Scan Potential
   │                  │                        Weapons For BENEFIT
   │                  ▼                        And Set Some Inactive.
   │    ┌─────────────────────────────┐        If Appropriate
   │    │ Tag Groups Assigned To      │
   │    │ Remain Active,              │
   │    │ Set INACTIVE = -100         │
   │    └─────────────┬───────────────┘
   │  31              │
   │    ┌─────────────▼───────────────┐
   │    │ Set "Effective"             │
   │    │ Value If Weapon Deleted,    │
   │    │ VTDEF = VTMIN               │
   │    └─────────────┬───────────────┘
   │  33              │
   │    ┌─────────────▼───────────────┐
   │    │ Calculate BENEFIT From      │
   │    │ Weapon (Difference In       │
   │    │ Effective Value Minus       │
   │    │ Premium For Deleting It)    │
   │    └─────────────┬───────────────┘
   │                  │                         15
   │             ╱────▼─────╲                 ┌──────────────┐
   │            ╱ Does Weapon ╲               │ Set Large    │
   │           ╱ Destroy Fraction Of ╲   No   │ Negative     │
   │           ╲ Value Greater Than  ╱───────►│ "Profit"     │
   │            ╲  MINDAMAG?  ╱                │ For Weapon   │
   │             ╲────┬─────╱                  └──────┬───────┘
   │                  │ Yes                           │
   │  16              ▼                               ▼
   │    ┌─────────────────────────────┐             ┌────┐
   │    │ Calculate "Profit"          │             │ 14 │
   │    │ BENEFIT-COST For            │             └────┘
   └────┤ Weapon  And Note            │
        │ Weapon  IDPMN With          │
        │ Minimum  Profit DPMN        │
        └─────────────────────────────┘
```

Figure 57. (Part 2 of 5)

307

Figure 57. (Part 3 of 5)

Figure 57.　(Part 4 of 5)

309

Figure 57. (Part 5 of 5)

310

3.12 Subroutine DEFALOC[*]

PURPOSE: The purpose of this subroutine[*] is to allocate
                   missiles to an individual target which is defended
                   with terminal ballistic missile interceptors (i.e.,
                   MISDEF > 0).

ENTRY POINTS: DEFALOC

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C30, C33, CONTRO, DEFCOM, DEFRES, DYNAMI,
                   PAYSAV, PREMS, SALVO, SURPW, WADFIN, WADOTX,
                   WADWPN, WEPSAV, WPFIX

SUBROUTINES CALLED: ADDSAL, HEAD, INITSAL, LAMGET, NEXTTT, PREMIUMS,
                   PRNTALL, RESTORE, RESVAL

CALLED BY: MULCON

Method:

When MULCON has read in data associated with a new target, it examines
MISDEF to determine if the target is defended with terminal ballistic
missile interceptors. If MISDEF = 0, indicating no defenses, it pro-
ceeds to call STALL for the allocation of weapons. If MISDEF > 0, then
there are terminal interceptors present; DEFALOC is called after call-
ing STALL to allocate the missiles to the target, and the most profit-
able allocation (STALL or DEFALOC) is chosen.

The input variables describing the target's local ABM capability allow
uncertainties to be introduced in the number of interceptors present.
MISDEF is the nominal number of interceptors on the target, each with
kill probability PKTX against unhardened warheads, and RADPK is the ran-
dom area defense kill probability. In addition, four other parameters
are defined (the same for all targets) which introduce uncertainties in
MISDEF. RX(1) (input as LOWFAC) is a factor which, when multiplied by
MISDEF, gives a lower estimate of interceptors which has probability
PX(1) (input as PROBLOW) of occurring. Likewise, RX(2) (input as
HIGHFAC) and PX(2) (input as PROBHIGH) define the overestimate of in-
terceptor availability. Thus, if there is imperfect knowledge of the
terminal ABM capability, the allocator can hedge against these uncer-
tainties when assigning weapons.

In addition to the target-associated ABM data, it is possible to de-
scribe penetration aids suitable for the various missiles by means of
the payload table. For a particular payload index IPAY, the following
variables are defined which describe the terminal missile defense pene-
tration aids:

---

[*] This subroutine is the first of segment DEFAL.

311

NWHD   =   Number of warheads per independent reentry vehicle.

NTDECOYS =   The number of aim points the terminal defense sees
                   for each independent reentry vehicle (in addition
                   to the warheads). These are terminal decoys.

XDEG   =   A factor by which the PKTX is multiplied to obtain
                   terminal interceptor kill probability against this
                   weapon type. It reflects additional hardening of
                   the warhead or electronic penetration aids which
                   can degrade interceptor effectiveness.

The first decision in DEFALOC concerns the verification pass. If PROGRESS
= 2 and IVERIFY = 2, the current call on DEFALOC is a part of verifica-
tion pass to determine the effect of a new correlation factor. Since
DEFALOC does not consider interweapon correlations, no processing is done
in this case.

Before allocating any missiles, DEFALOC determines if STALL has been
called. STALL will not be called if the number of fixed assignments
exceeds 30. If STALL was called, the surplus weapon indicators SURPWP
are reset (statement 76) as if STALL were not called. This procedure
provides for the correct premium computations in RESVAL.

As part of the initialization, DEFALOC calls INITSAL to set the arrays
in block /SALVO/ for this target. In removing the weapons allocated by
STALL, routine ADDSAL was called to restore the salvoed weapons to the
stockpile. After calling INITSAL, DEFALOC computes the maximum number
of weapons that can be allocated from each group because of salvo re-
strictions (NSL(G)). For nonsalvoed groups, NSL(G) is set to the num-
ber of weapons in the group, NWPNS(G). For salvoed groups, NSL(G) is
set to the difference between the current stockpile and the maximum
stockpile. Before PROGRESS = 1.0, the maximum stockpile is 225 more
than the available number. During the verification pass (i.e., PROGRESS
greater than 1.0) NSL(G) is set to 1000 for all groups.

If there are fixed assignments on this target (statement 79), DEFALOC
assigns these weapons first. On the second and later passes (statement
78) the operation is relatively simple. The old allocation contained
in the IG and KORRX arrays of /DYNAMI/ is loaded into the IFW array and
the NOWEP array. The NOWEP array defines the potential DEFALOC alloca-
tion. If the number of fixed assignments exceeds 30, the value of the
KORRX array is equal to the negative of the number of weapons assigned
(see statements 72 to 96). This procedure allows specification of more
than 30 weapons within the 30 elements of the IG array. If the fixed
assignments are less than 30 in number, each entry in the IG array
represents one weapon.

Processing differs if there are fixed assignments on the first pass.
The Do loop starting at statement 97 and continuing to statement 85
allocates each fixed weapon. There are several tests made on proposed

fixes in this loop. First, if the fixed weapon is a bomber, DEFALOC cannot be used since it allocates missiles only. Statement 40 tests if STALL were called. If so, the STALL allocation to this target is used without further DEFALOC processing. If STALL was not called by reason of an excess of 30 fixes, statement 41 prints an error message and the fix request is ignored. If the fixed weapon is a missile, its active flag is tested in statement 50. If the weapon cannot be allocated, an error message is printed* and the request ignored. Otherwise the weapon is assigned. A group counter is kept (statement 86) to determine the number of unique groups represented on target. Statement 91 assigns the weapon. The following statements increase the total cost and number is computed in array ISALFX(G) for all fixed salvoed weapons. For salvoed fixed weapons function LAMGET is used to calculate the weapon cost.

After assigning all fixed weapon requests currently input, DEFALOC determines if there are more requests to be read in the first pass. This is done by continuing to cycle the ASGWPN chain. IFIXEND is used to indicate when this process is complete.

Statement 75 is the exit from the fixed assignment processing. The DO loop from statement 75 to statement 74 places each fixed weapon into the IFW and KORR arrays to be saved for later use. In addition, the expected number of objects (warheads and terminal decoys) perceived by the terminal defense is computed and stored in variable NOBJ. The available number of weapons, NSL, is also documented.

Before allocating any weapons to the target, DEFALOC calculates an approximate maximum rate of return for attacking the target with the best missiles available, using exhaustion tactics (statement 84).

The missile allocation proceeds as follows: first, those missiles with the cheapest terminal objects (warheads and terminal decoys) are allocated until the terminal interceptors are exhausted. Then each missile type in turn is tried to determine which type has the greatest payoff per unit cost when added to this exhausted mix of weapons.

Since the payoff function for a defended target is generally not convex, one cannot look at only the rate of return of the next missile to determine whether the target is profitable to be attacked. Rather, it is necessary to allocate weapons beyond the exhaustion point and then search for that allocation which yields the highest average rate of return. If this average rate is greater than one; i.e., a profit is realized by attacking the defended target, then the allocation can actually proceed.

The exhaustion phase of the allocation is carried out in the statements between statements 400 and 600. The postexhaustion phase starts at statement 600 and continues to statement 2000.

---
* This print includes an indication of the reason for the weapon inactivity. The reason is placed in array MORRX for all inactive weapons.

313
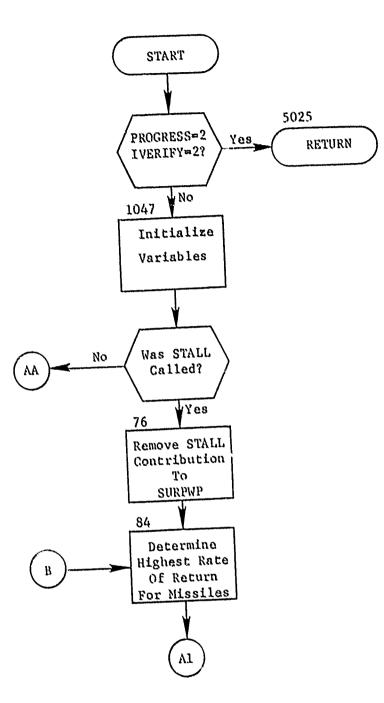
In all calculations of target damage, subroutine RESVAL is called to determine the residual target value (VTDX) for the specific mix of weapons allocated at the time RESVAL is called. Appearance of VTDX in the flowchart implies a call on RESVAL. The program, however, can only allocate missiles from a maximum of 30 groups in total, which must be kept in mind when specifying target defenses. That is, if 30 groups cannot provide sufficient objects to exhaust the defense, this tactic is excluded by the allocator. In addition, only 40 percent of the weapons in any one group can be allocated to a single defended target.

At each stage of allocation, DEFALOC determinys that the number of weapons allocated from any salvoed group does not exceed the number available for allocation. Array NSL is decremented for each allocation. When it reaches zero, all the available weapons are allocated.

When DEFALOC has completed its laydown, it compares the resulting profit to the STALL profit. If the STALL profit is greater, DEFALOC sets NBLN = MISDEF, and restores the STALL allocation. If DEFALOC has a greater profit, it sets NBLN = -MISDEF. If only one of the subroutines has produced an allocation which met the required MINKILL, that allocation is chosen regardless of profit. Then DEFALOC loads the IG and KORRX arrays. First, the fixed weapons are placed in the arrays and then the nonfixed ones. In all cases, the KORRX array contains a negative number corresponding to the number of missiles allocated from the group specified in the corresponding position of the IG array. Subroutine ADDSAL is used to modify the salvoed weapon stockpile.

This subroutine also calculates a modified value for deleting a weapon from the target. This value, DELVT in /WADFINAL/, is used by MULCON to compute the relative damage caused by each allocation, RVAL. For fixed weapons, the value of DELVT is set to the original target value, VTO. For all other weapons, DELVT is equal to the difference between the final residual value, VTDX, of the entire allocation and the residual value if all nonfixed weapons of the same group were removed.

Figure 58 illustrates segment DEFALOC.

Figure 58.   Segment DEFALOC
             Part I: Normal Processing
             (Part 1 of 5)

315

Figure 58. (Part 2 of 5)

Figure 58.   (Part 3 of 5)

Figure 58. (Part 4 of 5)

Figure 58.   (Part 5 of 5)

Figure 58. Part II: Fixed Weapon Assignment
Processing (Part 1 of 2)

Figure 58. Part II: (Part 2 of 2)
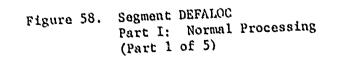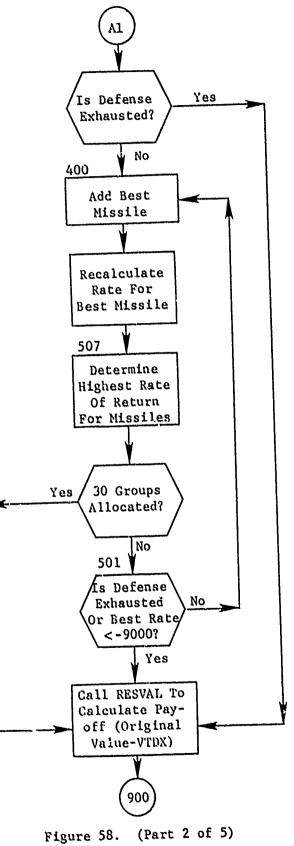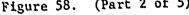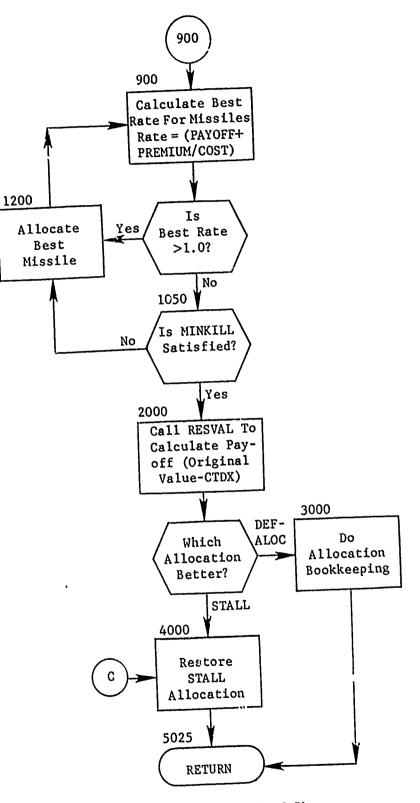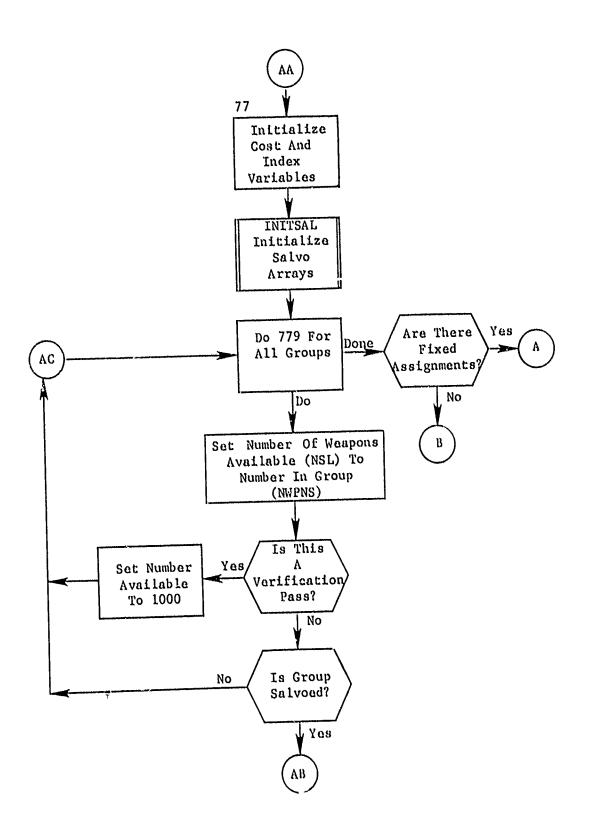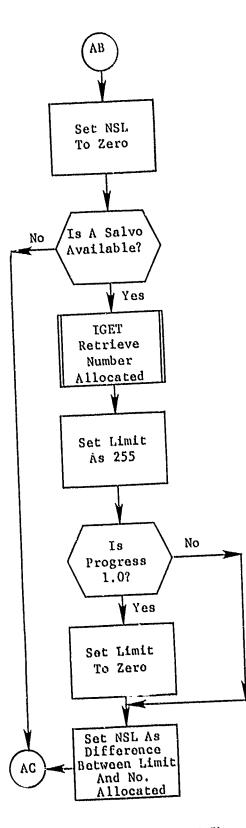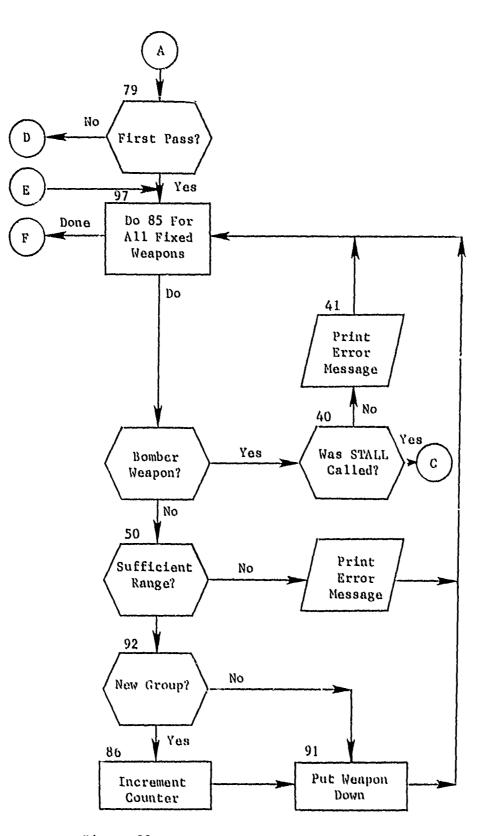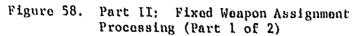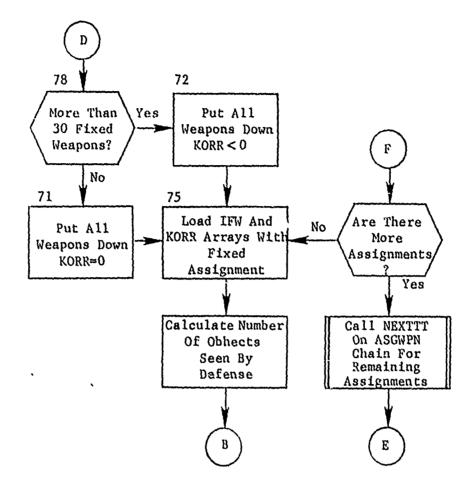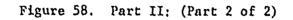
3.12.1 Subroutine PRNTOD

PURPOSE: To produce optional prints for overlay DEFAL.
                        (Options 27 and 28)

ENTRY POINTS: PRNTOD

FORMAL PARAMETERS: IOPT - Print option number

COMMON BLOCKS: C30, DEFCOM, DEFRES
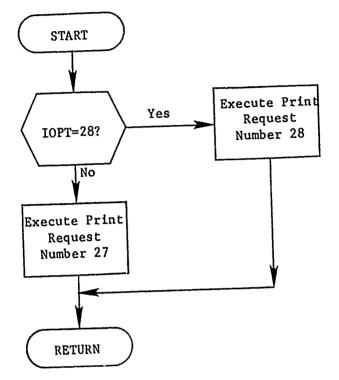
SUBROUTINES CALLED: None

CALLED BY: PRNTNOW

Method:

The formal parameter IOPT determines whether option 27 or 28 appears.
The result of the option appears in the Users Manual, UM 9-77, Volume
III.

Subroutine PRNTOD is illustrated in figure 59.

Figure 59. Subroutine PRNTOD

3.12.2 Subroutine RESVAL

PURPOSE: This routine calculates the surviving value of a
                   target defended with terminal ballistic missile
                   interceptors when attacked with missiles with or
                   without penetration aids.

ENTRY POINTS: RESVAL

FORMAL PARAMETERS: None

COMMON BLOCKS: C30, DEFRES, PAYSAV, TGTSAV, WADWPN, WEPSAV

SUBROUTINES CALLED: FMUP, PRNTALL, TABLEMUP

CALLED BY: DEFALOC

Method:

RESVAL first orders the weapons by time of arrival on the target and
then computes the total number of expected terminal objects contained
in the weapons specified by NOWEP(G). (NOWEP(G) = number of weapons
of group G allocated.) The single shot survival probability of the tar-
get from a weapon from group G on hardness component J is equal to the
previously calculated XMUP(G,J) (common /WADWPN/). This survival pro-
bability must be modified for multiple weapon attacks. As the number
of attackers exceeds the number of defenders, the single shot survival
probability will decrease.

Three different levels of terminal interceptors (NTX(I)) are calculated
for each defended target, and a probability of the occurrence of each is
given by PX(I),[*] such that

$$\sum_{I=1}^{3} PX(I) = 1.$$

These values are calculated from MISDEF, the nominal number of terminal
interceptors at the target, as follows:

$$NTX(1) = MISDEF * RX(1)^{**}$$

$$NTX(2) = MISDEF$$

$$NTX(3) = MISDEF * RX(2)^{**}$$

---

[*] PX(1) is the user-input parameter PROBLOW. PX(3) is the user-input
     parameter PROBHIGH. PX(2) = 1 -PX(1) -PX(3).

[**] RX(1) is the user-input parameter LOWFAC. RX(2) is the user-input
     parameter HIGHFAC.

324

The probability that a warhead from the weapon G is killed by the terminal defense is then given by:

$$PKW(I) = PKTX * XDEG(G) \text{ if } NOBJ \quad NTX(I)$$

$$= \frac{NTX(I)}{NOBJ} * PKTX * XDEG(G) \text{ if } NOBJ \quad NTX(I), \quad \text{for } I=1,2,3$$

where NOBJ is the number of warheads plus decoys in the attack and the XDEG factor degrades PKTX for weapon group G. Hence, the probability that target component J survives NOWEP(G) weapons from group G is given by a calculation involving the use of the functions TABLEMUP and FMUP, which are described in other sections of this chapter.

The former function takes as input the modified single shot survival probability,

$$MSSSP(G,J,I) = PWK(I) + ((1 - PWK(I)) * XMUP(G,J))$$

and computes the kill factor,

$$KF(G,J,I) = TABLEMUP(MSSSP(G,J,I)).$$

The kill factors for all the weapons allocated to the target from each group are summed to generate the group total kill factor,

$$GTKF(G,J,I) = KF(G,J,I) * NOWEP(G) * NWHD(G).$$

(NWHD(G) is number of warheads per weapon from group G.) This factor is input to the function FMUP to generate the probability that target component J survives NOWEP(G) weapons from group G; i.e.,

$$S(J,G,I) = FMUP(GTKF(G,J,I)).$$

Hence, the total surviving target value is calculated from:

$$\text{Surviving Target Value} = \sum_{I=1}^{3} PX(I) \sum_{J=1}^{M} \sum_{N1=0}^{NN} VTOA(N1,J)$$

$$-VTOA(N1 + 1,J) \quad * \prod_{G=1}^{N1} S(G,J,I)$$

where

    VTOA(N1,J0   = value of component J when weapon N1 arrives

    NN   = total number of weapon groups

    VTOA(0,J)   = VO(J) = value of hardness component J

325

and

$$VTOA(NN + 1, J) = 0.$$

The innermost sum over N1, the weapon groups, must be carried out in order of the weapons' time of arrival; i.e., the first term corresponds to the N1 with shortest time of arrival, etc.

Hence the residual target calculation in RESVAL takes into account (1) uncertainties in the terminal interceptor stockpile, (2) target value dependence on time, (3) multiple hardness components of the target, (4) various penetration aids and decoy capabilities of attacking weapons, and (5) a detailed target-warhead interaction calculation.

This apparently complicated manner of calculating the target survival probability is required by the optional use of two damage laws. The functions TABLEMUP and FMUP determine which damage law is being used on the current target and modify their calculations accordingly. Since subroutine RESVAL is called a very large number of times for each missile-defended target, certain intermediate results are not saved in order to decrease execution time. In particular, the variables for the modified single shot survival probability, $MSSSP(G,J,I)$, the kill factor, $KF(G,J,I)$, and the total survival probability, $S(G,J,I)$, are never explicitly saved. The group total kill factor, $GTKF(G,J,I)$, is saved in a temporary storage variable, S. Thus, these four intermediate variables do not appear explicitly in the program.

Subroutine RESVAL is illustrated in figure 60.
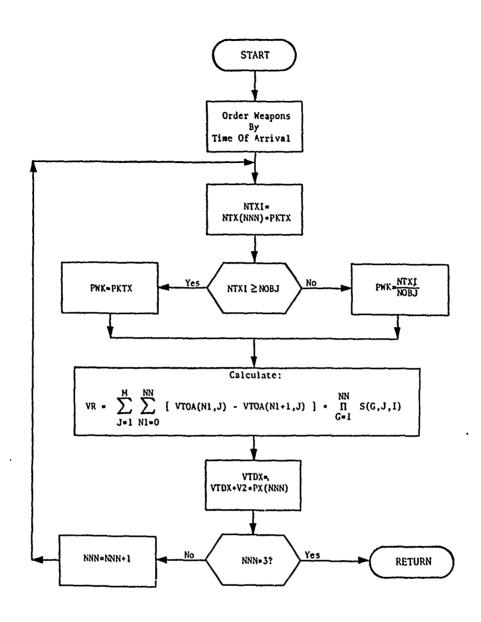
Figure 60. Subroutine RESVAL

327

SECTION 4. EVALALOC MODULE

4.1 Purpose

The purpose of module EVALALOC is to summarize the planned allocation of
weapons to targets and provide an expected value estimate of the results.
Provision is also included to evaluate the allocation for variations in
the values assigned selected parameters (planning factors) associated with
the weapons and targets. The evaluation can be made for either the whole
plan or for only targets in selected countries. EVALALOC may be run at
two stages of plan development, before module ALOCOUT or after module
PLANOUT. If run prior to the selection of desired ground zeros (DGZ)
for complex targets (accomplished in ALOCOUT), the analysis of aim point
offsets is not included. In this case, the results produced by EVALALOC
represent an upper limit estimate which assumes that each target element
in a complex is directly targeted. When EVALALOC is run after module
PLANOUT, the weapon aim points offsets are available and are included in
the expected value computations.

4.2 Input

EVALALOC may operate·at any stage after weapons have been assigned to
targets. EVALALOC interrogates the target list, the weapon gr·up chain
and stores attributes necessary for the evaluation, and obtain: strikes
from the assignment chain.

4.3 Output

EVALALOC does not update the data base for use by later processors; its
sole output is a set of printed summaries which present the expected
value results of the planned weapon allocation.

4.4 Concept of Operation

Subroutine ENTMOD reads user inputs and executes subroutine EVAL2 for
each requested plan evaluation. Once executed, EVAL2 controls all flow
for the given evaluation.

4.5 Identification of Subroutine Functions

4.5.1 Subroutine EVAL2. Subroutine EVAL2 processes the targets one at
a time. For each target (or target element of a complex target), the
assigned weapons are read from the data base and ordered by time of
arrival. Surviving target values are calculated (within subroutine
EVALPLAN), utilizing the same damage functions used in module ALOC
(subroutine WAD), except that correlations are ignored. After the
survival probability of each target is computed, the target weapons
are classified for summarization purposes.

329

4.5.2 <u>Subroutine TGTMODIF</u>. For each individual target, subroutine TGTMODIF is called by EVAL2 for determination of altering selected target parameters as user directed.

4.5.3 <u>Subroutine WPNMODIF</u>. Similar to subroutine TGTMODIF but modifies weapon attributes.

4.6 <u>Common Block Definition</u>

Common blocks used by EVALALOC are outlined in table 10. Common blocks that communicate with the COP are given in appendix A of Program Maintenance Manual, Volume I.

Table 10. Module EVALALOC Common Blocks
(Part 1 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
CLAUSES	ONPRINTS SETTING SORT COUNTRIES TGTMOD WPNMOD	Set to -1 for each evaluation; reset to starting location into INSGET's arrays if the corresponding adverb exists.
DAMAGE	NALLTYPE(7)	Number of weapons of category based on FUNCTION scheduled against targets
	ATTYPE(7)	Number of weapons of category based on FUNCTION delivered to targets
	SKDWPTYP(7)	Yield from weapons of category based on FUNCTION scheduled against targets
	DELWPTYP(7)	Yield from weapons of category based on FUNCTION delivered against targets
	DELYLD	Megatonnage actually arriving at the target
	PLANYLD	Megatonnage allocated to the target
	VALDES	Value of target destroyed
	VALESC	Value of target escaping
	SURV	Fraction of the target surviving
	VALFAC	Fraction of total complex value represented by the target
	NPASS	Pass number of current evaluation
/GROUPS/	NAMECLAS(250)	Class name of the group
	NAMETYPE(250)	Weapon type name of the group
	IWEAP(250)	Index based on attribute FUNCTI

Table 10. (Part 2 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
	GRPCEP(250)	CEP of weapons in group
	GRPSBL(250)	SBL of weapons in group
	WEPREL(250)	REL of weapons in group
	GRPKNV(250)	PKNAV of weapons in group
	GRPYLD(250)	YIELD of non-ASM weapons in group
	GRPREL(250)	REL of weapons in group
	ASMCEP(250)	CEP of ASMs in group
	ASMREL(250)	REL of ASMs in group
	ASMYLD(250)	YIELD of ASMs in group
	NGROUP	Number of groups
/OPERATOR/	COMMA	Index number for operator Comma
	LPAREN	Index number for operator left parenthesis
	RPAREN	Index number for operator right parenthesis
	EQUALS	Index number for operator equals
	DASH	Index number for operator dash
	SLASH	Index number for operator slash
	ALFOS	Index number for alphabetic follows and alphabetic input values
	FLOFO	The same as ALFOS but for floating point values
	NUMATRIB	The same as ALFOS but for a numeric attribute
	VALGRP	The same as ALFOS but for the attribute GROUP

Table 10. (Part 3 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
/STRIKES/	IG(30)	Group number of assigned weapons
	KORDOR(30)	Weapon penetration corridor
	RELVAL(30)	Relative value of weapon allocation
	PENPROB(30)	Weapon penetration probability
	TOA(30)	Weapon time of arrival on target
	ISAL(30)	Salvo number of weapon. If bombers equal 1 if ASM allocation
	IHOB(30)	Desired height of burst
	BLAT(30)	Offset latitude
	BLON(30)	Offset longitude
	ISORTN(30)	Sortie sequence number
	DE(30)	Survival probability of target after arrival of weapon I
	TIMEVAL(30)	Survival probability of time dependent target value after arrival of weapon I
	NSTRIKE	Number of weapon assignments
/TYPCLS/	VOTYPE	Total target value for the type
	VREMTYPE	Total remaining value for the type
	VDESTYPE	Total value destroyed for the type
	VESCTYPE	Total value escaped for this type
	SKEDTYPE	Megatonnage allocated to the type
	DELTYPE	Megatonnage delivered to the type
	VOCLAS	Value for the class
	VREMCLAS	Value remaining for the class

333

Table 10. (Part 4 of 4)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
	VDESCLAS	Value destroyed for this class
	VESCCLAS	Value escaped for the class
	SKEDCLAS	Megatonnage allocated to the class
	DELCLAS	Megatonnage delivered to the class

334
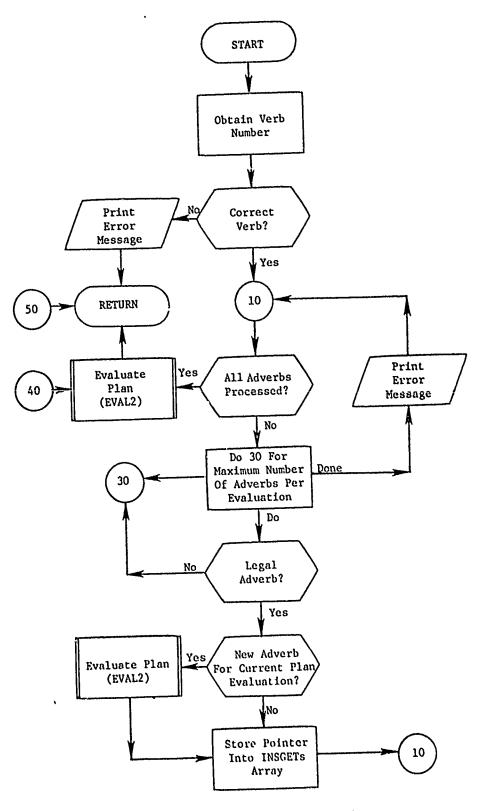
4.7 Subroutine ENTMOD

PURPOSE: To conduct the flow of execution as user directed.

ENTRY_POINTS: ENTMOD (first subroutine called when overlay link
                         EVAL is executed)

FORMAL PARAMETERS: None

COMMON BLOCKS: CLAUSE, OOPS, OPERATE

SUBROUTINES CALLED: EVAL2, INSGET

CALLED BY: COP

Method:

Subroutine ENTMOD controls the flow of each plan evaluation as user
directed. ENTMOD reads user requests and executes subroutine EVAL2
which in turn queries and evaluates all target weapon assignments. Any
number of target passes are possible within one EVALALOC execution. If
any adverb repeats itself, it is assumed the repeating adverb is for a
new allocation evaluation and, therefore, implies EVAL2 may be executed
for any previously read adverbs. Also, EVAL2 is called upon processing
all adverbs.

Subroutine ENTMOD is illustrated in figure 61.

Figure 61. EVALALOC Module

336

4.8 Subroutine EVALPLAN

PURPOSE: To classify the weapons allocated to each target and compute the corresponding change in target value after the attack.

ENTRY POINTS: EVALPLAN

FORMAL PARAMETERS: None

COMMON BLOCKS: CLAUSE, C30, DAMAGE, GROUPS, STRIKE

SUBROUTINES CALLED: ABORT, DIST, INITPR, INSGET, SSKPC, SSSPCA, VALTAR

CALLED BY: EVAL2

Method:

EVALPLAN is called once for each target to consider the damage done by all weapons allocated to the target. When called, internal arrays are initialized and the number of target value components is checked. If the target has more than five value components, an error message is printed and RETURN executed.
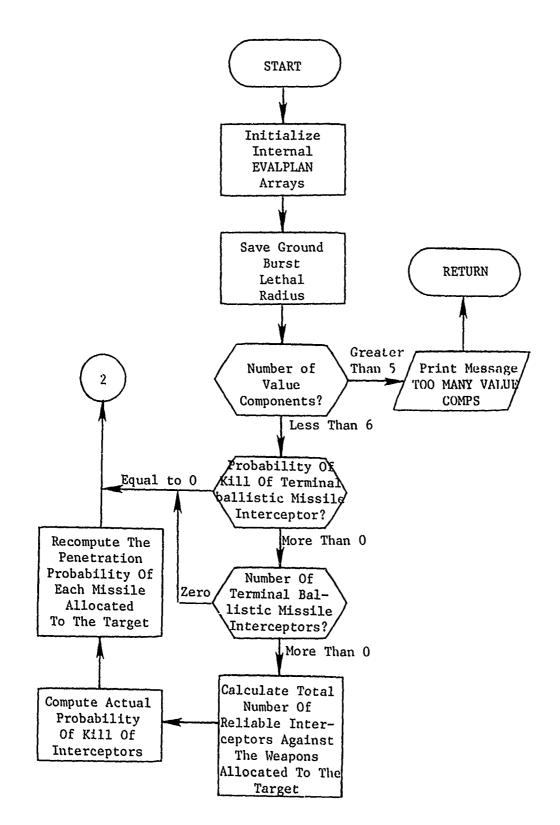
EVALPLAN determines if the target is defended by effective terminal ballistic missile interceptions. If so, EVALPLAN recomputes the penetration probability for each missile allocated to the target before performing target survival calculations.
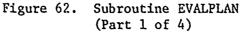
For each assignment height of burst, CEP, YIELD, and REL are retrieved from block /GROUP/ and stored locally.

If this is a first pass, EVALPLAN classifies each weapon into one of seven categories: alert LRA, nonalert LRA, TAC, SLBM (combined with SLCM), MRBM, IRBM or ICBM. The correct index for these categories has been placed within array IWEAP by subroutine EVAL2. It then updates arrays for summarizing prints.

For each weapon, EVALPLAN updates DELYLD and PLANYLD for all passes of EVALALOC and calculates the value FVALTOA of the target at the weapon time of arrival. Then it uses functions DIST and SSKPC and subroutine SSSPCALC to calculate kill and survival probabilities SSK and SSS, except for naval weapons where it uses PKNAV to calculate SSK and SSS. These probabilities are subsequently used to compute values for PRODSS, a hardness component probability factor; CUMDES, the value of each hardness component destroyed and CUMESC, the value of each hardness component escaping during the attack. The values of DE and TIMEVAL are then calculated using these variables. Finally EVALPLAN calculates the target survival probability (SURV), the total target value destroyed (VALDES), and the total target value escaping during the attack (VALESC).

Subroutine EVALPLAN is illustrated in figure 62.

337

START

Initialize
Internal
EVALPLAN
Arrays

Save Ground
Burst
Lethal
Radius

RETURN

Number of
Value
Components?

Greater
Than 5

Print Message
TOO MANY VALUE
COMPS

2

Less Than 6

Probability Of
Kill Of Terminal
ballistic Missile
Interceptor?

Equal to 0

More Than 0

Recompute The
Penetration
Probability Of
Each Missile
Allocated
To The Target

Zero

Number Of
Terminal Bal-
listic Missile
Interceptors?

More Than 0

Compute Actual
Probability
Of Kill Of
Interceptors

Calculate Total
Number Of
Reliable Inter-
ceptors Against
The Weapons
Allocated To The
Target

Figure 62.   Subroutine EVALPLAN
(Part 1 of 4)

338

```
                              ( 2 )
                               │
                               ▼
                    ┌─────────────────────┐
                    ║ Call INITPROB To    ║
                    ║    Initialize       ║
                    ║    Arrays In        ║
                    ║    SSSPCALC         ║
                    └─────────────────────┘
                               │
                               ▼
┌──────────────────┐   Done  ┌─────────────────────┐
│ Restore Ground   │◄────────│ Do 90 For Each      │◄──── ( 90 )
│ Burst Lethal     │         │    Weapon           │
│ Radius           │         │ Allocated To        │
└──────────────────┘         │    The Target       │
          │                  └─────────────────────┘
          │                            │ Do
          ▼                            ▼
┌──────────────────┐         ┌─────────────────────┐
│ Calculate Target │         │   Set Lethal        │
│    Survival      │         │    Radius           │
│  Probability     │         │  According To       │
│    Value Of      │         │     HOB             │
│    Target        │         └─────────────────────┘
│ Destroyed, And   │                   │
│ Value Of Target  │                   ▼
│ Escaping During  │         ┌─────────────────────┐
│  The Attack      │         │ Set NWP Equal       │
└──────────────────┘         │    To The           │
          │                  │  Multiplicity       │
          ▼                  │ Of The Weapon       │
     ╭──────────╮            │ (-KORDOR(I) For     │
     │  RETURN  │            │   Missiles)         │
     ╰──────────╯            └─────────────────────┘
                                       │
                                       ▼
                             ┌─────────────────────┐
                             │ Calculate RELT-     │
                             │ Relative Proba-     │
                             │ bility Of Kill      │
                             │  For Weapon         │
                             └─────────────────────┘
                                       │
                                       ▼
                                     ( A )
```

Figure 62.   (Part 2 of 4)

```
                    ┌───┐
                    │ A │
                    └───┘
                      │
                      ▼
        ┌──────────────────────────┐
        │  Save Yield and Delivery │
        │      Error Data For      │
        │          Weapon          │
        └──────────────────────────┘
                      │
                      ▼
                  ╱─────────╲                ┌──────────────┐
                 ╱  Is Weapon ╲      Yes      │  Reset Kill  │
                ╱     An        ╲────────────▶│  Probability │
                ╲     ASM?      ╱             │  Using ASM   │
                 ╲             ╱              │ Kill Factors │
                  ╲─────────╱                └──────────────┘
                      │                              │
                      No ◀───────────────────────────┘
       2002           │
                      ▼
                  ╱─────────╲
                 ╱   First    ╲      No         ┌─────┐
                ╱    Pass       ╲──────────────▶│ 109 │
                ╲      ?        ╱               └─────┘
                 ╲             ╱
                  ╲─────────╱
                      │
                      Yes
       86             │
                      ▼
        ┌──────────────────────────┐
        │     Classify Weapon      │
        │     By Function -        │
        │  Alert LRA, Nonalert     │
        │   LRA, TAC, SLBM,        │
        │  ICBM, MRBM, Or IRBM     │
        └──────────────────────────┘
                      │
                      ▼
                  ┌─────┐
                  │ 111 │
                  └─────┘
```

Figure 62.   (Part 3 of 4)

```
                    ( 109 )
                       │
                       ▼
          ┌────────────────────────┐
          │   Calculate Value Of   │
          │     Target At Time     │
          │  Of Arrival of Weapon  │
          │     Over  All  Time    │
          │       Components       │
          │       (FVALTOA)        │
          └────────────────────────┘
                       │
                       ▼
     ┌──────────────────┐   Done   ┌──────────────────┐
 ┌──▶│  Do 95 For Each  │─────────▶│   Restore Yield  │
 │   │     Hardness     │          │    And Delivery  │
 │   │   Component Of   │          │     Error Data   │
 │   │    The Target    │          │     For Weapon   │
 │   └──────────────────┘          └──────────────────┘
 │             │ Do                          │
 │             ▼                             ▼
 │  ┌────────────────────────────┐  ┌──────────────────────────┐
 │  │  DIST,  SSKPC,  SSSPCALC    │  │     Calculate Survival   │
 │  │ Calculate Survival Probability│ │   Probability Of Target, │
 └──│ Factor, PRODSS, Value of Hard-│ │ DE, And Survival Probability│
    │  ness Component Destroyed   │  │   Of Time Dependent Target│
    │  CUMDES,  Value  Of Hardness │ │    Value  After  Arrival │
    │  Component Escaping During  │  │      Of  The  Weapon     │
    │    The Attack, CUMESC       │  └──────────────────────────┘
    └────────────────────────────┘              │
                                                 ▼
                                             ( 90 )
```

Figure 62.   (Part 4 of 4)

4.9 Subroutine EVAL2

PURPOSE: To read target weapon allocation data, evaluate and
                     classify it, summarize it and store calculations
                     for summary prints.

ENTRY POINTS: EVAL2

FORMAL PARAMETERS: None

COMMON BLOCKS: CLAUSE, C10, C15, C30, DAMAGE, GROUPS, OOPS,
                     OPERATE, STRIKE

SUBROUTINES CALLED: DIRECT, EVALPLAN, HEAD, HDFND, INSGET, ITLE,
                     NEXTTT, ORDER, PREVAL, PTIME, REORDER, RETRV,
                     SORTIT, TGTMODIF, WPNMODIF

CALLED BY: ENTMOD (of EVALALOC)

Method:

EVAL2 begins by processing input clauses as introduced by adverbs
ONPRINTS, SETTING, COUNTRIES, or TGTMOD. For the print clause, the upper-
bound target number (ITGTMAX) to be printed is stored. Parameters PKTX
and LAW are set according to the SETTING clause. If COUNTRIES exists,
the method of comparison for target evaluation consideration resides
within parameter DESIRE. The beginning (ICL) and end (LCL) pointers
into INSGETs for the COUNTRIES clause are defined for further inter-
rogation. Finally, the TGTMOD clause is queried for syntax errors.

By chaining the weapon groups, attributes necessary for target-weapon
assignment evaluation are stored for use within subroutine EVALPLAN.
Weapon categories are indexed (array IWEAP) based on attribute FUNCTI
which serves for collecting items necessary for summary prints. Also
ASM data is stored, if applicable.

Now the individual target list (TARNUM) will be queried and tested
against the COUNTRIES clause for tests of inclusion within the current
evaluation. If tests prove satisfactory, target data is placed on file
unit 21. Only those items necessary for print or allocation evaluation
are written.

After chaining the target list, subroutine SORTIT, orders the targets
based on region, country location and DESIG consideration if adverb
SORT exists. By collecting target elements in this fashion, EVAL2 is
easily amendable to any new future sort requests. In addition, core
utilization remains minimal even with the restriction of open-ended
targets.

Individual targets record are now read from data unit SORTED and the wea-
pon assignment chain (ASGWPN) queried for each target record. Assignment

342

parameters are collected and eventually reordered by time of arrival which
is necessary for evaluation. If directed, subroutine TGTMODIF makes mod-
ifications to the targets as user directed. Finally EVALPLAN evaluates
the target's allocation. Individual target data items are printed, if
requested.

At this stage all evaluation and prints concerning the current target
have been completed. All that remains consists of storage of calculated
items necessary for summary prints. Rather than temporarily storing items,
results are written onto data file unit 22. Five separate records are
written per target. The first word of each record defines the summary
report number. The second and third words always equal attributes CLASS
and TYPE. Elements beyond the third words depend upon the summary re-
port criterion.

After all targets are processed, data unit 22 is sorted based on the
first three words which is a sort perfect for print purposes. Now sub-
routine PREVAL reads the sorted unit (LUNTAB) and generates reports.
Upon returning, control is passed to ENTMOD for consideration of remain-
ing passes.

Subroutine EVAL2 is illustrated in figure 63.
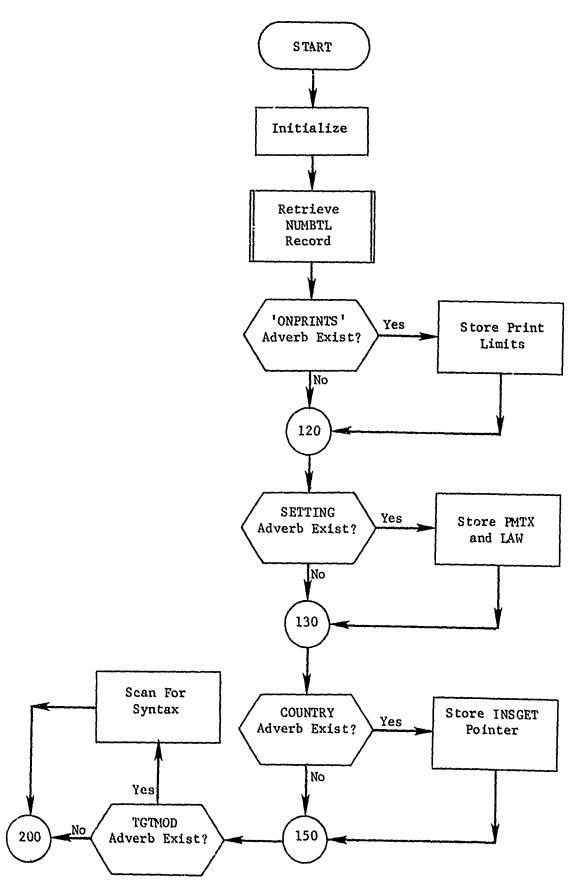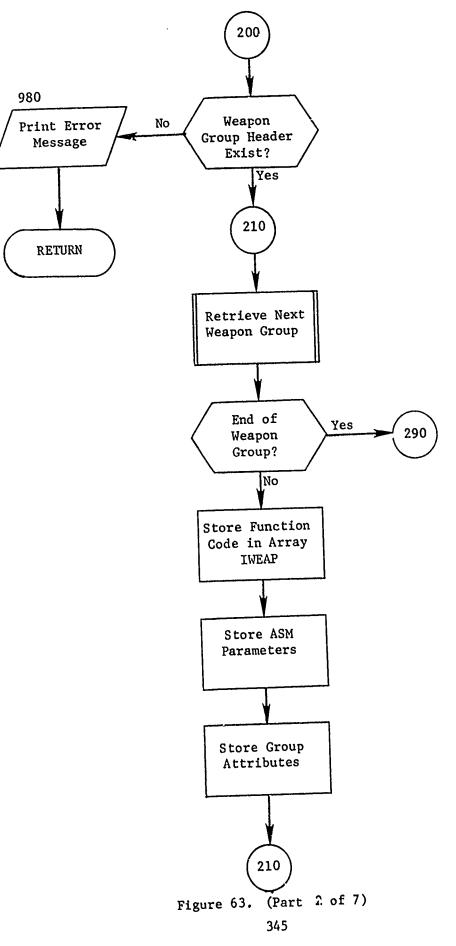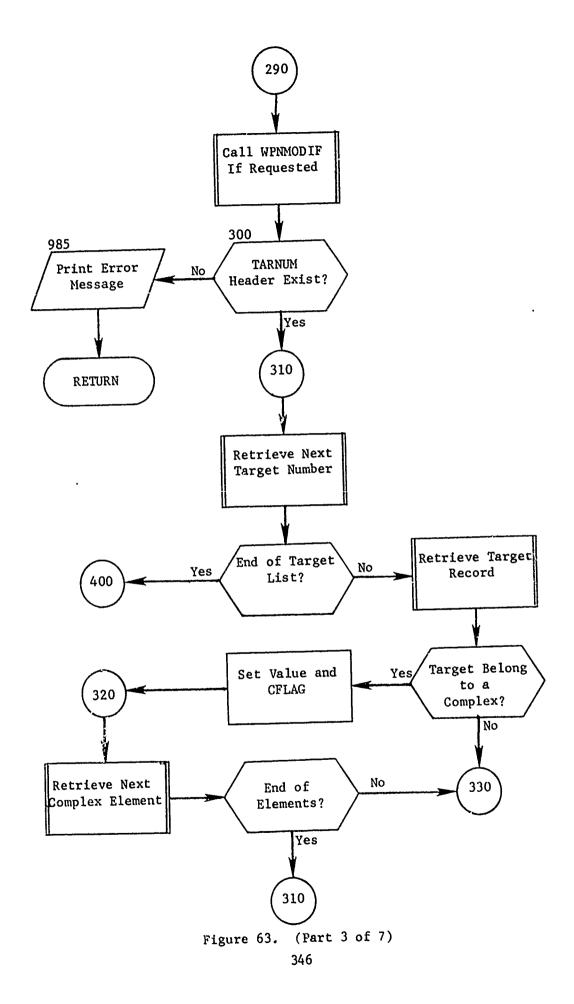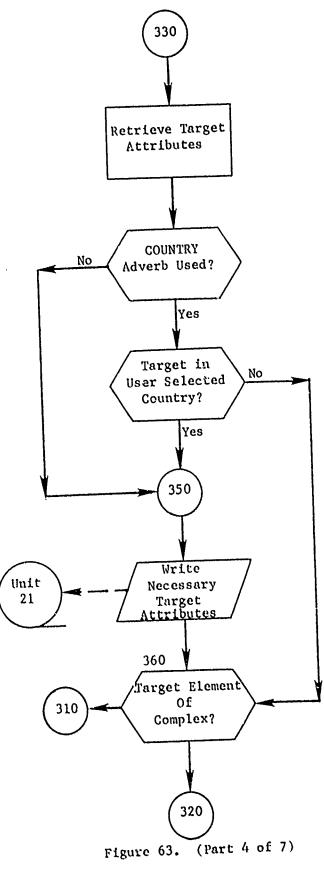
Figure 63.   Subroutine EVAL2 (Part 1 of 7)

Figure 63. (Part 2 of 7)

345

Figure 63. (Part 3 of 7)

346

Figure 63. (Part 4 of 7)

Figure 63. (Part 5 of 7)

Figure 63. (Part 6 of 7)

Figure 63. (Part 7 of 7)
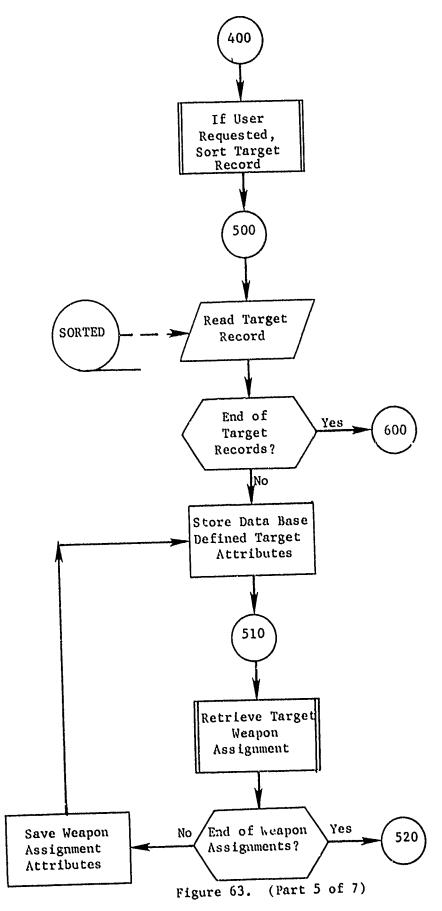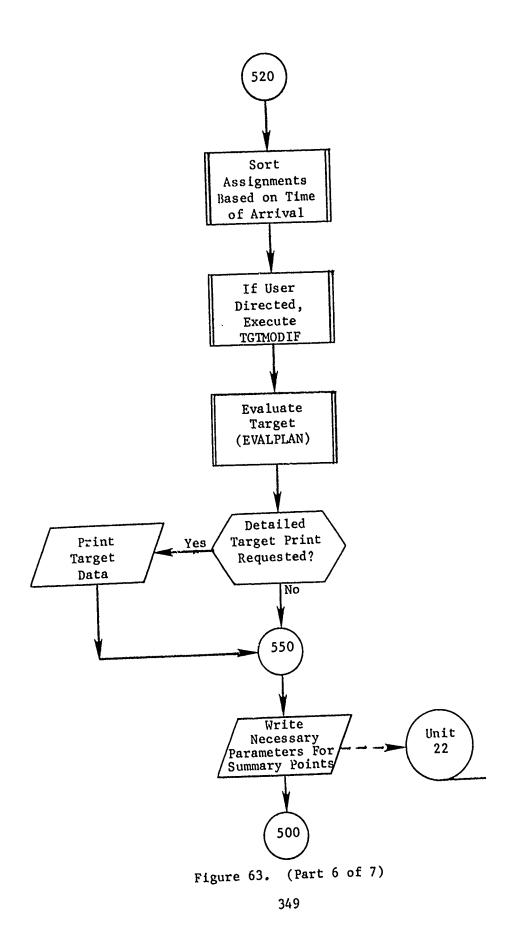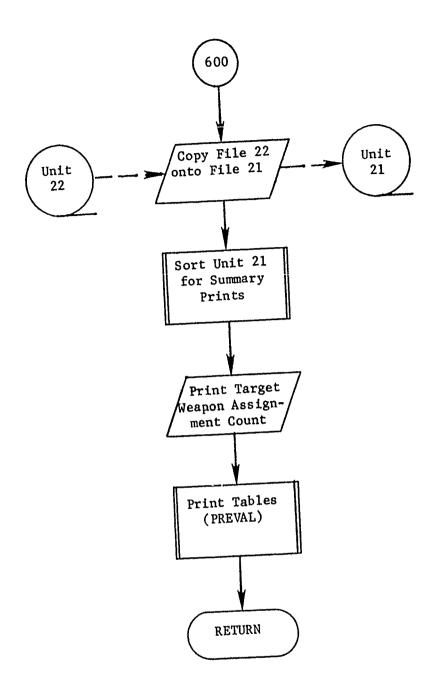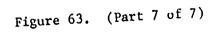
4.10 Subroutine PREVAL

PURPOSE: Print summary tables concerning the allocation.

ENTRY POINTS: PREVAL

FORMAL PARAMETERS: LUNTAB – File unit number where print related data
                                 resides
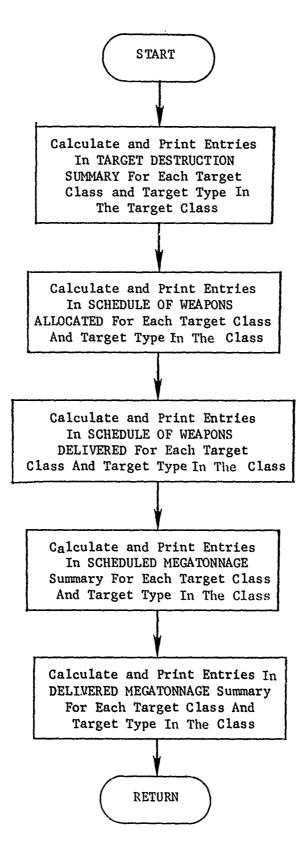
COMMON BLOCKS: DAMAGE, TYPCLS

SUBROUTINES CALLED: None

Method:

Subroutine PREVAL's sole function consists of reading data unit (LUNTAB)
as prepared by EVAL2 and generating reports as outlined within figure 64.
Records on unit LUNTAB are sorted based on report code and within that
sort arranged according to class and type. The second and third level
sort orders ease the burden in producing counts for similar type and
class intersections. PREVAL, then, simply reads and prints.

Subroutine PREVAL is illustrated in figure 64.

```
                        ┌─────────────┐
                        │    START    │
                        └─────────────┘
                               │
                               ▼
              ┌────────────────────────────────────┐
              │      Calculate and Print Entries    │
              │        In TARGET DESTRUCTION        │
              │       SUMMARY For Each Target       │
              │        Class and Target Type In     │
              │            The Target Class         │
              └────────────────────────────────────┘
                               │
                               ▼
              ┌────────────────────────────────────┐
              │      Calculate and Print Entries    │
              │         In SCHEDULE OF WEAPONS      │
              │     ALLOCATED For Each Target Class │
              │       And Target Type In The  Class │
              └────────────────────────────────────┘
                               │
                               ▼
            ┌──────────────────────────────────────────┐
            │        Calculate and Print Entries       │
            │           In SCHEDULE OF WEAPONS         │
            │            DELIVERED For Each Target      │
            │     Class And Target Type In The  Class  │
            └──────────────────────────────────────────┘
                               │
                               ▼
              ┌────────────────────────────────────┐
              │      Calculate and Print Entries    │
              │        In SCHEDULED MEGATONNAGE     │
              │      Summary For Each Target Class  │
              │        And Target Type In The Class │
              └────────────────────────────────────┘
                               │
                               ▼
              ┌────────────────────────────────────┐
              │     Calculate and Print Entries In  │
              │     DELIVERED MEGATONNAGE Summary   │
              │         For Each Target Class And   │
              │          Target Type In The Class   │
              └────────────────────────────────────┘
                               │
                               ▼
                        ┌─────────────┐
                        │   RETURN    │
                        └─────────────┘
```

Figure 64.   Subroutine PREVAL

4.11 Subroutine SSSPCALC

PURPOSE: To calculate target survival probabilities for
                        multiple-weapon attacks. This routine will con-
                        sider either the exponential or square-root
                        damage law.

ENTRY POINTS: INITPROB, SSSPCALC

FORMAL PARAMETERS: SSS - A single-shot survival probability
                        NWP - A number of weapons
                          J - Index to hardness component

COMMON BLOCKS: LAW, LITTLE

SUBROUTINES CALLED: None

CALLED BY: EVALPLAN

Method:

The INITPROB entry point is used to initialize two local arrays which are
used in the calculations. This entry is called once for each target
before processing the weapon damage calculations in EVALPLAN. The formal
parameters have no effect on this entry point. The two local arrays are
indexed by hardness component. They are defined as follows:
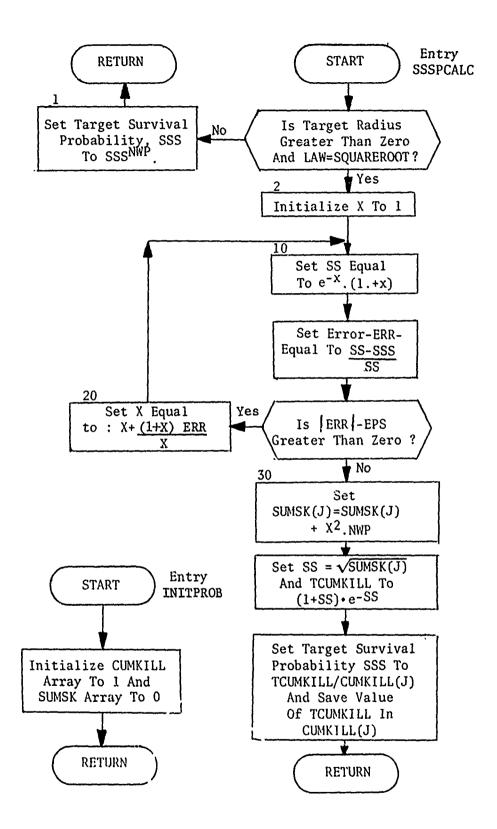
> CUMKILL(K)   Current fraction of Kth hardness component
>   surviving. Initialized to 1.0.
>
> SUMSK(K)   Current sum of kill factors for Kth hardness
>   component. Initialized to 0.0.

Entry SSSPCALC computes the multiple-weapon survival probability from the
single-shot survival probability. If the exponential damage option has
been selected, then the multiple-weapon survival probability is equal to
the product of all the single-shot survival probabilities for each weapon.

If the square-root damage law option has been selected, the routine checks
to see if the target radius is greater than zero. If not, the exponential
damage function is used. If so, the routine must calculate the square-
root kill factor corresponding to the input single-shot survival probabil-
ity. The algorithm used for this is the same one that is used in subrou-
tine SETABLE in program ALOC. The algorithm is a recursive, one-dimensional
search procedure to find the appropriate kill factor. The new kill factors
determine a new sum. This new sum defines the new fraction of the target
that survives. The multiple-weapon survival probability is then the ratio
of the new fraction surviving to the old fraction surviving.

Subroutine SSSPCALC is illustrated in figure 65.

353

Figure 65. Subroutine SSSPCALC

354

4.12 Subroutine TGTMODIF

PURPOSE: To enable the user to modify five target parameters:

FVULN1 -- The hardness of the target component

VOZ -- The value of the target at hardness FVULN1

T(1-5) -- Time components when value changes

FVALT -- Fraction of value in first time component

PEN -- Penetration probability of a weapon allocated

ENTRY POINTS: TGTMODIF

COMMON BLOCKS: CLAUSE, C30, GROUPS, OPERATE, STRIKE

SUBROUTINES CALLED: INSGET, ITLE

CALLED BY: EVAL2

Method:

Modifications of target parameters are made in accordance with the TGTMOD clause as user directed. The target type may be ALLTGTS or a specific class or type name such as MILITARY, BEAR, etc.

The penetration probability PEN is weapon-target dependent. If it is to be modified, weapon type names to which the modification applies follow the operator right parenthesis within the TGTMOD clause.

In all cases TGTMODIF modifies the specified target parameter for the specified target type by multiplying it by factor XTGTATT which is also user defined.

Subroutine TGTMODIF is illustrated in figure 66.

355
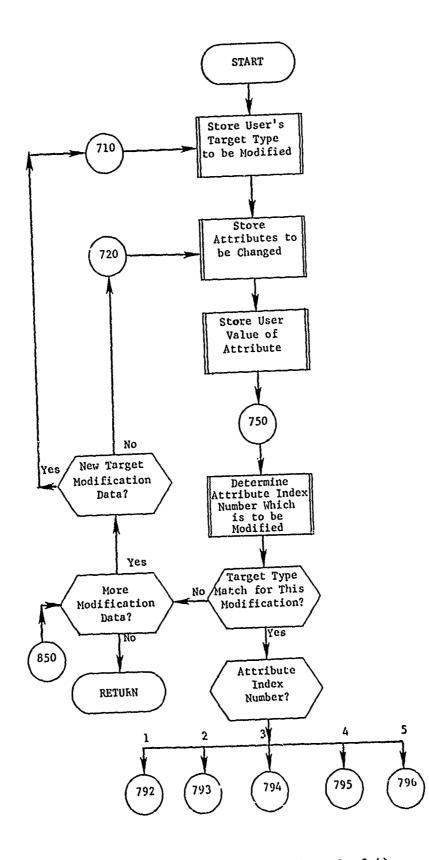
Figure 66.   Subroutine TGTMODIF (Part 1 of 4)
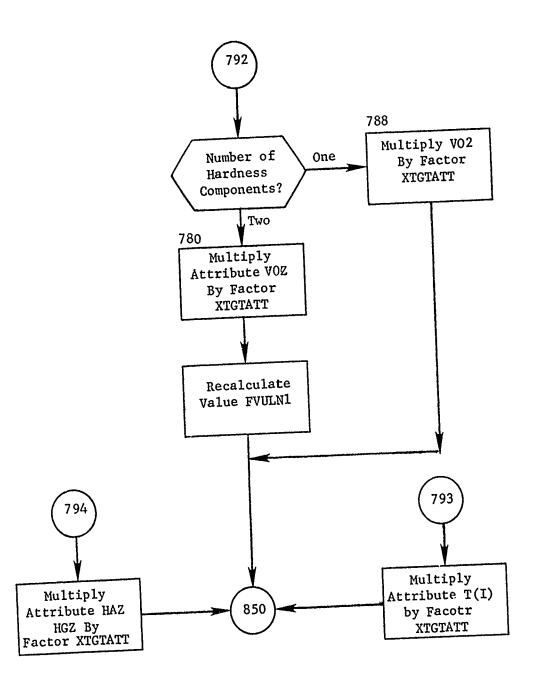
Figure 66. (Part 2 of 4)
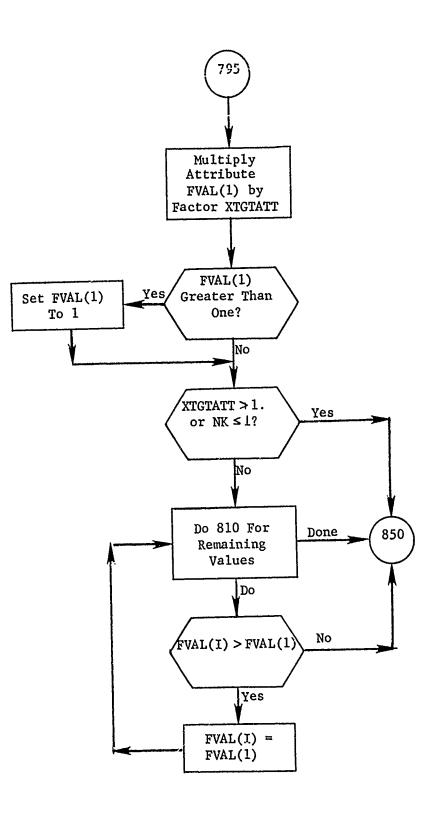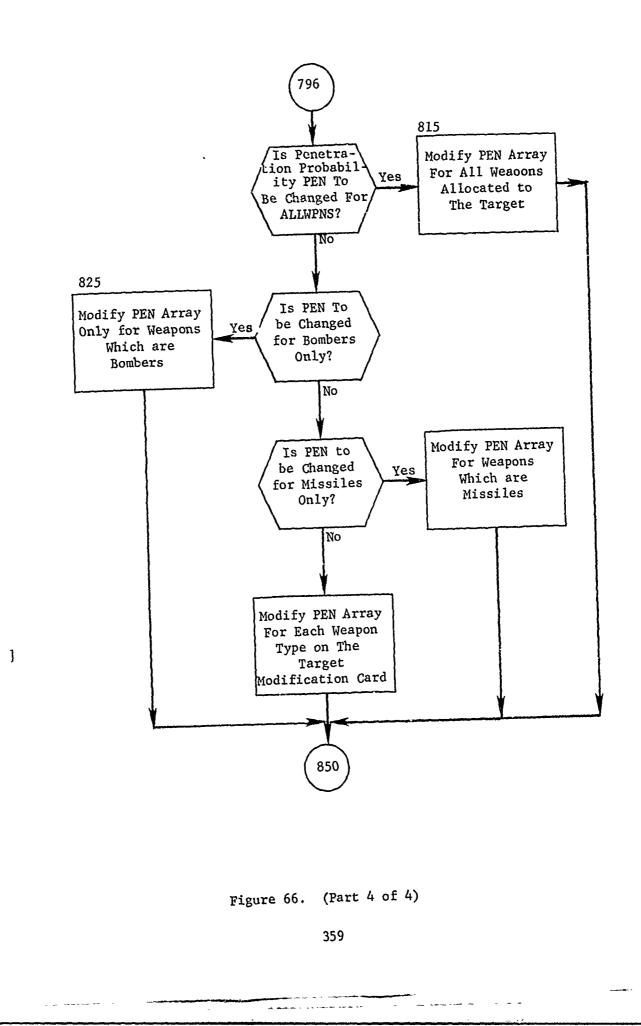
Figure 66. (Part 3 of 4)

358

Figure 66. (Part 4 of 4)

4.13 Subroutine WPNMODIF

PURPOSE: To allow the user to modify reliability REL, cir-
                   cular error probability CEP, weapon YIELD, or DBL
                   probability.

ENTRY POINTS: WPNMODIF

COMMON BLOCKS: CLAUSE, GROUPS, OOPS, OPERATE

SUBROUTINES CALLED: INSGET, ITLE

CALLED BY: EVAL2

Method:

WPNMODIF allows the user to modify specified weapon parameters for chosen
weapon types or for a selected weapon group. The weapon type name may be
ALLWPNS, BOMBERS, MISSILES, a specific type name such as B-52 or a group
number. Local parameter XWPNATT contains the multiplier which multiplies
the weapon parameter.

Subroutine WPNMODIF is illustrated in figure 67.
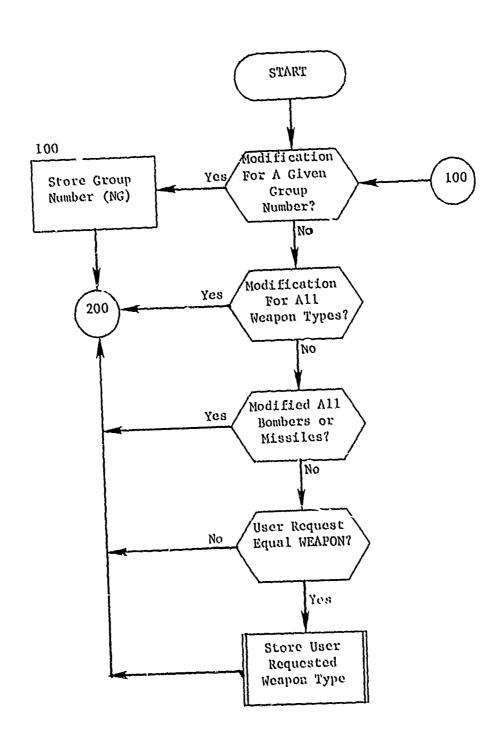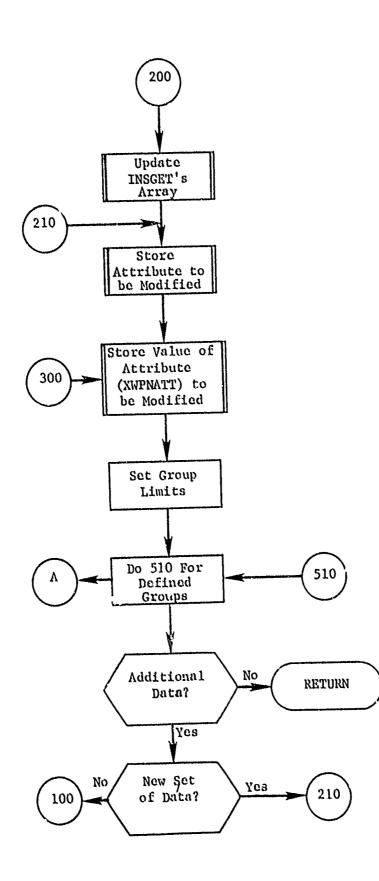
Figure 67. Subroutine WPNMODIF (Part 1 of 3)

361
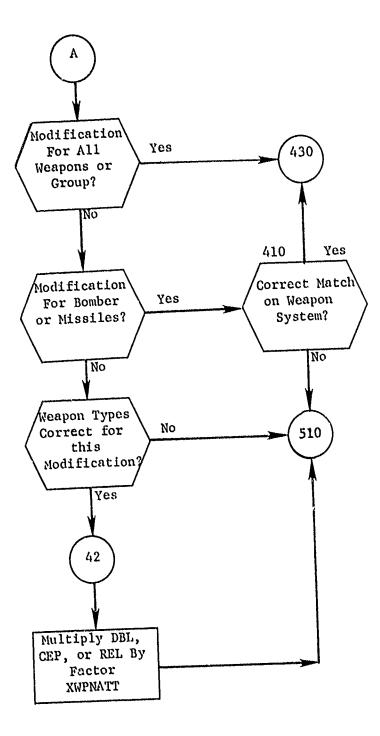
Figure 67. (Part 2 of 3)

362

Figure 67. (Part 3 of 3)

SECTION 5. MODULE ALOCOUT

5.1 Purpose

Module ALOCOUT is responsible for selecting optimum DGZs (desired ground zeros), also called weapon aim points for weapon allocated to target complexes. ALOCOUT, also resorts weapon assignments at the group level for use within the Sortie Generation subsystem.

Module ALOC specifies weapon groups assigned to targets together with associated targeting data. ALOCOUT extracts data from these records and computes any aiming offsets required by the plan. For simple or multiple targets, no calculations are performed. In the case of complex targets which can have several elements at slightly different coordinates, ALOCOUT employs subroutine DGZ (desired ground zero selector) to select optimum aim points within the target complex.

5.2 Input

ALOCOUT operates after module ALOC assigned weapons to targets. These records (ASSIGN) in addition to the supporting data base structure must be defined for proper execution.

5.3 Output

No new data base records are created during the execution of ALOCOUT. However, the weapon assignment records (called ASSIGN) are modified in two ways. First, for assignments to complex targets or for assignments to cities with nonzero radius, offsets as determined within the module are included within the ASSIGN record. Second, the assignment records at the weapon group level are resorted for use within the Sortie Generation subsystem. For missile groups, the sort is based on decreasing values of attribute RVAL. For bomber groups, the order is based on penetration corridor index and within the corridor sorted based on attribute RVAL. The penetration corridor that contains the largest number of strikes appears first within the sort, followed by the penetration corridor of the next largest number of strikes and so on.

5.4 Concept of Operation

ALOCOUT (that is, subroutine ENTMOD) operates with two overlay links. The first overlay reads the target list (TARNUM) passes controls to subroutine PROCCOMP for offset calculations when applicable and finally supplies optional prints. After all targets have been processed controls passes to the second overlay which consists entirely of subroutine SUMPRN which reorders strikes at the weapon group level and, if requested, produces prints concerning the individual assignments.

5.5 Identification of Subroutine Functions

5.5.1 Subroutine PROCCOMP. This subroutine controls the bulk of pro-
cessing for offset determination. It is executed by subroutine ENTMOD
only for those individual targets that require offset calculations.
After offsets have been determined the assignment record (ASSIGN) is
updated to include the values. Then, PROCCOMP returns to subroutine
ENTMOD for acquisition of the next target and the associated strikes.

5.5.2 Subroutine SUMPRN. This subroutine constitutes the entire second
overlay of ALOCOUT. Its purpose consists of resorting the weapon strikes
at the group level and providing optional prints.

5.6 Common Block Definition

Common blocks used by EVALALOC are outlined in table 11. Common blocks
that communicate with the COP are given in appendix A of Program Main-
tenance Manual, Volume I.

Table 11. ALOCOUT Common Blocks
(Part 1 of 2)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
CITY	ICITY	Set to nonzero for targets with attribute RADIUS not equal to zero
C1	XO(J), YO(J)	Coordinates of target element J
	VI(J)	Initial target element values
	RADL(J)	Lethal radius of target element J
	VTOA(J,I)	Value of target element J immediately following arrival of weapon I
	S(J,I)	Survival probability of target element J relative to weapon I
	VEFF(J,I)	Effective value of target element J relative to weapon I
	X(I),Y(I)	Offset coordinates of weapon I
	PDEL(I)	Probability of delivery of weapon I
	ERDEL(I)	Error in delivery of weapon I
	YDSCL(I)	Scaled yield for weapon I
	VESC(I)	Intermediate computational value used in subroutine VAL for determination of total escaping target value
	NI	Number of weapons for complex
	NJ	Number of target elements for complex
IONPRT	IPINDAT	User supplied print frequency for print option 1
	PRINCE(9)	Set TRUE if user requested option
ISKIPDGZ	ISKIPDGZ	Use indicator for DGZ. Normally it is 0. Compress resets it to 1 if more than 20 calls to it are made to reduce the number of target elements for a complex target; DGZ is not used again for the target in this case

367

Table 11. (Part 2 of 2)

BLOCK	VARIABLE OR ARRAY	DESCRIPTION
JAZ	F(400,26)	Holding arrays for sort purposes
LOCFIN	LOCFIN	Starting location into IRSET's arrays for adverb FINDMIN instructions
STRIKE	TOA(I)	Weapon time of arrival to target
	IREFSTRK(I)	Reference code of weapon strike
	N	Number of strikes
WPGT	YDMIN	Minimum allowable weapon group yield
	IGRP	Group number

5.7 Subroutine ENTMOD

PURPOSE: Read user inputs, collect target weapon assignments, and execute subroutine PROCCOMP for DGZ determination.

ENTRY POINTS: ENTMOD (first subroutine executed when overlay ALOCOUT is called)

FORMAL PARAMETERS: None

COMMON BLOCKS: CITY, C10, C15, C30, IONPRT, LOCFIN, STRIKE, WPGT

SUBROUTINES CALLED: DGZ, DIRECT, HDFND, HEAD, INSGET, NEXTTT, PROCCOMP, RETRV, SUMPRN, TIMEME, WEPGET

CALLED BY: COP

Method:
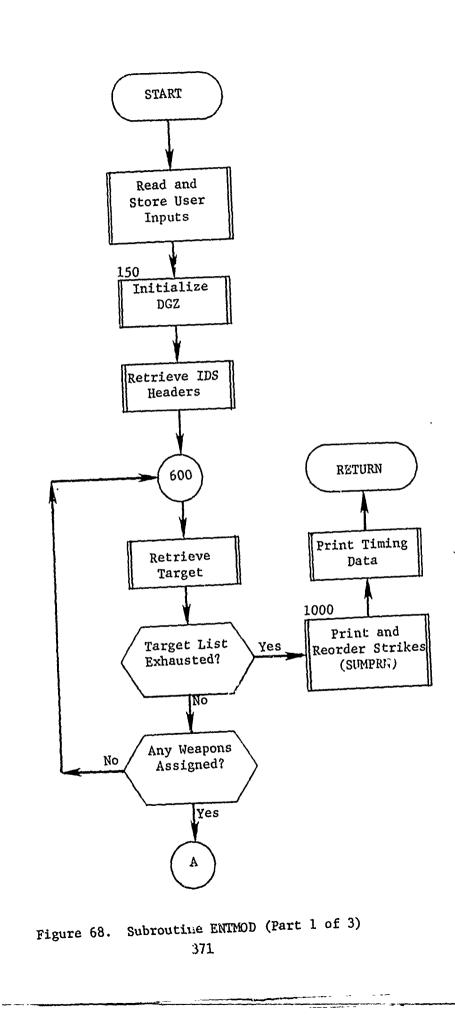
Subroutine ENTMOD reads and stores users input, walks the individual target chain (TARNUM), collects weapon assignments for the current target, and calls subroutine PROCCOMP for DGZ determination if the target represents a complex or is a city (attribute RADIUS greater than zero). After processing all targets, subroutine SUMPRNT (second overlay) reorders weapon assignments on a weapon group basis for use within the Sortie Generation subsystem.

Module ALOCOUT recognizes user supplied adverbs FINDMIN and ONPRINTS. FINDMIN sets the number of iterations subroutine FINDMIN uses for offset determination. ONPRINTS sets user options; results maintained in array PRINCE.

ALOCOUT now walks the identical target chain (TARNUM) which module ALOC made weapon assignments to. For each target, weapon assignments are stored on chain ASGWPN. If no strike exists processing continues by retrieving the next target on the list. Otherwise, for each weapon assignment, subroutine WEPGET retrieves weapon related attributes and updates necessary counts. Checks determine the nature of the target. Offsets are calculated only if the target represents a complex or is a city and has a nonzero RADIUS.

A complex target (or target complex) is a combination of target elements sufficiently close in geographic location that a weapon on any one of them will have some probability of killing other elements in the complex. Such target complexes are targeted as a unit by the allocator which allocates weapons against their total value, using one set of coordinates. In order to maximize targeting efficiency against such a complex, optimum aim points among the target elements must be selected. These aiming offsets are specified relative to the first target element only and are passed on in that form to subsequent modules.

When ENTMOD encounters a complex target, subroutine PROCCOMP is called.
PROCCOMP is responsible for assembling the data on a complex target in
a form that can be used efficiently for DGZ selection. Each target com-
ponent of the complex generates a standardized target element in the
arrays used by DGZSEL. (Targets with more than one hardness component
generate more than one such target element, and targets with a specified
target radius will generate several elements spread over the area of the
target to represent a value spread over the area.)

Subroutine ENTMOD is illustrated in figure 68.

Figure 68. Subroutine ENTMOD (Part 1 of 3)

371
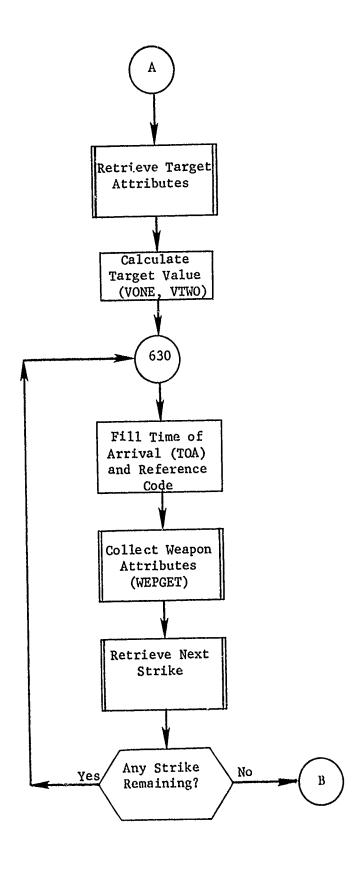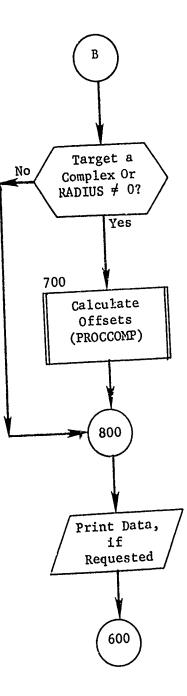
Figure 68. (Part 2 of 3)

Figure 68. (Part 3 of 3)

373
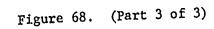
5.7.1 Subroutine COMPRESS

PURPOSE: For computational efficiency and/or to avoid
            exceeding maximum program dimensions, COMPRESS
            recombines those target elements which are
            near one another and have approximately the
            same lethal radius.

ENTRY POINTS: COMPRESS

FORMAL PARAMETERS: OPENTOL (type INTEGER). If OPENTOL is 0,
            distance and lethal radius tolerances will not
            be eased to decrease the number of target
            elements. If OPENTOL is 1, the tolerances
            will be eased.

COMMON BLOCKS: C1, C30, ISKIPD

SUBROUTINES CALLED: IMIN
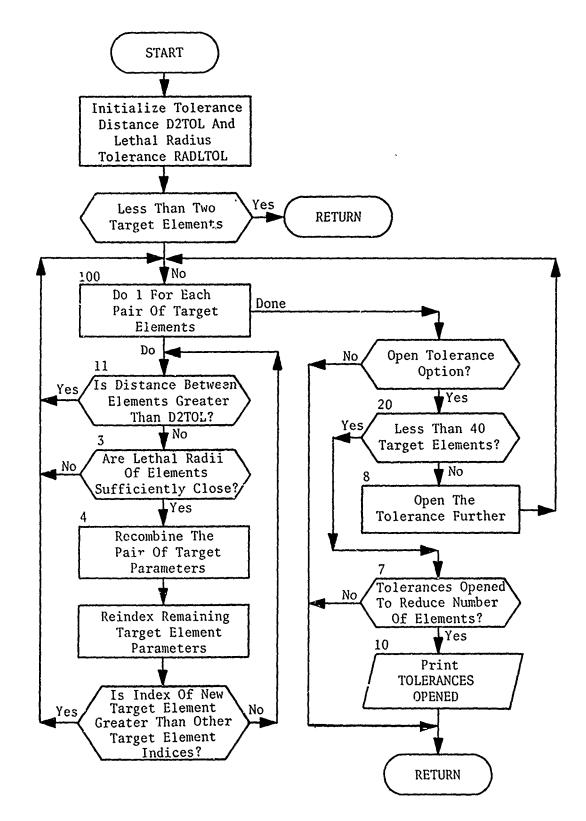
CALLED BY: PROCCOMP

Method:

When OPENTOL is zero, COMPRESS merely recombines target elements which
are close enough together that their lethal radii nearly coincide.
COMPRESS in this mode is called by PROCCOMP just prior to calling
DGZSEL in order to improve the efficiency of DGZSEL.

In the event that maximum program dimensions are reached, OPENTOL is
set to 1 by PROCCOMP; COMPRESS will then loosen its tolerances, if
necessary, to assure enough recombination of target elements to
eliminate the problem, at least temporarily. A print is also issued
in this case which gives the number of times the tolerances were
doubled.

The flowchart for COMPRESS is shown in figure 69.

Figure 69. Subroutine COMPRESS

375

5.7.2 Function CUMINV

PURPOSE:	To determine the value X such that Z is the probability that $x \leq X$.
ENTRY POINTS:	CUMINV
FORMAL PARAMETERS:	Z - The probability that $x \leq X$
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	PROCCOMP

Method:

Function CUMINV is illustrated in figure 70. By definition,

$$Z = P[x \leq X] = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{X} e^{-\frac{t^2}{2}} \cdot dt \text{ for } 0 < Z < 1$$

CUMINV uses the following approximation X^1 for X:
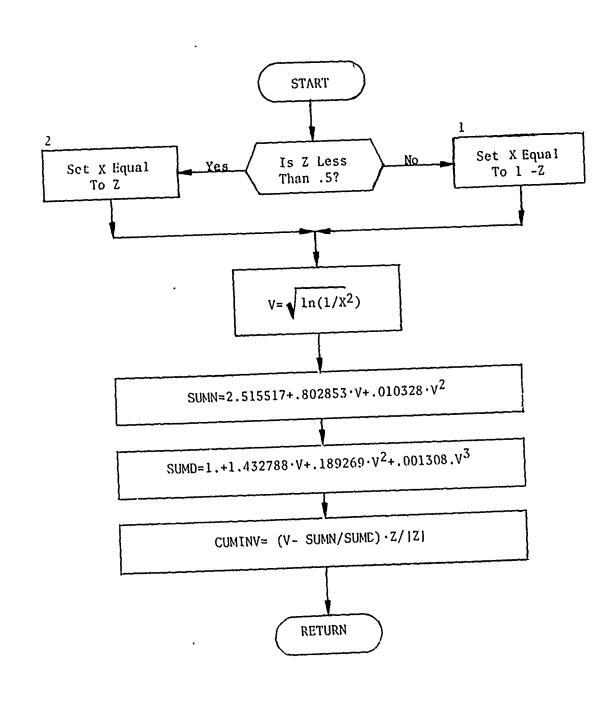
$$X^1 = \pm \left[V - \left(\frac{A_1 + A_2 \cdot V + A_3 \cdot V^2}{1 + B_1 \cdot V + B_2 \cdot V^2 + B_3 \cdot V^3} \right) \right]$$
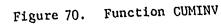
where

$$V = \sqrt{\ln (1/Q^2)}, \quad Q = Z \quad \text{or} \quad 1 - Z \quad \text{such that } 0 \leq Q \leq .5$$

and

$A_1 = 2.515517$	$B_1 = 1.432788$
$A_2 = .802853$	$B_2 = .189269$
$A_3 = .010328$	$B_3 = .001308$

Figure 70. Function CUMINV

377

5.7.3 Subroutine DGZ

PURPOSE:	DGZ is the controlling subroutine for the optional selection of DGZs for weapons allocated to complex targets.
ENTRY POINTS:	DGZ
FORMAL PARAMETERS:	None
COMMON BLOCKS:	C1, IONPRT, LOCFIN
SUBROUTINES CALLED:	FINDMIN, INSGET, PERTBLD, SEECALC, SEEINPUT, TIMEME, VAL, VMARG
CALLED BY:	ENTMOD, PROCCOMP

Method:

The optimization of DGZs explicitly considers the time dependence of target value and the time of arrival of warheads. It does not reanalyze the correlation of delivery probabilities, which is assumed to have been treated in the cross targeting provided by the allocator. The selection of DGZs is a two-step process. First, the prescribed warheads are assigned initial coordinates through a "laydown" process in which each successive warhead is targeted directly against that target element where the highest payoff is achieved, taking into account collateral damage to all other target elements. Second, a general-purpose function optimizer FINDMIN is called which calculates the derivatives of the payoff as a function of X and Y coordinates of each weapon and adjusts the coordinates to minimize the surviving target value. FINDMIN will exit either after a maximum number of iterations (which are specified on an input card), or after it finds that it can no longer make significant improvements in the payoffs.

On the first call to DGZ, the specification of the maximum number of iterations for use in FINDMIN is set. Figure 71 illustrates the DGZ calling hierarch; the subroutines are grouped according to how they are used in the selection process. Figure 72 is the DGZ flowchart.

378
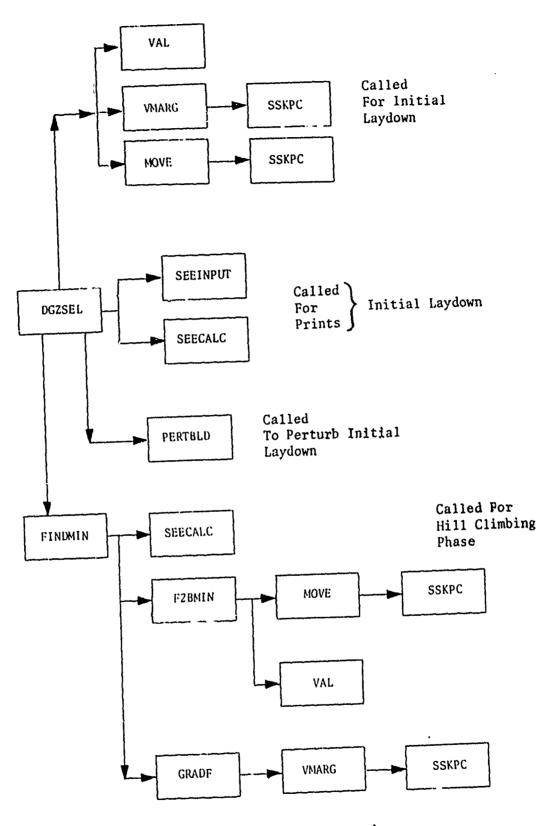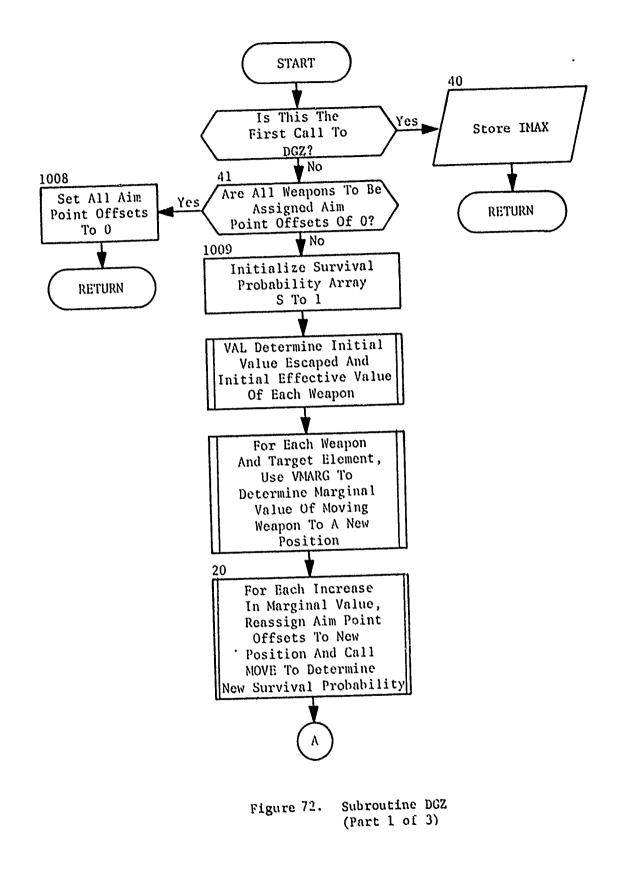
Figure 71. DGZ Calling Hierarchy

Figure 72.   Subroutine DGZ
(Part 1 of 3)

380
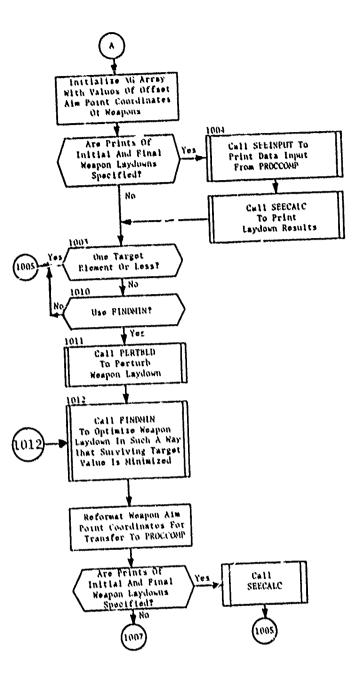
Figure 72. (Part 2 of 3)
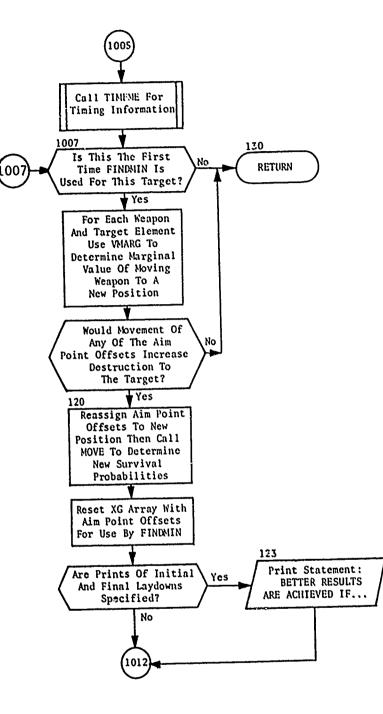
Figure 72.   (Part 3 of 3)

5.7.4 Function ERGOT1

PURPOSE:	To return next number in most uniform ergodic series. (Numbers for up to 10 distinct series can be called for concurrently.)
ENTRY POINTS:	ERGOT1, ERGOT2, ERGOT3
FORMAL PARAMETERS:	I - Index of the series for which the next number is desired
COMMON BLOCKS:	None
SUBROUTINES CALLED:	None
CALLED BY:	PROCCOMP, PERTBLD
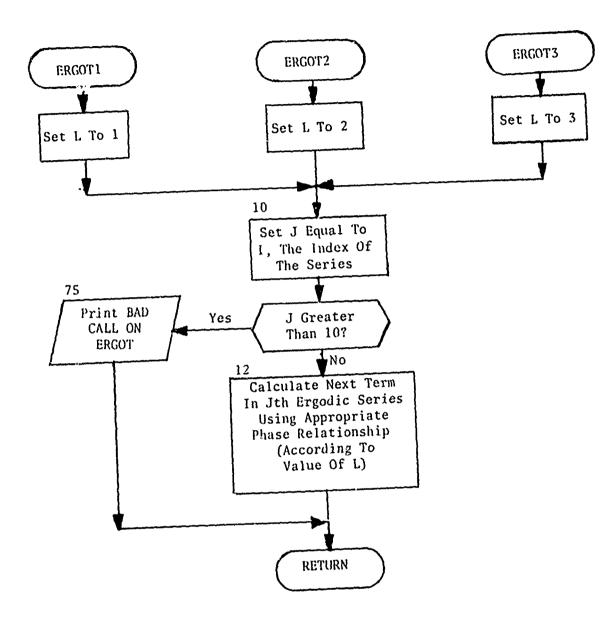
Method:

Depending on whether the entry ERGOT1, ERGOT2, or ERGOT3 is used, the index L is set to 1, 2, or 3, respectively. Then the next number in the Ith ergodic series is calculated. This function is illustrated in figure 73.

Figure 73. Function ERGOT1

5.7.5 Subroutine FINDMIN

PURPOSE: This subroutine uses a steepest descent method
            to determine a local minimum of a function of
            several variables. An initial estimate of the
            minimum position is input, together with various
            tolerances. FINDMIN uses two auxiliary routines,
            F2BMIN and GRADF, which define the function to be
            minimized and its gradient, respectively. DGZ
            uses FINDMIN to find the DGZs for complex targets.

ENTRY POINTS: FINDMIN

FORMAL PARAMETERS: XO   - Initial guess at aim point offsets
                      N   - Length of XO vector
                      IMAX - Maximum number of iterations for FINDMIN
                      E1,E2 - Tolerances for the minimization
                      X   - Best aim point offsets as determined by
                              FINDMIN
                      F1   - Minimum value found for escaped target
                              value
                      IFLAG - Print control flag
                              -2 : DGZSEL computation value print is
                                   produced
                             > 0 : FINDMIN debug prints will be pro-
                                   duced

SUBROUTINES CALLED: F2BMIN, GRADF, SEECALC

CALLED BY: DGZ
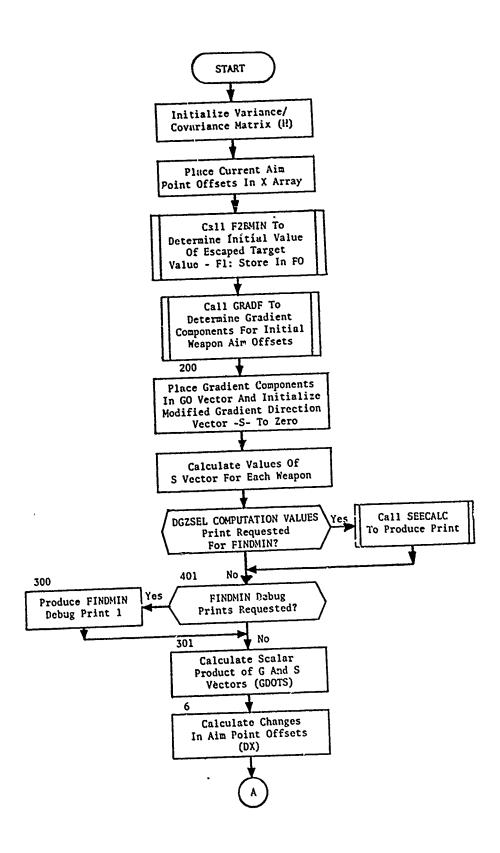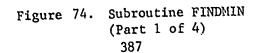
Method:

Given a function $F(X_1, X_2)$, the gradients $G_1 = \dfrac{\partial F}{\partial X_1}$, and $G_2 = \dfrac{\partial F}{\partial X_2}$, and
an initial guess (XO_1, XO_2) at the aim point offsets, FINDMIN finds the
local escaped target value F and its associated aim point offset coor-
dinates (X_1, X_2). Each iteration consists of a function, F, and gradient,
(G_1, G_2), evaluation followed by determination of the minimum function
value along the line associated with the modified steepest descent direc-
tion. F is redetermined at each iteration and is defined in such a way
that it converges after two iterations. FINDMIN uses two subroutines
during its processing. The first, F2BMIN, defines the escaped target
value function F in terms of aim point offsets X_1 and X_2. The second,
GRADF, defines the gradient components G_1 and G_2. In addition, at the

user's option, subroutine SEECALC is called after each iteration to print the results of the optimization.

Subroutine FINDMIN is illustrated in figure 74.

```
                    ┌─────────┐
                    │  START  │
                    └────┬────┘
                         │
              ┌──────────▼──────────┐
              │ Initialize Variance/│
              │ Covariance Matrix (H)│
              └──────────┬──────────┘
                         │
              ┌──────────▼──────────┐
              │  Place Current Aim  │
              │ Point Offsets In X Array│
              └──────────┬──────────┘
                         │
              ┌──────────▼──────────┐
              │  Call F2BMIN To     │
              │ Determine Initial Value│
              │  Of Escaped Target  │
              │ Value - Fl: Store In F0│
              └──────────┬──────────┘
                         │
              ┌──────────▼──────────┐
              │   Call GRADF To     │
              │ Determine Gradient  │
              │ Components For Initial│
              │  Weapon Aim Offsets │
              └──────────┬──────────┘
              200        │
              ┌──────────▼──────────┐
              │Place Gradient Components│
              │ In G0 Vector And Initialize│
              │Modified Gradient Direction│
              │   Vector -S- To Zero │
              └──────────┬──────────┘
                         │
              ┌──────────▼──────────┐
              │ Calculate Values Of │
              │ S Vector For Each Weapon│
              └──────────┬──────────┘
                         │
            ╱───────────▼───────────╲  Yes  ┌──────────────┐
           ╱ DGZSEL COMPUTATION VALUES╲─────▶│ Call SEECALC │
           ╲   Print Requested        ╱      │To Produce Print│
            ╲    For FINDMIN?        ╱       └──────┬───────┘
             ╲─────────┬───────────╱                │
          401     No   │                            │
                       │◀───────────────────────────┘
  300              ╱───▼────╲
┌──────────┐ Yes  ╱ FINDMIN Debug╲
│Produce FINDMIN│◀╲ Prints Requested?╱
│Debug Print 1 │  ╲──────┬───────╱
└─────┬────────┘    301  │ No
      │                  │
      └──────────────────┤
              ┌──────────▼──────────┐
              │ Calculate Scalar    │
              │ Product of G And S  │
              │   Vectors (GDOTS)   │
              └──────────┬──────────┘
              6          │
              ┌──────────▼──────────┐
              │ Calculate Changes   │
              │ In Aim Point Offsets│
              │       (DX)          │
              └──────────┬──────────┘
                         │
                    ┌────▼────┐
                    │    A    │
                    └─────────┘
```

Figure 74. Subroutine FINDMIN
(Part 1 of 4)

387

Figure 74. (Part 2 of 4)

Figure 74. (Part 3 of 4)

389

Figure 74. (Part 4 of 4)

390
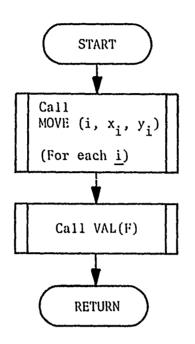
5.7.6 Subroutine F2BMIN

PURPOSE: F2BMIN defines the function which is to be minimized by FINDMIN. (FINDMIN minimizes survival probabilities of the target element and total escaping target value.)

ENTRY POINTS: F2BMIN

FORMAL PARAMETERS: XX - Vector containing offset coordinate for weapons
                             F - Total escaping target value for this weapon configuration

COMMON BLOCKS: .   C1

SUBROUTINES CALLED: MOVE, VAL

CALLED BY: FINDMIN

Method:

The offset coordinates for all weapons are input and a call on MOVE is made for each weapon to determine new survival probabilities. Then a call on VAL gives the new function value (total escaping target value) as well as the new effective values. The x, y coordinates of weapon I are given, respectively, by XX(2*I-1) and XX(2*I).

Subroutines F2BMIN is shown in figure 75.

```
                  ┌─────────────┐
                  │    START    │
                  └──────┬──────┘
                         │
                         ▼
        ╔════════════════════════════╗      (Determines Survival Probabi-
        ║  Call                      ║      lity for all Target Elements
        ║  MOVE (i, x_i, y_i)        ║      Relative to Each Weapon.)
        ║                            ║
        ║  (For each i)              ║
        ╚════════════════════════════╝
                         │
                         ▼
        ╔════════════════════════════╗      (Determines Effective Values
        ║                            ║      of Target Elements for This
        ║      Call VAL(F)           ║      Weapon Configuration and Gives
        ║                            ║      Total Escaping Target Value.)
        ╚════════════════════════════╝
                         │
                         ▼
                  ┌─────────────┐             (x_i, y_i) = Coordinates Assigned
                  │   RETURN    │             to Weapon i.
                  └─────────────┘
```

Figure 75. Subroutine F2BMIN

5.7.7 Subroutine GRADF

PURPOSE: GRADF determines the components of the gradient associated with the function which is to be minimized by FINDMIN.

ENTRY POINTS: GRADF

FORMAL PARAMETERS: XX - Vector giving weapon offset coordinates
                         G   - Vector computed by GRADF giving gradient components for each weapon

COMMON BLOCKS: C1
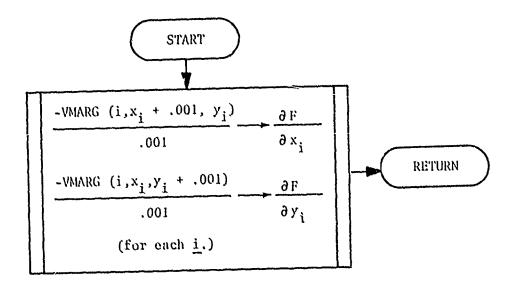
SUBROUTINES CALLED: VMARG
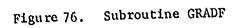
CALLED BY: FINDMIN

Method:

Two calls on VMARG for each weapon (one for each coordinate) are made to generate the gradient components.

Subroutine GRADF is illustrated in figure 76.

Figure 76. Subroutine GRADF
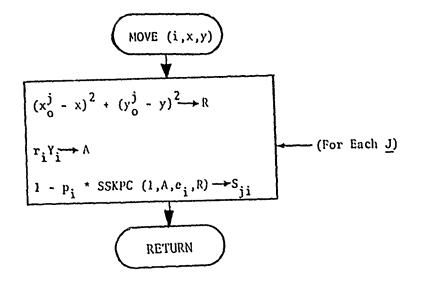
5.7.8 Subroutine MOVE

PURPOSE: Subroutine MOVE determines the survival proba-
                           bility, for all target elements, for a specific
                           weapon moved to a given position.

ENTRY POINTS: MOVE

FORMAL PARAMETERS: IM - Index for weapon
                           XM - X coordinate of weapon aim point offset
                           YM - Y coordinate of weapon aim point offset

COMMON BLOCKS: C1

SUBROUTINES CALLED: SSKPC

CALLED BY: DGZ, F2BMIN

Method:

For weapon IM with aim offset (XM, YM) the survival probabilities,
$S(J, IM)$, for each target element J, are redetermined using SSKPC.

Subroutine MOVE is illustrated in figure 77.

Figure 77. Subroutine MOVE

$$(x_o^j - x)^2 + (y_o^j - y)^2 \longrightarrow R$$

$$r_i Y_i \longrightarrow A$$

$$1 - p_i * \text{SSKPC } (1, A, e_i, R) \longrightarrow S_{ji}$$

(x, y) = Offset coordinates for weapon I.

r_j = Lethal radius of target J.

(x_o^j, y_o^j) = Coordinates of target J.

Y_i = Scaled yield of weapon I.

p_i = Probability of delivery of weapon I.

e_i = Error in delivery of weapon I.

5.7.9 Subroutine PERTBLD

PURPOSE: PERTBLD perturbs the weapon coordinates assigned
                     by the laydown algorithm in such a manner as to
                     assure a unique treatment by FINDMIN for each
                     weapon.

ENTRY POINTS: PERTBLD

FORMAL PARAMETERS: XG

COMMON BLOCKS: C1
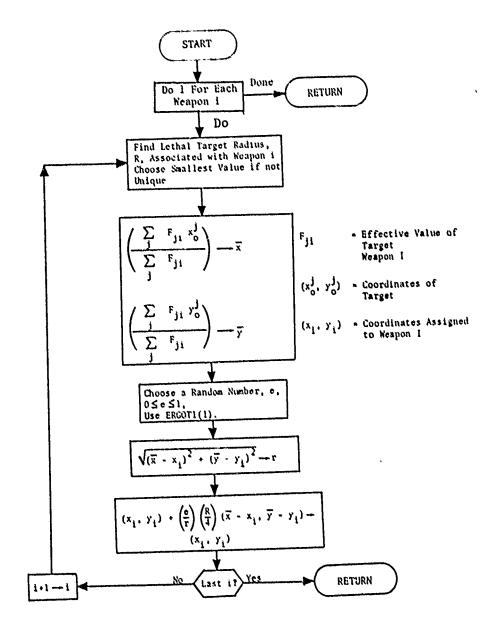
SUBROUTINES CALLED: ERGOT1

CALLED BY: DGZ

Method:

There is the possibility that, from some point on in time, all target
element values become constant. In this case, all weapons input to
FINDMIN with identical characteristics and later delivery times, which
have been assigned to the same target element by the laydown procedure,
would remain together. To eliminate this problem, subroutine PERTBLD
is called just prior to calling FINDMIN.

Subroutine PERTBLD is shown in figure 78.

START

Do 1 For Each
Weapon i → Done → RETURN

Do

Find Lethal Target Radius,
R, Associated with Weapon i
Choose Smallest Value if not
Unique

$$\left(\frac{\sum_j F_{ji} x_o^j}{\sum_j F_{ji}} \right) \longrightarrow \bar{x}$$

$$\left(\frac{\sum_j F_{ji} y_o^j}{\sum_j F_{ji}} \right) \longrightarrow \bar{y}$$

F_{ji} = Effective Value of
Target
Weapon i

(x_o^j, y_o^j) = Coordinates of
Target

(x_i, y_i) = Coordinates Assigned
to Weapon i

Choose a Random Number, e,
$0 \le e \le 1$,
Use ERGOT1(1).

$$\sqrt{(\bar{x} - x_i)^2 + (\bar{y} - y_i)^2} \longrightarrow r$$

$$(x_i, y_i) + \left(\frac{e}{r} \right) \left(\frac{R}{4} \right) (\bar{x} - x_i, \bar{y} - y_i) \longrightarrow (x_i, y_i)$$

Last i? — No → $i+1 \longrightarrow i$

Last i? — Yes → RETURN

Figure 78. Subroutine PERTBLD

398

5.7.10 Subroutine PROCCOMP

PURPOSE: To set up arrays in common block /C1/ for the com-
                        plex target so that the subroutine DGZ can use the
                        arrays during the selection of optimal aim point
                        offsets for the weapons allocated to the target;
                        and to modify target weapon assignments records
                        for inclusion of the computed offsets.

ENTRY POINTS: PROCCOMP

FORMAL PARAMETERS: None

COMMON BLOCKS: CITY, C1, C10, C30, ISKIPD, STRIKE, WPGT

SUBROUTINES CALLED: COMPRESS, CUMINV, DGZ, DIRECT, ERGOT1, ERGOT2,
                        HEAD, MODFY, NEXTTT, ORDER, REORDER, TIMEME,
                        VALTAR
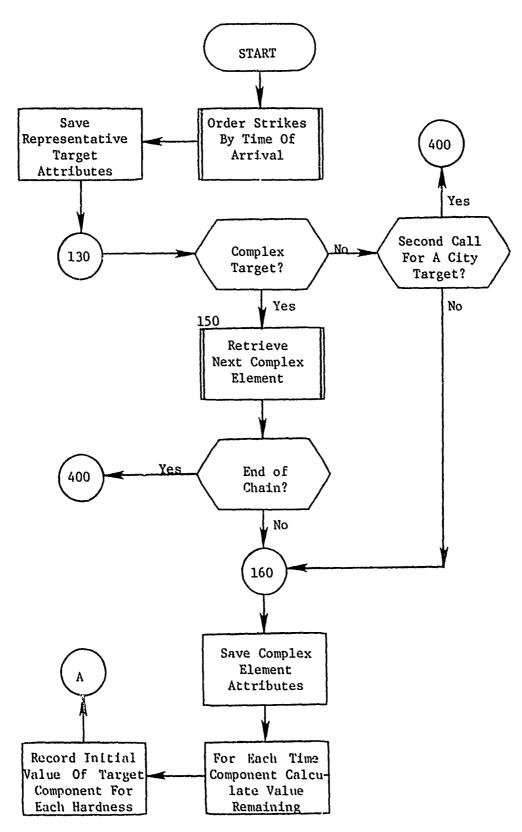
CALLED BY: ENTMOD
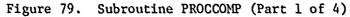
Method:
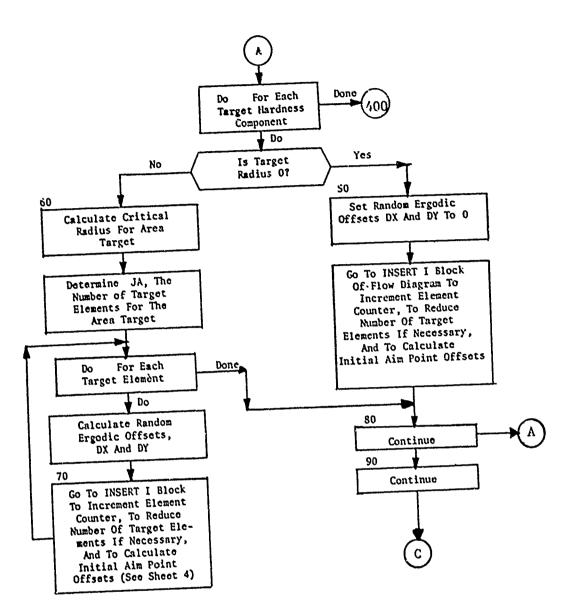
When ENTMOD encounters a complex target, PROCCOMP is called in order to
assemble data in a form that can be efficiently used for DGZ selection.
Each target component of the complex generates a standardized "target
element" in the working arrays used by subroutine DGZ (common /C1/).
Targets with more than one hardness component generate more than one
such target element, and targets with a specified target radius will
generate several elements spread over the area of the target to repre-
sent a value over the area. For complexes, individual target elements
are obtained by walking the data base chain called 'CMPTGT'.

If the number of target elements so generated exceeds the maximum pro-
gram dimensions (50), subroutine COMPRESS is called to recombine target
elements near each other having nearly the same lethal radius. In any
case, for efficiency in DGZ, a call to COMPRESS is made just before
calling DGZ. On return from DGZ, PROCCOMP modifies weapon assignment
records (ASSIGN) for definition of the computed offsets.

Subroutine PROCCOMP is illustrated in figure 79.

Figure 79. Subroutine PROCCOMP (Part 1 of 4)

Figure 79.   (Part 2 of 4)

401

```
                    ( 400 )
                       |
                       v
              +-----------------+
              ||      Call      ||
              ||  TIMEME(-2)    ||
              +-----------------+
                       |
                       v
            /---------------------\
      Yes  /      Are All          \
    <------<   Elements Of The       >
     |      \       Complex         /
     |       \---------------------/
     |                 | No
     |                 v
     |       +--------------------+
     |       |Call COMPRESS To    |
     |       |Reduce Number Of    |
     |       |Target Elements     |
     |       |Without Opening     |
     |       |   Tolerances       |
     |       +--------------------+
     |                 |
     |                 v
     |       +--------------------+
     |       |   Calculate Aim    |
     |       |   Point Offsets    |
     |       |      (DGZ)         |
     |       +--------------------+
     |                 |
     |                 v
     |       +--------------------+
     +------>|| Modify Strikes   ||
             ||  To Include      ||
             ||    Offsets       ||
             +--------------------+
                       |
                       v
                ( RETURN )
```
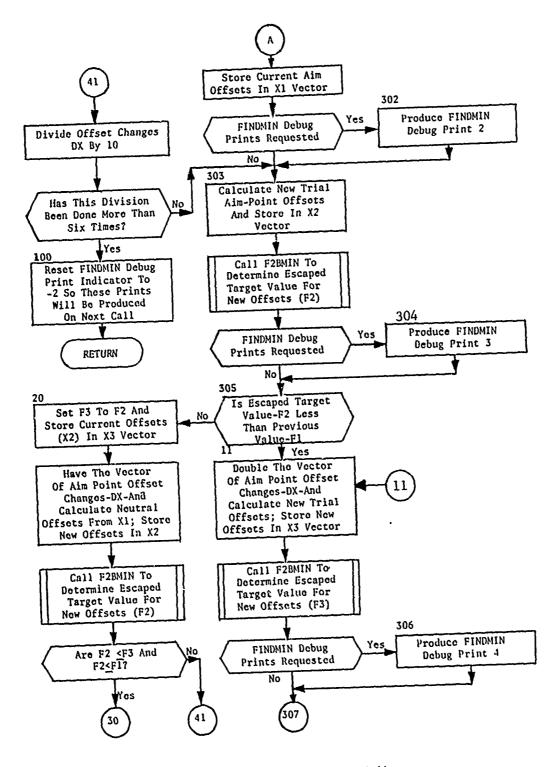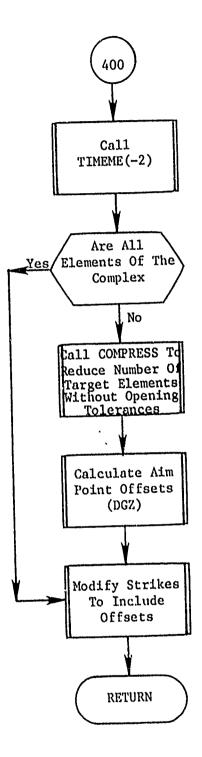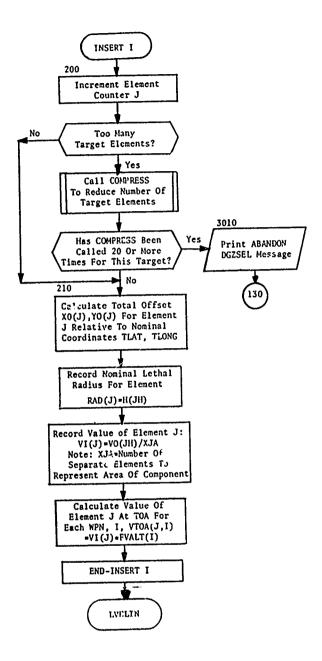
Figure 79.   (Part 3 of 4)

Figure 79. (Part 4 of 4)

5.7.11   Subroutine SEECALC

PURPOSE:	To print the computation values relevant to the selection of aim point offsets at various points within the DGZSEL subarea of program ALOCOUT.
ENTRY POINTS:	SEECALC
FORMAL PARAMETERS:	VESCTOT : Total escaping target value XX : Vector containing the aim point offset positions for the weapons
COMMON BLOCKS:	C1, WAROUT
CALLED BY:	DGZ, FINDMIN

Method:

When called by DGZ or FINDMIN subroutine SEECALC prints the title
DGZSEL COMPUTATION VALUES and column headings. Then for each weapon
allocated to the target, SEECALC prints the internal weapon number, the
aim point offsets, and the survival of each target element relative to
the weapon. At the end of the print for each target, the total escaped
target value is printed.

Subroutine SEECALC is illustrated in figure 80.
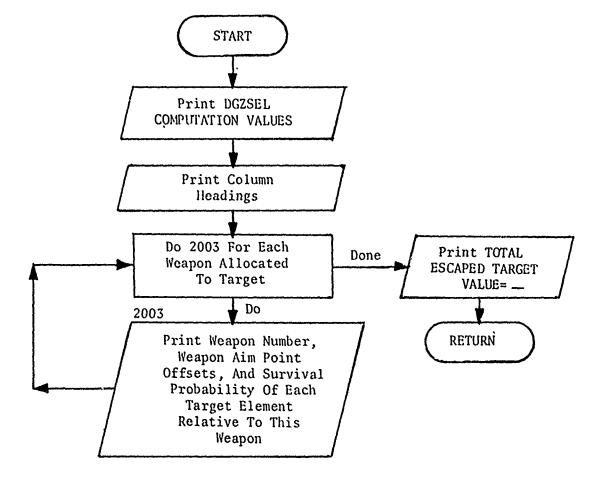
Figure 80. Subroutine SEECALC

405

5.7.12 Subroutine VAL

PURPOSE: VAL determines the target value which has escaped for a given weapon configuration and also determines the effective value, F_{ij}, for each target element as seen by each weapon.

ENTRY POINTS: VAL

FORMAL PARAMETERS: VESCTOT

COMMON BLOCKS: C1

SUBROUTINES CALLED: None

CALLED BY: DGZ, F2BMIN

Method:

This computation uses the effective values, VEFF(J,I), the survival probabilities, S(J,I), and the time dependent target values.

Subroutine VAL is illustrated in figure 81.

$$E_{ji} = \prod_{k=1}^{i-1} (S_{jk}) \; [V_j(T_{i-1}) - V_j(T_i)]$$

$$(j = 1, 2, 3, \ldots, NT)$$
$$(i = 1, 2, 3, \ldots, N+1)$$

$$F_{ji} = \frac{\sum\limits_{p=i+1}^{N+1} E_{jp}}{S_{ji}} . \qquad \begin{aligned} i &= 1, 2, \ldots, N \\ j &= 1, 2, \ldots, NT \end{aligned}$$

$$\sum_{j=1}^{NT} \sum_{i=1}^{N+1} E_{ji} \longrightarrow F$$

E_{ji} - Value of target element j
after arrival of all weapons

$V_j(T_i)$ - Value of target element j
immediately after arrival
of weapons 1 through i

S_{jk} - Survival probability of target
element j relative to weapon k

F_{ji} - The effective value of target
element j as seen by weapon i

F   = Total escaping target value=VESCTOT

Figure 81. Subroutine VAL

407

5.7.13 Function VMARG

PURPOSE:

Given a particular weapon configuration, function VMARG determines the marginal value of moving a specific weapon to a new position.

ENTRY POINTS:

VMARG

FORMAL PARAMETERS:

IT - Index weapon
XT - X coordinate of weapon aim point offset
YT - Y coordinate of weapon aim point offset

COMMON BLOCKS:

Cl

SUBROUTINES CALLED:

SSKPC

CALLED BY:

DGZ, GRADF

Method:
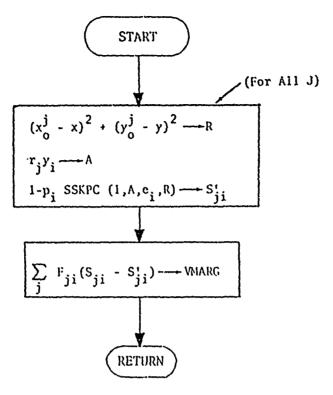
A modified set of survival probabilities for all target elements for this weapon is used to determine the marginal value.

Function VMARG is shown in figure 82.

(x,y) = Position of Weapon I

(x_o^j, y_o^j) = Coordinates of Target Element J

r_j = Lethal Radius, Target J

y_i = Scaled Yield, Weapon I

p_i = Probability of Delivery, Weapon I

e_i = Error in Delivery, Weapon I

S_{ji} = Survival Probability of Target J Relative to Weapon I

S'_{ji} = Survival Probability of Target J Relative to Weapon I when it is Assigned to Position (x,y)

Figure 82. Function VMARG

5.7.14 Subroutine WEPGET
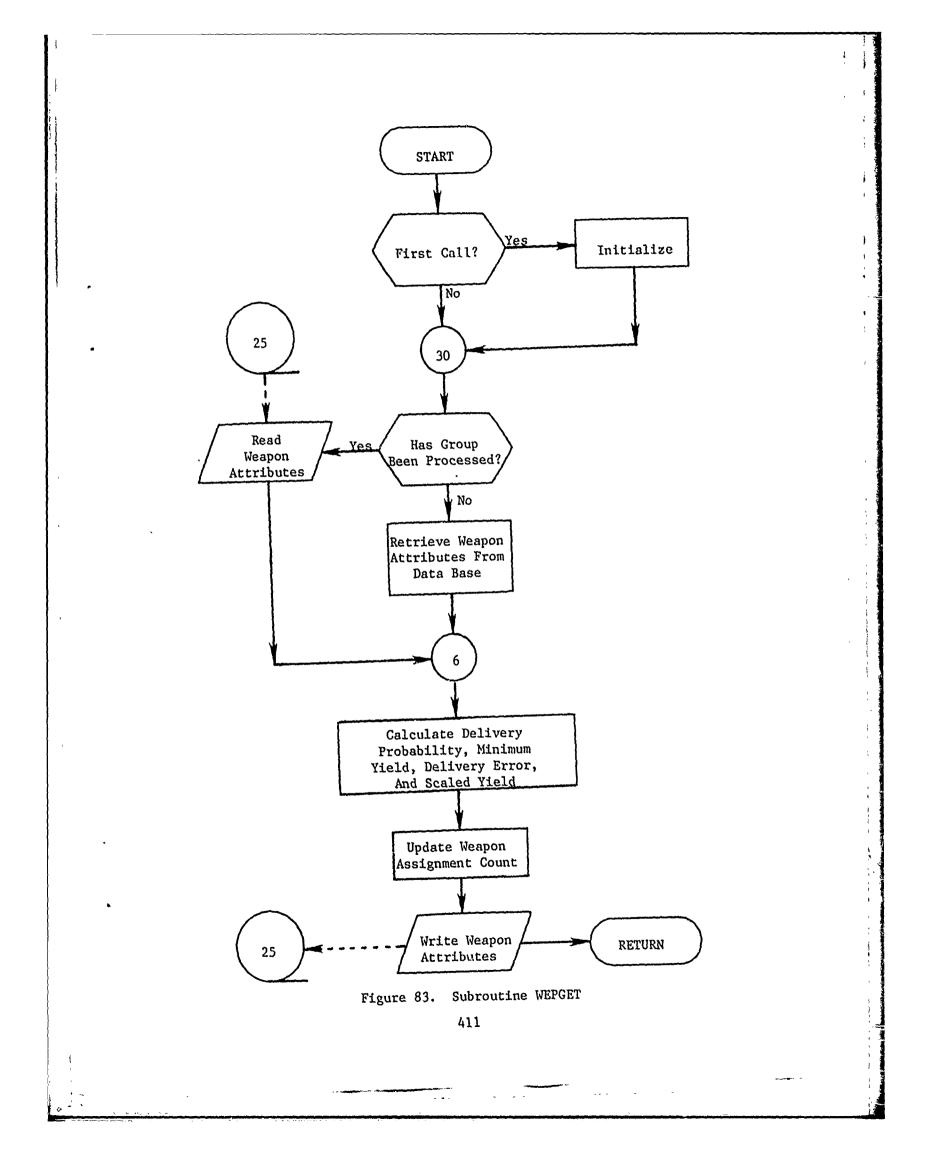
PURPOSE: Retrieve weapon attributes per assignment, and up-
                    date assignment counts based on corridor.

ENTRY POINTS: WEPGET

FORMAL PARAMETERS: None

COMMON BLOCKS: Cl, C10, C30, WPGT

SUBROUTINES CALLED: DIRECT, HEAD, NEXTTT, RANSIZE, TIMEME

CALLED BY: ENTMOD

Method:

Subroutine ENTMOD executes WEPGET for each target weapon assignment.
WEPGET retrieves weapon related attributes (YIELD, CEP, etc.) and de-
fines arrays in common block /Cl/ for use by subroutine DGZ. Also, a
count of each weapon assignment categorized by group and penetration
corridor is updated.

The weapon attributes necessary for calculating offsets for each strike
are indexed at the weapon group level. That is, individual strikes
launched from the same weapon group have the same attribute values.
Therefore it is necessary to interface with the data base only once per
weapon group request. Upon initial extraction, these attributes are
written on an indexed random file for future reference.

Subroutine WEPGET is illustrated in figure 83.

410

Figure 83. Subroutine WEPGET

411

5.8 Subroutine SUMPRN

PURPOSE: To sort weapon group assignment chains and print
                        out assignment summaries

ENTRY POINTS: SUMPRN

FORMAL PARAMETERS: None

COMMON BLOCKS: C10, C30, IONPRT, JAZ

SUBROUTINES CALLED: DIRECT, DLETE, HEAD, NEXTTT, ORDER, REORDER,
                        SORTIT, TIMEME

CALLED BY: ENTMOD
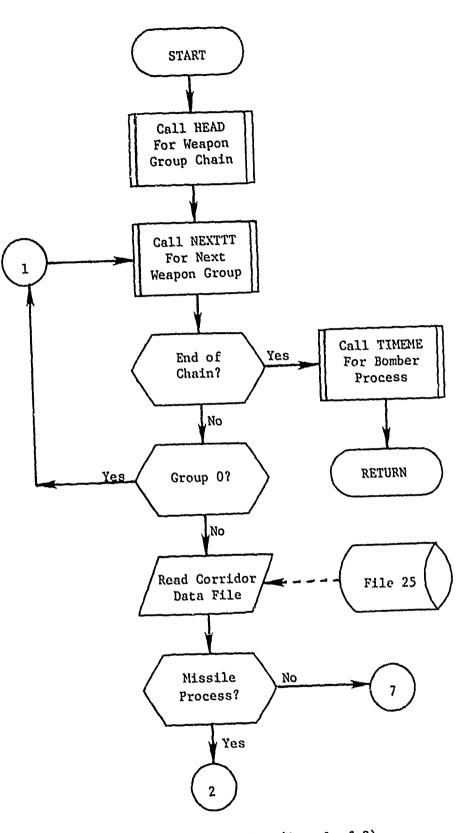
Method:

This subroutine replaces existing ASSIGN records which are in no parti-
cular order on the MYASGN chain with the same set of ASSIGN records in
the following sorts: For missile groups, assignments are sorted on
salvo number ascending, and, within salvo, on the value of the RVAL
attribute descending. For bomber groups, assignments are sorted on cor-
ridor with the corridor most often assigned occurring first, and, within
corridor, on the value of the RVAL attribute descending.

The method used is to cycle the weapon group chain and perform essen-
tially the same process for each group. First a record is read from
random access file 25. This record contains counts of the weapon groups
assignments. For missile groups the total number is in CORCNT(1) and
-1 in CORCNT(2). For bombers the contents of CORCNT(I) corresponds to
the number of assignments to corridor I. At this point, if the group is
a missile group, the print header is produced. If the group is a bomber
group, the assignments are totaled and the proper corridor order is
stored in CORORD.
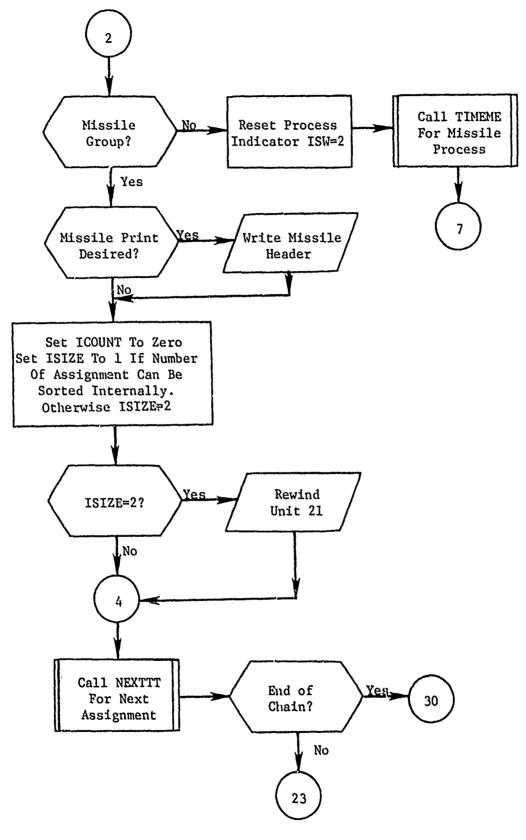
Now the assignments are read from the MYASGN chain and stored. The
method of storage depends upon the total number of assignments. If
the number is such that the data may be sorted internally, the data is
stored in array F. Otherwise, it is written onto a file. The sort keys
are created at this time. When the assignments have all been processed,
the old assignments are deleted from the MYASGN chain. Then the assign-
ments are sorted either internally by routines ORDER and REORDER, or
externally by SORTIT. Now each assignment, in sort is either read from
the SORTIT output or retrieved from array F. A new ASSIGN record is
stored in the MYASGN chain and the assignment is printed. The bomber
header is produced each time a new corridor is encountered.
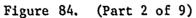
Subroutine SUMPRN is illustrated in figure 84.

412

Figure 84. Subroutine SUMPRN (Part 1 of 9)

413

Figure 84. (Part 2 of 9)

```
                        ( 23 )
                           │
                           ▼
                  ╔════════════════╗
                  ║  Call HEAD     ║
                  ║  For TARCDE    ║
                  ║  Record        ║
                  ╚════════════════╝
                           │
                           ▼
                  ┌────────────────┐
                  │  Save TARCDE   │
                  │ Reference Code │
                  └────────────────┘
                           │
                           ▼
                  ╔════════════════╗
                  ║  Call DIRECT   ║
                  ║  And HEAD For  ║
                  ║  Target Data   ║
                  ╚════════════════╝
                           │
                           ▼
                                                24
                    ╱────────╲           ┌────────────────┐
                   ╱  Missile ╲   Yes    │  Save Salvo    │
                  (  Process?  )────────▶│  Number As     │
                   ╲          ╱          │ First Sort Key │
                    ╲────────╱           └────────────────┘
                        │                         │
                       No│                        │
         20             ▼                         │
        ┌────────────────┐                        │
        │   Calculate    │                        │
        │   Recovery     │                        │
        │   Distance     │                        │
        └────────────────┘                        │
                 │                                 │
                 ▼                                 ▼
        ┌────────────────┐                     ( 26 )
        │ Save Corridor  │                        ▲
        │ Order As First │────────────────────────┘
        │   Sort Key     │
        └────────────────┘
```

Figure 84. (Part 3 of 9)

415

Figure 84. (Part 4 of 9)

416

Figure 84. (Part 5 of 9)

```
                    ( 2275 )
                       │
                       ▼
            ┌─────────────────┐
            ║ Call NEXTTT     ║
            ║ For Next        ║
            ║ Assignment      ║
            └─────────────────┘
                       │
                       ▼
              ╱─────────────╲                    ┌──────────────────┐
             ╱   End of      ╲      Yes          │ Set  ICOR As     │
            ╱    Chain?       ╲─────────────────▶│ Corridor Order   │
             ╲               ╱                   │ From CORORD      │
              ╲─────────────╱                    └──────────────────┘
                   │ No                                   │
                   ▼                                      ▼
                 ( 30 )                                 ( 23 )
                   │
                   ▼
              ╱─────────────╲      Yes          ┌──────────────────┐
             ╱               ╲                  ║ Call ORDER       ║
            ╱   ISIZE=1?      ╲────────────────▶║ And REORDER To   ║
             ╲               ╱                  ║ Sort Assignment  ║
              ╲─────────────╱                   └──────────────────┘
                   │ No                                  │
                   ▼                                     │
             ╱──────────────╲                            │
            ╱  Set 21 As      ╲                           │
           ╱   Sort Input      ╲                          │
           ╲   Unit           ╱                           │
            ╲────────────────╱                            │
                   │                                      │
                   ▼                                      ▼
            ┌─────────────────┐              ┌──────────────────┐
            ║ Call SORTIT To  ║              ║ Call NEXTTT      ║
            ║ Sort            ║─────────────▶║ And DLETE To     ║
            ║ Assignment      ║              ║ Remove Old       ║
            └─────────────────┘              ║ Assignment       ║
                                             └──────────────────┘
                                                      │
                                                      ▼
                                                    ( 32 )
```

Figure 84.   (Part 6 of 9)

Figure 84. (Part 7 of 9)

Figure 84. (Part 8 of 9)

Figure 84. (Part 9 of 9)

APPENDIX A

ALOC ANALYTICAL CONCEPTS AND TECHNIQUES

This appendix describes the major analytical concepts, techniques, and algorithms employed within the allocator (module ALOC). Topics of discussion consist of explanation of corridor routing, weapon/target interaction, weapon correlation, weapon allocation, derivation of Lagrange multiplier adjustment, and derivation of formulae for correlation in weapon delivery probability.

A.1 Corridor Routing
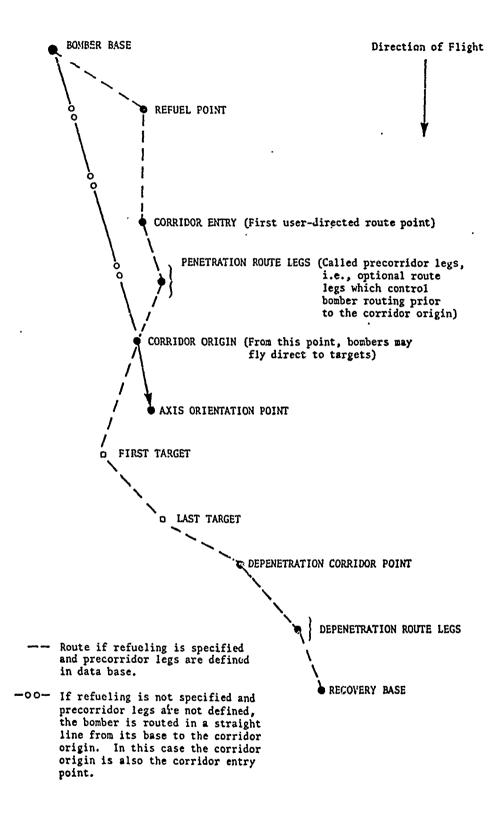
Penetration/Depenetration Corridors: In QUICK, bomber routing for penetration and depenetration of enemy territory is controlled by the use of flight corridors as reflected in figure 85. These corridors are established by the user and are defined in the data base. The user is permitted to specify a number (up to 30 per side) of alternative penetration corridors that can be used by the bomber force. A penetration corridor is defined by an entrance point and a corridor origin. From the corridor origin, the aircraft is permitted to fly in a direct route to the target. The corridor also has a specified orientation or axis, which is used to indicate the general direction of the defense suppression effort. There will be a tendency for bombers to penetrate more deeply parallel to the direction of the penetration axis than at right angles to it, since the attrition rate will be less (see Bomber and Missile Defenses, this appendix). The corridor axis is specified in the data base by a coordinate for the origin and a coordinate for the axis orientation point (denoted by the arrowhead in figure 85). In addition, the user may establish pre-corridor legs. This may be useful in order to avoid areas in which the expected attrition is high.
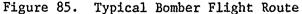
The user must also establish depenetration corridors which define the routing from enemy territory to a recovery base. A maximum of 50 depenetration corridors, each with up to four recovery bases, may be defined for each side. The system seeks, for each target, the most convenient depenetration corridor and associates it with the target. The depenetration corridor is specified in the data base by a depenetration point and one or more depenetration legs. The system will search from the last leg of the depenetration route and select an appropriate recovery base (see Detailed Sortie Specifications, this appendix).

Under the corridor concept, the routing of long-range strategic bombers is as follows. The aircraft is programmed to launch from its launch base; fly to a refueling area, if there is one; fly to the entrance of the penetration corridor; and fly down the corridor until it reaches the corridor origin. From this point, the bomber is permitted to fly in a direct route to the target. After the last target, the bomber is programmed to fly to the depenetration corridor entry point and fly down the depenetration corridor to a recovery base.

BOMBER BASE

Direction of Flight

REFUEL POINT

CORRIDOR ENTRY (First user-directed route point)

PENETRATION ROUTE LEGS (Called precorridor legs, i.e., optional route legs which control bomber routing prior to the corridor origin)

CORRIDOR ORIGIN (From this point, bombers may fly direct to targets)

AXIS ORIENTATION POINT

FIRST TARGET

LAST TARGET

DEPENETRATION CORRIDOR POINT

DEPENETRATION ROUTE LEGS

—— Route if refueling is specified and precorridor legs are defined in data base.

—oo— If refueling is not specified and precorridor legs are not defined, the bomber is routed in a straight line from its base to the corridor origin. In this case the corridor origin is also the corridor entry point.

RECOVERY BASE

Figure 85. Typical Bomber Flight Route

424

In actuality, not all bombers travel through geographic corridors to reach their targets. Two types, tactical bombers (those carrying nuclear weapons) and naval bombers (those restricted to attacking targets in class NAVAL), fly directly from their launch point (or refuel point) to their targets. However, to facilitate the creation of flight plans for these two types of aircraft, two dummy corridors (one for each type) are defined in the data base. While these corridors have no geographic significance, their assigned parameters do reflect the attrition to which aircraft will be subjected as they fly to their targets.

Corridor Attributes: The QUICK System allows up to 30 corridors per side to be used in a war game. Each corridor is associated with a set of type characteristics (attributes). These type characteristics, with exception of attrition on precorridor legs (attribute KORSTY), are used within the Plan Generator to establish the area attrition rates for bombers (see Bomber and Missile Defense, this appendix). Following is a description of the corridor attributes.

ATTRCO	Normal attrition rate for high-altitude aircraft using the corridor
ATTRSU	A reduced attrition rate for high-altitude aircraft applicable near the main axis of the corridor
DEFRAN	Typical range of interceptor aircraft on bases near a corridor (nautical miles)
HILOAT	The ration of low altitude attrition to high-altitude attrition (decimal fraction)
KORSTY	Attribute used to control the mode of corridor penetration (referred to as parameter k when used in the calculation of curvilinear coordinates--see Basic Sortie Generation, this appendix).

Bomber Defenses and Corridor Selection: In the case of bomber/area defenses, the penetration probability is estimated on the basis of the nominal attrition rates ascribed to the penetration corridors. Each corridor is ascribed at least two attrition rates:

ATTRCO	Normal attrition rate for high-altitude aircraft using the corridor
ATTRSU	A reduced attrition rate for high-altitude aircraft applicable near the main axis of the corridor.

In addition, attrition rates can be specified if desired for any prescribed legs between the entrance and origin of the corridor, and

425

attrition can be specified in the connection with penetration to defended
targets (TARDEFs). These attrition rates are used to estimate the pene-
tration probability. However, it is also assumed that the attrition
rates can be reduced by the factor HILOAT for portions of the route where
the aircraft can fly low. Any excess range available to the aircraft at
high altitude is used to provide a low-altitude flight -- assuming a
conversion factor RANGEDEC between low-altitude and high-altitude fuel
consumption. The estimated low-altitude range is then allocated among
the legs of the mission to minimize attrition.

To represent the effect which area and terminal defense will have upon
the successful execution of any bomber attack plan, a probabilistic
approach is used. The level of defense in a given area will directly
affect the probability that a bomber which travels through this area
will successfully reach its subsequent flight points. Therefore, each
section of geography over which bombers fly is characterized by attrition
parameters which reflect the level of area and local defenses for that
section. These parameters will, in turn, determine SURV(I), the proba-
bility that the bomber will survive to reach flight point I. Finally,
VALSORTY, the total value of a sortie, is defined as follows:

$$\text{VALSORTY} = \sum_{\substack{\text{all flight} \\ \text{points}}} \text{SURV(I)} * \text{V(I)}$$
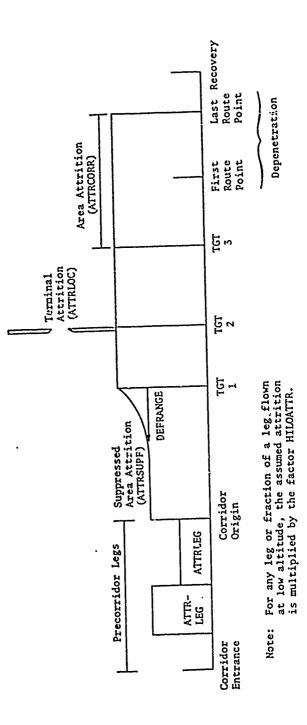
where V(I) = estimated value of reaching flight point I. This value
V(I) is the relative value RVAL generated during weapon allocation by
program ALOC (see Basic Sortie Generation).

The computation of SURV(I) for the formula is based on a simple expo-
nential attrition law. If the integrated attrition probability on each
individual leg to a point J is given by ATLEG(J), then the survival prob-
ability for the bomber to the point I will be given by:

$$\text{SURV(I)} = \text{EXPF} \left[- \sum_{J=1}^{J=I} \text{ATLEG(J)} \right]$$

The attrition ATLEG(J) includes both area and terminal attrition for the
leg. Figure 86 illustrates the attrition attributes and variables used
in the POSTALOC module.

The area attrition for each leg is computed by integrating the assumed
area attrition rate over the length of each leg. After the first tar-
get, this assumed area attrition rate per nautical mile is a constant,
equal to the data base variable ATTRCO supplied for the corridor.
Prior to the first target, the assumed attrition rate decreases expo-
nentially toward the limiting value ATTRSU which is also a data base
variable for the corridor. Thus the variable representing the assumed
area attrition rate between the origin and the first target is given by:

426

Figure 86. Illustration of Attrition Attributes and Variables (Used in Program POSTALOC)

427

$$Rate = ATTRSU + (ATTRCO - ATTRSU) * EXP (- X/DEFRAN)$$

where X = the distance in nautical miles between the corridor origin and the first target and DEFRAN is the typical range in nautical miles of interceptors on bases near the corridor. Attrition rates (ATTRLE) may also be specified for the precorridor legs leading into the corridor.

The terminal attrition ATTRLOC (see TGT2 in figure 86) is estimated directly from the data base variable TARDEF. Each potential target with a local (terminal) surface-to-air missile (SAM) defense is assigned the attributes TARDFH and TARDFL. The value assigned these attributes reflects the level of bomber defense, at high and low altitudes, provided by local SAM units. Considering the bomber's altitude (e.g., high) the local attrition ATTRLOC is estimated as follows:

$$ATTRLOC = .1*TARDFH$$

Naturally, this local attrition is of concern only when the route point characterized by this local attrition is itself a target for a bomb. It produces no effect if the target with which it is associated is attacked by an ASM (air-to-surface missile) that is launched from another route point. Moreover, even if the sortie definition indicates that the ASM is launched at the target from the vicinity of the target itself, it is assumed that the actual launch point will be such that the aircraft will not be required to penetrate the local defenses. Thus, any local attrition associated with the ASM target is again ignored. Finally, it is assumed that all local attrition is applied only to the incoming leg to the target and that on any leg or fraction of a leg flown at low altitude the attrition rates will be reduced by the factor HILOAT. In order to estimate the expected value of the sortie, therefore, an estimate must be made of how the available low-altitude range should be applied (discussed under Basic Sortie Generation, this appendix). Notice that a change in the assumed attrition rate for any leg or part of a leg will change the probability of survival to any point I (SURV(I)) which is required to evaluate VALSORTY.

During the weapon allocation phase (module ALOC), detailed sortie information (i.e., routing and sequential targeting) has not yet been generated. Therefore, bomber penetration of area defenses is treated as follows.

In weapon allocation, only one target is under consideration per vehicle. Therefore, in allocating low-altitude range among the legs of a mission to minimize attrition, much less weight on attrition is placed after the target has been reached. The algorithm assumes that the normal corridor attrition ATTRCO applies to the entire route from the target to depenetration, and to a portion of the route prior to the target equal to the perpendicular distance of the target from the main axis of the penetration corridor. The suppressed attrition ATTRSU is assumed to apply for the remainder of the route from the corridor origin to the target.

In computing the range of the aircraft, the normal range RANGE is used starting from the centroid of the weapon group for nonrefueled aircraft (IREFUEL=0) and from the specified refueling area for area type refueling (IREFUEL ≥ 0). In the case of buddy refueling, the refueled range RANGEREF is used, but distances are again measured from the weapon group centroid.

The penetration calculation is implemented by dividing the aircraft attrition elements into four "LEGS."

LEG = 1   Corridor entrance to origin (distance equal to sum of all such legs with attrition specified -- attrition equal to sum of attrition on all such legs)

LEG = 2   Corridor origin toward target as far as suppressed attrition (ATTRSU) is applicable

LEG = 3   End of LEG 2 to target -- ATTRCO applies but is augmented by any local attrition at a defended target TARDEF

LEG = 4   Target to depenetration -- ATTRCO still applies but value of mission and seriousness of attrition (RATE) is assumed to be less by a factor of approximately .25.

The available low altitude is then distributed among these legs, and the penetration probability is estimated. To select the preferred penetration corridor, a weight, .75, is given to reaching the target; the remaining weight, .25, is assigned to reaching the depenetration corridor. The corridor showing the highest value (Σ weight*penetration probability) is chosen, and the penetration probability to the target via that corridor is recorded for the group. If the group has been specified for nonrecovery (IRECMODE = -1, the recovery distance is simply set to zero.

On leg 3, the terminal attrition parameter TARDEF is modified by the parameter TARFAC. TARFAC is a user-input parameter which allows adjustment of the perceived terminal bomber defenses during program ALOC. The modified terminal bomber defense attrition is therefore defined as:

TARDEF x TARFAC

This attrition is ignored when calculating the delivery probability of an air-to-surface missile (ASM).

Missile Defenses: Ballistic missile defenses involve a simpler model. Only a random defense is considered for area attrition of missiles. Each warhead, regardless of its assigned target, has the same probability of being destroyed by the random area defenses. One random area kill probability is input for each side.

429

Terminal defenses are modeled by a subtractive model. Each target with terminal defenses is assigned a number of terminal ballistic missile interceptors. This number of interceptors (variable MISDEF) is input in the data base via the attribute NTINT which must be defined for each defended target.

The input variables describing the target's terminal defense capability allow uncertainties to be introduced in the number of interceptors present. MISDEF is the "nominal" number of interceptors on the target, each with kill probability PKTX against an unhardened warhead. In addition, four other parameters are defined (the same for all targets) which introduce uncertainties in MISDEF. RXLOW is a factor which, when multiplied by MISDEF, gives a lower estimate of interceptors which has probability PXLOW of occurring. Likewise, RXHIGH and PXHIGH define the overestimate of interceptor availability. Thus, if there is imperfect knowledge of the defense capability, the allocator can hedge against these uncertainties when assigning weapons.

In addition to the target-associated defense data, it is possible to describe penetration aids suitable for the various missiles by means of the Payload Table. For a particular payload index, the following variables* describe the penetration aids:

NWHD   =   Number of warheads per independent reentry vehicle package.

NTDECOYS   =   The number of "aim points" the terminal defense sees for each independent reentry vehicle (in addition to the warheads).

An independent reentry vehicle package is a set of warheads and terminal decoys that can be guided to a target point (or points) independently. For missile boosters with a multiple independently targetable reentry vehicle capability (MIRV), there may be several independent RVs per booster. Otherwise, each booster delivers one set of warheads and decoys.

The penetration probability of any warhead is a function of all the missiles allocated to the target. The model computes the total number of objects allocated to the target, NOBJ, as the sum of all warheads and decoys** allocated to the target. The number of perfect interceptors, variable PINT, is defined as:

------

* NWHD is data base attribute NWHDS; NTDECOYS is attribute NDECOYS

** For each weapon, this is the sum of NWHD and NTDECOYS multiplied by the product of the survival before launch probability, weapon system reliability, and command and control reliability.

$$PINT=PKTX*[(PXLOW*RXLOW)+(PXHIGH*RXHIGH)+(1-PXLOW-PXHIGH)]*MISDEF$$

This variable is the expected number of objects to be removed by the terminal defense interceptors.

The penetration probability for any warhead is defined as

$$1.0 - \left[\frac{PINT}{NOBJ}\right]$$

If this probability is less than (1.0 - PKTX) it is reset to that value.

A.2 Weapon/Target Interaction

The quality of the plans, in terms of realism and sophistication, will be a direct reflection of the realism incorporated in the payoff function. In order to produce plans of maximum realism, the payoff function should reflect all the major factors that would be considered by an experienced military planner. The design incorporates:

1. Time of arrival of weapons

2. Time dependence of target values, which can reflect a planner's uncertainty in the time of arrival of weapons relative to change in target value

3. Weapon range limitations

4. Uncertainty in target vulnerability

5. Correlations in the effectiveness of weapons of similar nature reflecting such factors as reliability, DBL probability, and defense effectiveness.

To evaluate the capability of any weapon group against any target, program ALOC requires six basic numbers. These are:

SBL(G)   The probability assumed that weapons in group G are not destroyed before launch

CC(KR)   The assumed command and control reliability associated with the region for group G

REL(K)   The assumed reliability for weapon type K used by group G

PEN(G)   The estimated penetration probability for weapons from group G to the target

SRK(G,JH)   The estimated kill probability of warheads in group G if delivered against the JH hardness component of the target

TVALTOA(G)   The estimated target value at the time of weapon
              arrival for weapon from group G (this factor is com-
              puted from the time of arrival for a weapon from
              group G, TOA[G]).

These numbers reflect the planning factors the user has specified for
the plan generation and do not necessarily reflect the values that the
user specifies for external simulation. The number is noted as "assumed"
where it is a direct user input supplied in the data base. It is de-
scribed as "estimated" where it is a derived quantity, based on other
input data.

Actually, the numbers reflect only two types of information -- the time
of arrival information, and the kill probability data. The single shot
kill probability is simply a product of the first five items. The break-
down of the single shot kill probability into these five separate fact-
ors, however, is required in order to estimate correlations in delivery
probability between several warheads delivered to the same target.

Most of the processing of weapon/target interactions deals with the six
quantities given above. These quantities are then used in the calcula-
tion of weapon payoff.

The basic payoff calculation is modified by the inclusion of weapon cor-
relation considerations. For each single weapon, four factors are cal-
culated: the single shot kill probability and three auxiliary quanti-
ties required by the correlation model (see Weapon Correlations, this
appendix).

If we define the overall single shot kill probability on one hardness
component J as: SSK = REL * CC * SBL * PEN * STK
then   MUP(G,J) = - LOGF(1.0 - SSK)
and   SSIG(G,J) - MUP(G,J)/-LOGF(SSK)

If the option to use the square root damage law is selected, MUP is
defined in a different manner. It is defined so that:

$$(1.0 - SSK) = \left(1 + \sqrt{MUP(G,J)}\right) * \exp\left(- \sqrt{MUP(G,J)}\right)$$

The use of the square root damage function is further explained in a
later section (see Multiple Weapon Attacks -- Square Root Law, this
appendix).

MUP is in effect a measure of the effectiveness of the weapon against
the specified hardness component. If all weapons were independent, the
survival probability for the component with respect to multiple weapons
IG would be simply:

$$EXPF -\left(\Sigma\ MUP(IG,J)\right)$$

(This is called the exponential damage law.)

432

If the square root law option is selected, then the survival probability would be:

$$\left(1 + \sqrt{\Sigma MUP(IG,J)}\right) \; *exp \left(- \sqrt{\Sigma MUP(IG,J)}\right)$$

The actual formula, using correlations, reduces to this form in the limit or no correlations but requires the array (SSIG(G,J) as an auxiliary quantity.

Estimation of Correlation Factors, RISK(A,G,J): The mathematics of the correlation calculation will be treated in detail below. Qualitatively, however, the technique requires an estimate of the extent to which the probability of failure for each weapon system is correlated with other weapons of the same class, type, alert status, etc.

The RISK array provides an estimate of this information. For any weapon system, the importance (or risk involved) in each failure mode (e. g., SBL, REL) can be represented in an additive form by taking the logarithm of the associated reliability. Thus, the total risk of failure for the weapon system -- LOGF(SSK) -- is given by: $\sum_{L=1}^{5} SM(L)$ where:

$$SM(1) = - LOGF(SBL)$$

$$SM(2) = - LOGF(CC)$$

$$SM(3) = - LOGF(REL)$$

$$SM(4) = - LOGF(PEX)$$

$$SM(5) = - LOGF(STK)$$

An array SMAT(A,L) is input by the user at the beginning of the allocation to provide a nominal estimate of the fraction of each risk SM(L) that is correlated with other weapons sharing each attribute A, where the attributes A represent:

A = 1   All weapons

A = 2   Weapons in the same group

A = 3   Weapons in the same region

A = 4   Weapons in the same class

A = 5   Weapons in the same type

A = 6   Weapons in the same alert status

For each weapon group G the RISK array by class, type, etc., is

433

estimated (for each hardness component J) simply as:

$$RISK(A,G,J) = \sum_{L} SM(L)*SMAT(A,L)$$

This simple technique for considering weapon correlation is used because it is a reasonable representation of correlation and the allocations do not seem very sensitive to the details of the correlations. Additionally, input data for a more detailed representation would be difficult to develop.

Adaptability of Input Data: The foregoing three arrays are derived from the basic six variables listed earlier: SBL(G), CC(KR), REL(K), PEN(G), STK(G,JH), and TVALTOA(G).

The techniques used to calculate these six basic quantities allow a great deal of flexibility to adapt to new concepts in timing and penetration strategy. Thus it can be expected that the specific form of their computations will change as experience is gained in actual applications of the program.

The computations now in use illustrate both the factors involved and the flexibility that is available. We will now consider the present techniques for computing these six variables.

Planning Factors -- (SBL, CC, REL): Two of the six (CC and REL) are contained directly in the data base. SBL is also in the data base -- except that the meaning there is probability of destruction before launch. To retain mathematical parallelism with other reliabilities, the SBL used here is defined as a probability of surviving and is obtained simply as (SBL = 1.0 - DBL). Obviously the specific value of DBL supplied in the data base should depend on both the alert status and the probability distribution of warning times for which the planner wishes to design the plan.

Evaluation of Value at Time of Arrival (TVALTOA(G)): The estimated target value at the time of weapon arrival for a weapon from group G, TVALTOA(G) is computed using the formula shown in the Time Dependent Target Value Subsection of the Planning Factor Processing Section of this chapter. TVALTOA(G) is equal to F(t) as calculated in the equation of that section, where t is the time of arrival of a weapon from group G, called TOA(G).

The time of arrival is computed differently depending on whether an initiative or a reactive plan is desired, and whether a missile or bomber is being considered.

In the case of a reactive plan it is assumed that all weapon systems launch as soon as possible (subject to their specified delays) after a decision to launch is made. The time of arrival in this case is computed as PLAN DELAY + LAUNCH DELAY + FLIGHT TIME. PLAN DELAY is either

434

the alert delay or nonalert delay (ALRTDL or NLRTDL) specified for the weapon in the data base. LAUNCH DELAY is computed as the product of LCHINT and (SALVO -1), where LCHINT is the time between successive launches as specified in the data base and SALVO is the salvo number of the launch (1 for first salvo, 2 for second salvo, etc.). (In the weapon allocation phase, all bombers are considered to be first salvo (LAUNCH DELAY = 0) to conserve storage because LAUNCH DELAY would be such a small fraction of total time of arrival.)

FLIGHT TIME is computed differently for bombers and missiles. For bombers this time is computed as FLIGHT DISTANCE/SPEED. For missiles, the FLIGHT TIME is calculated as described in the Missile Flight Time Calculation section of Appendix A, Program Maintenance Manual, Volume II. For missiles, the flight distance is computed as the great circle distance over a rotating earth from the weapon group centroid to the target. For aircraft, the distance is the sum of the great circle distances for each leg on the following path:

1. Weapon group

2. Specified refueling area (if appropriate)

3. Entrance to chosen penetration corridor

4. All specified intermediate route points for the penetration corridor (if any)

5. Origin of penetration corridor*

6. Target.

In the case of buddy refueling or nonrefueling, the second point on the path is omitted. (Note that the times of arrival used at this point are approximate, in the case of bombers, in that they use a constant nominal speed and do not allow for excursions to other targets on the way.)

In the case of an initiative strike, the times of launch are coordinated to reduce warning time. This is accomplished by coordinating the plan relative to an assumed warning time. In the case of alert missiles, the user specifies (in the parameter CORMSL) what fraction of the flight time should have elapsed at the coordination time. With CORMSL = 1.0 all missiles impact at the coordination time plus the LAUNCH DELAY described earlier. With CORMSL = 0.0 all missiles launch at the coordination time plus the LAUNCH DELAY described earlier. This

---

* Aircraft must fly to the origin of the corridor, but are not required to fly along the corridor axis to the corridor axis orientation point itself.

parameter is used in the weapon allocation phase. The sortie genera-
tion phase, which constructs the detailed plans, may use more sophis-
ticated CORMSL data to achieve more highly coordinated missile attacks.

In the case of bombers, the user specifies (in the parameter CORBOMB)
the remaining flight distance to the entrance of the penetration corri-
dors at the coordination time. For alert vehicles, launch times are
coordinated to make good this position at the coordination time --
except that no alert aircraft are held on the ground after the coordi-
nation time. The launch time and time of arrival for nonalert vehicles
differ from that for the alert vehicles by just the difference in the
alert and nonalert delays. In the sortie generation phase, the bomber
launch times are staggered according to the LAUNCH DELAY time described
earlier in this section.

<u>Penetration Probability (PEN)</u>: The computation of this factor is dis-
cussed in the section entitled Missile/Bomber Defenses, this appendix.

<u>Evaluation of Warhead Kill Probability (STK)</u>: The warhead kill proba-
bility is estimated as follows.

The lethal radius for a one-megaton weapon against the jth target hard-
ness component is computed using the VN function in program PLANSET and
is scaled to the weapon yield* using the 2/3 power yield area scaling
law. The kill probability is computed using the formula

$$P_R = 1 - \left[\frac{\sigma_D^2}{\sigma_D^2 + \frac{1}{2W} R_K^2} \right]^W$$

where

$$R_K = \text{lethal radius}$$

$$\sigma_D^2 = \sigma_{CEP}^2 + \sigma_{Tgt}^2$$

$$\sigma_{CEP} = 0.8493 * CEP$$

$$\sigma_{Tgt} = R_{95}/2.448$$

$$R_{95} = \text{radius containing 0.95 of total target value.}$$

---

* For gravity bombs and missiles this is the yield calculated for the
group. (See Basic Yield (Bombers) and MRV/MIRV Payloads sections in
this chapter.) For ASMs, this is the actual warhead yield.

For gravity bombs and missiles, the CEP is the CEP of the vehicles. For ASMs, the CEP is the CEP of the ASM as input in class ASM in the data base. The lethal radius is calculated in module program PLANSET for both ground burst and optimal air burst. The kill probability calculation uses the larger of these radii unless the user specifies (in module PREPALOC) the burst height (air or ground) for the target.

This kill function is computed from a very general actual-range/kill-probability law described in the Damage Function section of this appendix. When the parameter W equals 3, sigma-30 damage curves are closely approximated, appropriate to soft targets (below 15 psi); for W equal to 6, sigma-20 curves are approximated, appropriate for hard targets. The use of these sigmas is inherent to the VN system as outlined in Physical Vulnerability Handbook -- Nuclear Weapons (U), Defense Intelligence Agency (CONFIDENTIAL).

For weapons restricted to targets in class NAVAL, this calculation is not performed. The value of the attribute PKNAV is used as the single shot kill probability. (Note that these weapons are identified by a value of PKNAV greater than zero.)

Multiple Weapon Attacks -- Square Root Law: When a number of weapons attack a single target, there are two ways to consider the total expected kill probability: the exponential (or power) law and the square root damage function.

The exponential, or power, law considers the total survival probability to be the product of the individual survival probabilities. This law is not as appropriate for area targets as for point targets. The user therefore has the option to use a square root damage function on area targets; i.e., targets with a radius greater than zero. The square root law operates as follows: For each weapon i, define a factor K_i as follows:

$$P_S = \exp\left(-\sqrt{K_i}\right) * \left(1 + \sqrt{K_i}\right)$$

where P_S = probability that target survives one weapon of type i. (This K_i^S factor is called MUP in this program.) If we have N_i weapons of type i, then the survival probability of the target, assuming independent weapons, is

$$P_{S,N_i} = \exp\left(-\sqrt{N_i K_i}\right) * \left(1 + \sqrt{N_i K_i}\right)$$

If we have N_j weapons of type j also allocated to the target, the survival probability, again assuming complete independence, is

$$P_{S,N_i,N_j} = \exp\left(-\sqrt{N_i K_i + N_j K_j}\right) * \left(1 + \sqrt{N_i K_i + N_j K_j}\right)$$

437

The weapons are not usually considered to be completely independent. Thus, the sums, $N_iK_i + ...$, must be modified to consider interweapon correlations. The method of modifying this sum is discussed in the Weapon Correlations section of this chapter (also see Derivation of Formula for Correlations in Weapon Delivery Probability).

A.3 Weapon Correlations

A basic consideration underlying the need for cross targeting is the existence of "shared risks" between weapons--not only of the same type, but also between weapons of similar or related types. For example, if the enemy air defense is better than expected, the actual penetration probability of all bombers will be lower than that planned. If ballistic missile guidance systems prove to be operationally less accurate than expected, the target kill probability will be lower for all such missiles. These possibilities are illustrative of risks that are "shared" by large numbers of weapon systems. Cross targeting is intended to avoid "putting all eggs in one basket." It is designed to increase the probability that important targets will be destroyed even if most or all of the weapons with certain identical characteristics fail to perform as planned. Cross targeting recognizes the fact that operational percentages of success or failure for weapon systems cannot be predicted in advance.

The basic model used for cross-targeting analysis therefore recognizes that operational performance reliabilities are uncertain, and treats them as random variables. War plans are then developed on the assumption that the actual reliabilities that may be encountered in practice are unknown, and that they will in effect be selected at random for each weapon type from appropriate probability distributions. Moreover, it must be recognized that the reliabilities are not independently random for each weapon type, because certain risks are shared by many weapon types. Thus, on a specific Monte Carlo selection, when one success percentage is low, certain other percentages should tend to be low also. A satisfactory plan generation model also should be capable of considering these relationships between the success percentages for various weapon types.

To provide input data for the generation and evaluation of a cross-targeting plan, it is convenient to express these relationships in terms of risks that are shared in various degrees by similar weapon systems. The QUICK Plan Generator deals with five possible failure modes (table 12): survival before launch, launch or in-flight failure, command and control failure, penetration failure, and failure to kill the target even if delivered successfully. Each such failure mode can involve certain risks that are shared with similar weapons. For each such mode of failure, the user can specify the extent to which he feels risks will be of a type that are shared by all weapons of the same group, type, class, region, and alert status. Residual risks that are not specified to be shared in this way are treated as independent from weapon to weapon. Two weapons that share any attribute, such as type or alert status,

438

can have a certain amount of shared risk. The failure correlation model used in the QUICK system considers each weapon to have seven attributes over which to distribute the effects of the five failure modes. Table 13 shows the seven weapon attributes.

Associated with the attributes and modes is a matrix which specifies the fraction of the risk in each mode that is shared by weapons with the same attribute. This failure mode/attribute matrix, the SMAT array, defines the amount of risk shared by similar weapons and was referred to previously as the correlation array.

The entries in the matrix are the fraction of the risk of failure in the failure mode that is assumed to be shared by weapons with like attributes; e.g., class, type, region, and alert status. The sum of each row of the matrix must be 1.0. Two weapons in the same group that are identical with respect to all of these attributes will share identical risks except for the independent component. This array is used in the QUICK Plan Generator to compute weapon delivery probabilities and expected target damage when multiple weapons are assigned.

Nature of Uncertainties: The basic objective of cross targeting (using more than one weapon type against a target) is to increase the probability that the target will be destroyed even if most or all of the weapons of any given type fail to operate as planned. In other words, the cross targeting is intended to hedge against the fact that the operational target kill probability for any weapon type is uncertain. In the conventional oversimplified calculation of expected target destruction, uncertainty in the percentage of targets destroyed is assumed to arise only as a consequence of the random selection of statistically independent individual weapon successes and failures (which are assumed to be drawn from an ensemble of known overall reliability). However, in realistic planning situations, these individual weapon-to-weapon statistical variations account for only a very small portion of the total uncertainty in the percentage of successes that will actually occur.

There are numerous other factors over and above this simple statistical variation that introduce uncertainty in the actual percentage of weapon successes. In the present model, all of these factors, regardless of their actual cause, are lumped as contributors to a single uncertainty which represents total uncertainty in each of the various planning factors. Thus, within the model, the overall uncertainty is divided into two separate parts. First, for each planning factor (such as in-flight reliability, launch reliability, penetration probability, or probability of surviving destruction before launch), the uncertainty is modeled by defining a probability distribution for the reliability factor. For any specific war game, the actual reliabilities are considered to be drawn at random from these distributions. After the random selection of these reliabilities, there still remains uncertainty in the actual success percentage. This second uncertainty derives from simple statistical fluctuations in the success percentages that occur when independent successes and failures are drawn from an ensemble of specified overall

439

Table 12. Failure Modes

MNEMONIC	DESCRIPTION
SBL	Probability of survival before launch
CC	Reliability of command and control system
REL	Weapon system hardware reliability
PEN	Penetration probability
STK	Probability of target kill by warhead

Table 13. Weapon Attributes

NAME	DESCRIPTION
ALL	Shared by all weapons in the stockpile
ALERT	The alert status of the weapon, either alert or nonalert
CLASS	Weapon class, either bomber or missile
TYPE	Weapon type (e.g., B-52G, Poseidon)
REGION	Region of launch base
GROUP	Weapons of same class, type, region, and alert status whose launch bases are close to one another
INDEPENDENT	Shared by no two weapons in the stockpile

reliability. However, in realistic planning situations, this latter cause of uncertainty is usually relatively minor. The really serious uncertainties and, in particular, the uncertainties that give rise to the need for cross targeting, are above and beyond this simple statistical variability. The following are examples of some of these important factors that contribute to the uncertainty represented in the model by the probability distribution for each of the planning factors.

1. The enemy strategy and tactics are unknown and these can have major effects on the probability of penetration and the probability of destruction before launch both for individual weapon types and the force at large.

2. The basic system reliabilities in an operational environment may differ from those estimated in a test environment, and even the test environment reliabilities are not known exactly.

3. The actual success or failure percentages for one weapon may physically influence the success or failure probabilities of others--for example, in defense suppression attacks and in saturation tactics.

Weapon Failure Modes and Target Survivability: A programmed weapon can fail to destroy a target for a variety of reasons (failure modes) such as destruction before launch, launch failure, in-flight failure, penetration failure, or delivery inaccuracy. Assuming that these various failure modes are statistically independent, the overall reliability of the weapon h (from group $i(h)$) will be simply the product of the reliabilities over all the possible failure modes j;

$$R_j = \prod_j R_{i(h)j}$$

where

R_h = reliability for weapon h

$R_{i(h)j}$ = reliability for weapon h with respect to failure mode j

The target will survive the weapon h with probability

$$s_h = 1 - R_h = 1 - \prod_j R_{i(h)j}$$

Assuming for the moment that all weapons programmed against the target are statistically independent, the total probability of target survival is given by

$$S = \prod_h s_h = \prod_h \left(1 - \prod R_{i(h)j}\right)$$

441

In simplified analysis models where the reliability with regard to various modes of failure is assumed to be independent from weapon to weapon (i.e., where the operational reliabilities are assumed to be exactly predictable), this relation gives rise to a very simple law for target survivability with regard to multiple weapons. Specifically, relative to any target, one can define a single parameter X_h for each weapon h, where

$$X_h = - \ell n \ s_h$$

The X_h in this equation can be thought of as a measure of the strength of the weapon against the target. The probability of target survival is then given by

$$S = \exp \left(- \sum_h X_h \right)$$

This relationship is widely used in military analysis work. It has the advantage that the effectiveness of weapons against a target can be measured in terms of a single additive quantity, and the efficiency of a weapon relative to its value can be measured simply by comparing this quantity, X_h, with the weapon cost or shadow value.

However, as soon as one admits the possibility of uncertainty in the reliability factors or of dependence of the reliabilities between weapon types, the simplicity of this relationship is lost. Since the X_h are related, a simple sum will no longer suffice to determine target survival. The incremental effectiveness of each weapon depends in part upon the other weapons which have been programmed against the target. It is no longer correct to increase the sum in the exponent as each weapon is added. The entire expression for target survival must be completely reevaluated with each weapon addition. Thus, the previous equation must be expanded to the form

$$S = \exp \left[\sum_h \ell n \left(1 - \prod_h R_{i(h)j} \right) \right] = \prod_h \left(1 - \prod_h R_{i(h)j} \right)$$

The computational complexity of this expression for S in terms of the $R_{i(h)j}$, although undesirable, seems to be unavoidable in a practical cross-targeting model.

One obvious and superficially attractive way of avoiding the complexity, however, may require some comment. It has been suggested that the complexity can be avoided simply by considering the X_h as the random variables, and allowing the user to specify the statistical dependence between the X_h rather than the $R_{i(h)j}$. Unfortunately, because of the complex and unintuitive relationships between the X_h that result from mutually shared risks, this approach appears to place an impossible burden on the user.

442

A simple example will serve to illustrate this point. Consider two weapons, A and B, that share an identical risk of destruction before launch. Weapon A is otherwise completely reliable, and weapon B has numerous other more important failure modes. The small risk of pre-launch destruction is the only risk that prevents the X_h for the reliable weapon from being infinite. Thus, the destruction before launch risk completely determines the value of the X_h for the reliable weapon, but this same risk will have very little effect (even on a percentage basis) on the X_h for the less reliable weapon. Thus an _identical_ shared risk produces grossly different effects on the X_h for the two weapons.

It seems clear that if a model is to successfully deal with the statistical dependence between weapons, the user must be permitted to express the relationships in terms of the sharing of risks, and the consequences in terms of the X_h must be derived by the model. It is unrealistic to expect the user to supply information directly in terms of the X_h, even though this might simplify the mathematics.

Correlation Input Information: The preparation of correlation information for the QUICK Plan Generator is simplified for the user through the use of a hidden variable approach. The specific hidden variables employed are generalized so that they can represent broad aggregations of risk elements. This has the advantage that a standardized set of risk elements can be used, and it is not necessary to redefine a new set of hidden variables for each application of the system.

For the purpose of dealing with these risks, the QUICK system classifies all possible ways a weapon can fail (to destroy its target) into the five generalized failure modes described previously.

Each weapon in the QUICK system is considered to be a member of a homogeneous group of weapons which are considered to be identical with regard to all parameters used in the development of a war plan. The "weapon group" in turn is categorized as being of a particular: Class (bomber or missile); Type (minuteman, B-52, Polaris, etc.); Alert Status (alert or nonalert); and Command and Control Region. The various specific risk factors that can contribute to each of the five failure modes are also further classified as to whether they represent risks that might be shared in some degree: by all weapons of the same class; by all weapons of the same type; by weapons of the same alert status; or by weapons which share any other weapon attribute. Thus for each generalized failure mode, the QUICK system operates as if there is a hidden risk variable for each weapon attribute (see table 13, Weapon Attributes). By the conventions used in QUICK, the risks represented, for example, by the hidden random variable "Penetration Risk - Class Bomber" are available to be used only in the calculation of penetration risk for weapons that are members of the class "Bomber." Another risk variable is available to be used for penetration uncertainties by all weapons that are of class "Missile." If there are penetration risks that are relevant only for a subset of weapons within a class, there is another hidden variable for each type and even for each group that can be used.

443

The risk correlation information supplied for the QUICK system thus takes the following form. For each failure mode j and each weapon group i, an expected reliability $\overline{R}_{ij}$ is specified. The total risk, or variance, associated with this reliability factor is thought of as being divided into two parts, an independent risk and a shared risk. The shared risk is shared by all weapons in the group and is a result of the variance of the actual reliability R_{ij} relative to the expected reliability $\overline{R}_{ij}$. The remaining variance is identified as an "independent" risk which is completely independent from weapon to weapon in the group. The division of variance between "shared" and "independent" thus determines the width or uncertainty assumed by the Plan Generator for the probability distribution of R_{ij} relative to $\overline{R}_{ij}$. The larger the percentage of independent risk, the lower the uncertainty in R_{ij}.

The portion of the variance that is assumed to be shared within the group is then further subdivided into portions that are attributed to the hidden variables for weapons of that particular class, group, type, etc. Thus for each failure mode, the risk attribution required by the QUICK system consists simply of a specification of the portion of the total risk that is to be associated with each of a number of weapon attributes. Specifically, the user must specify the portion of the risk associated with each of the seven weapon characteristics previously described.

The summation of risk percentages attributed to each of the above factors must of course equal 100%. The following table illustrates a typical risk attribution array (SMAT) used as input to the QUICK system.

	ALL	GROUP	REGION	CLASS	TYPE	ALERT	INDEPENDENT
SBL	0	.10	.10	.40	.10	0	.30
CC	0	.20	.30	.10	.10	0	.30
REL	0	.05	0	.10	.20	0	.65
PEN	0	0	.10	.20	.20	0	.50
STK	0	0	0	0	0	0	1.00

The fact that 100% of the STK risk variable is treated as independent in this example implies zero uncertainty in STK; thus in this example we are ignoring any uncertainty in weapon yield or CEP. The choice of .30 for the independent component of the SBL as opposed to .65 for REL implies the assumption of greater relative uncertainty in any SBL reliability than is assumed in corresponding launch or in-flight reliabilities, REL.

Since, by definition, each row of this array must add to 1.0, the final column is obviously implied by the numbers in the other six columns. The actual input format for QUICK therefore omits the final column, so the correlation or risk attribution data are actually supplied in the form of a 5 X 6 array, known as SMAT. By convention, in supplying these data for QUICK, the array is normally filled with numbers intended to

444

represent the maximum amount of uncertainty or shared variance that it seems reasonable to consider.

One other important simplifying assumption is made concerning the risk attribution data supplied. In principle, one might think that the user would like to specify different risk attributions by class, type, alert status, etc., for every individual weapon group. This approach would provide maximum flexibility to control the factor weightings for each group, but it would require a separate SMAT array for each of the groups used (up to 250) in the QUICK system. To avoid this data burden, the QUICK system actually uses only one SMAT array and the values used in the array are chosen to be a reasonably good compromise for all weapon groups.

For missiles with a MIRV capability, a different weapon correlation array is created. The user specifies what fraction of the variance attributed to the INDEPENDENT attribute is to be added to the variance attributed to the GROUP attribute for all MIRV groups. This specification has the effect of increasing intragroup correlations for these groups. Since this increased correlation is applicable only to those events which precede booster burnout, only the failure modes which affect the booster are modified. These modes are survival before launch (SBL), command and control reliability(CC), and weapon system reliability (REL). Two SMAT arrays are stored, one for MIRV groups and one for non-MIRV groups. As each group is processed, the appropriate array is used in computing weapon/target interaction parameters.

A.4 Weapon Allocation

Weapon to target allocation is accomplished by one of two methods. The user may specifically assign some or all of the weapons identified for the war game; those weapons not specifically assigned by the user are automatically (mathematically) allocated by module ALOC. This program allocates weapons over the specified target system, using input data concerning the structure of the target system, the inventory and capabilities of available forces, and the war objectives and strategy. It produces as output a detailed specification of the weapons assigned to each target.

The structure of the target system is represented by the location, value, and estimated vulnerability and defense capability of each target element. The available forces are represented by such factors as range, yield, accuracy, reliability, penetration parameters, response time, speed, survivability, location of deployment, and inventory.

The allocator (ALOC) uses generalized Lagrange multiplier optimization techniques. With this approach, it is practical to use comparatively detailed payoff functions reflecting realistic uncertainties and planning contingencies that are usually ignored in automatically generated plans. The approach provides sufficient flexibility to include targeting objectives and constraints which may not have been foreseen in the original

445

formulation of the payoff function.

The objectives and strategy reflected by the plan will be determined by:

o   The relative values assigned to various elements of the target
     system, and the time dependence (if any) of these values

o   Any minimum required kill probabilities which may be specified
     for particular targets or groups of targets

o   The portion of the available force specified (such specifica-
     tion is optional) for allocation*

The realism and sophistication of the plans produced by such an optimiz-
ation depend in large measure on how completely the intended objectives
(with realistic contingency or uncertainty considerations) are reflected
in the payoff function. The design objective has been to provide the
flexibility needed for any reasonable payoff function. Some of the
factors included in the payoff function by the QUICK Allocator are:

1.   The time dependence of target values

2.   The uncertainties in target vulnerability

3.   Correlations in delivery probability between weapons which
      share the same uncertainties of accuracy, reliability, pene-
      tration probability, and weapon survivability (for the second-
      strike applications)

4.   The uncertainty in target value and time dependence -- as a
      consequence of the unpredictability of enemy actions

5.   Uncertainty in the level of ABM interceptors defending the
      target.

In addition, program ALOC computes the marginal value of each weapon
allocation. This value (RVAL), whose calculation is described in the
Basic Sortie Generation section of this chapter, is used in the sortie
generation process to determine the worth of including a target in a
sortie.

---

* The same types of information are used to control the resources allo-
   cated for defense suppression. In principle, the allocation of effort
   to defense suppression targets should be chosen to maximize the de-
   struction of other elements of the target system -- and should follow
   as an automatic consequence of the values assigned to these other tar-
   gets. However, such a ful   automatic treatment of defense suppres-
   sion is beyond the present state-of-the-art. Consequently, the user
   must specify equivalent values or required kill probabilities for
   defense suppression as well as primary targets.

Concept of Operation: The efficient targeting of a limited inventory of weapons is a combinatorial problem primarily because of inventory constraints. The fact that weapons used against one target are not available for others introduces a resource interaction between targets that are otherwise independent. The Lagrange optimization technique provides an exact representation of this interaction, which permits the allocation of weapons to be accomplished one target at a time. In the Lagrange technique, the detailed resource interaction is represented by a single "price" or value established for each type or group of weapons. This "price" represents the value of the weapons in each group in relation to the specific requirements and objectives of each war plan. This "price" (or Lagrange multiplier) corresponds to the minimum payoff (in target value destroyed) that will justify the use of the weapon.

The QUICK Allocator utilizes a resource allocation technique published in Operations Research* which permits the application of Lagrange multipliers to discontinuous or nondifferentiable functions (such as the payoff targeting problems).

As applied to the targeting problem, the technique consists of assigning a trial "weapon price" for each "group" of weapons in the inventory to be allocated. (A "group" is defined here as a set of weapons which are so nearly identical both in characteristics and location that no distinction between them is necessary during the allocation.) The attacker's "profit" on each target is then defined as the target value destroyed minus the total "price" of the weapon or weapons expended. Weapons are allocated against any target in such a way that this "profit" is maximized. (When the allocation against any target is complete, there are no weapons in the total inventory which could achieve an added payoff on the target in excess of their assigned "weapon price." Also, there are no weapons actually assigned to the target which do not achieve a payoff in excess of their assigned "weapon price.")

For missile groups which have a launch timing interval (attribute LCHINT) greater than zero, the price for each salvo within the group is modified as described in the Salvoed Group Multiplier Adjustment section of this chapter.

If the allocation were carried out this way for all targets, a certain total number of weapons from each group would be assigned. This number could be more or less than the actual inventory available. However, the resulting allocation would be a true optimum allocation for a hypothetical stockpile consisting of the weapons actually used in this allocation. If the number of weapons allocated from any group were larger than the

---

* H. Everett III, "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," Operations Research, Vol. II, No. 3, May-June 1963. p. 399-417. For ease of reference, an excerpt from this publication is contained in appendix B.

actual group inventory, then the trial "weapon price" is too low, and the use of these weapons should be limited to those places where a higher return is achieved. If too few were allocated, the trial "price" is too high, and the weapons could be fruitfully employed where the payoff is somewhat less. The trial "weapon prices" could then be adjusted accordingly and a new allocation could be carried out until a satisfactory approximation to the actual inventory is achieved. Many iterations throughout the target list would thus be required to establish the correct prices which would cause the desired stockpile to be consumed.

In the mathematical allocator, the basic process described above is speeded up in several ways:

1. The targets are processed in a random order, so that serious errors in the initial trial "weapon prices" are detected promptly and are corrected by observing the _rate_ of allocation for each group of weapons. Thus, it is not necessary to carry an allocation to completion before correcting the trial "weapon prices."

2. Initial allocation rates are monitored for aggregated categories of weapons (i.e., weapons which share identical attributes), rather than individual groups. Thus, statistically useful information on the allocation rates is obtained from small samples of targets, and corrections are applied to the "weapon prices" for all the weapon groups within the aggregated categories.

3. Ordinarily, in such a process, it would be difficult to estimate the size of the error in the "weapon prices" from the size of the error in the allocation rates. For example, a trivial difference in "weapon prices" between essentially identical weapons could cause the one with the lower "weapon price" to be used to the complete exclusion of the other. The QUICK Allocator therefore incorporates a small "premium" which prevents such large and unnecessary deviations from the desired allocation rates, where the difference in profit is small. With the premium, a large error in the allocation rates can occur only if the error in prices is substantial. In this way, the magnitude of the error in the "weapon prices" can be estimated from the allocation rates, and corrections of the proper size in the "weapon prices" can be efficiently made.

4. The iteration process in trial "weapon prices" is terminated when "weapon prices" are approximately correct (typically within a few percent) even though the resulting allocation does not accurately fit the available stockpile. The allocation is then adjusted to fit the stockpile by removing weapons excessively allocated and substituting weapons underallocated. This adjustment of the allocation is done by adjusting the "premiums" in the closing phase in such a way that the loss in "profit" is

448

kept as small as practical. It has been mathematically proven
in the preceding reference that the <u>payoff</u> for the resulting
allocation will not be degraded by this closing phase by more
than the <u>observed</u> loss of "profit."

This final approximation technique provides a powerful method for con-
verging rapidly on war plans which are near optimum. The extent of the
observed loss of "profit" provides a valuable gauge of the efficiency
of any such approximation. (If a rigorous bound on deviations from
optimality is desired, it can be obtained by a final pass over the tar-
get list in which all premiums are removed.)

<u>Adjustment of Multipliers</u>: To understand the operations of the allo-
cator (module ALOC), it is helpful to think of the set of all targets
arranged in random order around a circle. Processing will continue for
several "pases" around the circle until the multipliers have converged
to acceptable values, and the weapon stockpile constraints are met. To
start the process, initial values for the multipliers (i.e., "weapon
prices") are selected, and an initial pseudoallocation is made in which
the weapons are distributed uniformly (without regard for integer weap-
on constraints) over the target set. Thus, in the beginning it appears
that weapons have been allocated at exactly the right rate. As each new
target is encountered, the pseudo allocation is removed, and actual
trial allocation is made using the current values of the multipliers.
Since the initial multipliers are not correct, this gradually produces
an error in the estimated rate of allocation. This error is then used
to determine how to adjust the Lagrange multipliers. Of course, sta-
tistically significant information on errors in the allocation rates
becomes available most quickly for those groups where the number of
weapons is large. To accelerate the adjustment of the multipliers,
ALOC monitors the allocation rates for large collections of weapons
(i.e., weapons which share weapon attributes, see table 13) which include
many groups. When it is observed that the overall allocation rate for
such a collection is in error, the Lagrange multipliers for all the
groups involved are adjusted simultaneously. To simplify this, the
Lagrange multiplier, LAM(G), for each individual group of weapons is
expressed as a product of collective "local multipliers," LA(J). Speci-
fically, the Lagrange multiplier for a group of weapons is represented
as the product of the local multipliers for all weapons; all weapons of
the same class; the same type; the same region; the same alert status;
and a final local multiplier unique to the specific group; i.e.,

$$LAM(G) = LA\left(J_{all}\right) *LA\left(J_{class}\right) *LA\left(J_{reg}\right) *LA\left(J_{alert}\right) *LA\left(J_{group}\right)$$

The concept for monitoring the allocation rates is as follows. If
there are a total of NTGTS targets, and the total number of weapons in
a particular collection of weapons indexed by J (e.g., J_{all}, J_{class}) is

449

NOWPS(J), then the expected number of these weapons to allocate per target is just

$$\text{Expected Rate} = \text{NOWPS}(J)/\text{NTGTS}$$

If the observed rate is less, the associated multiplier LA(J) should be lowered; if it is greater, it should be raised.

Particularly during the early phase of the allocation, when the Lagrange multipliers ("weapon prices") are changing rapidly, the allocation rate will also change rapidly. Thus, in evaluating the allocation rate, it is appropriate to place more weight on the allocation rate for more recently processed targets. The estimators of allocation rate used by the allocator, therefore, allow a variable weight to be assigned to the targets. The estimated allocation rate R for any collection of weapons J is computed as follows:

$$R(J) = \frac{\sum_i N(i,J) * W(i)}{\sum_i W(i)} = \frac{\text{RUNSUM}}{\text{WTSUM}}$$

where $W(i)$ is the weight assigned to the ith target* and $N(i,j)$ is the number of weapons from the collection J assigned to the ith target. The summation is always taken over all targets. However, in the early stages of the allocation, the weight attached to each successive target is increased quite rapidly, so that the estimated allocation rate is determined almost entirely by the most recently processed 10 to 20 targets. As the Lagrange multipliers come closer to correct values, the target weights are increased more slowly and the allocation rate, in effect, is averaged over a larger number of targets. Ultimately, the weight attached to succeeding targets is held fixed. Obviously, after all targets have been processed with identical weights, the above estimator of the allocation rate becomes an exact measure of the average allocation rate and if multiplied by the number of targets would give the exact number of weapons on all targets. Thus, the same estimating machinery can be used in the final stage of the allocation as a guide in converging to the exact stockpile.

Actually, for each collection of weapons J, three separate estimators of the allocation rate are maintained. These estimators differ in the rate of change of the target weights that are used in computing the estimates. In effect, they correspond to averaging the allocation rate over different numbers of targets. The algorithm requires that all three estimates provide the same sign of the estimated error rate before it will change the value of the Lagrange multipliers. This feature provides a conservative approach to changes in the multipliers and reduces the chance of overcorrecting.

---

*Target weight is initialized at 1.0 and modified during processing, as described Calculations, Lagrange Multiplier Adjustment.

The allocation process evaluates its own progress in converging the multipliers and determines when to terminate the process. The variable which reflects this evaluation is called PROGRESS. PROGRESS is an arbitrary variable set internally by program ALOC to monitor the allocation state. The values 0, .4, .5, .75, 1.0, and 2.0 are arbitrarily assigned by the program according to procedures specified within ALOC. Qualitatively, the PROGRESS states are as follows:

1. PROGRESS = 0   This is the initial state. Its main purpose is to prevent the allocator from terminating very quickly because the pseudo allocation seems satisfactory.

2. PROGRESS = .4   This state indicates that the estimated allocation rates reflect primarily the actual rather than the pseudo-allocation.

3. PROGRESS = .5   From this point on, the rate of change of the target weight is not permitted to increase -- i.e., the allocation estimators are required to move monotonically toward the state where all targets are weighted equally.

4. PROGRESS = .75   Target weights have stopped increasing -- multipliers are assumed to be nearly stable.

5. PROGRESS = 1.00   This occurs only after at least one full pass of the target set with PROGRESS = .75. At this point the multipliers are frozen, and the premium (see below) for meeting the exact allocation is gradually increased. During this phase, multiple targets previously allocated as a unit may be split to receive independent allocations, if this will aid in meeting stockpile constraints.

6. PROGRESS = 2.00   Allocation is complete. Three options for further processing are provided depending on value of IVERIFY supplied by user.

   IVERIFY = 0   Current allocation simply transferred to normal output file, and process halts.

   IVERIFY = 1   Allocation transferred as above, but a verification allocation (not recorded on file) is made to obtain a bound on the maximum theoretical payoff if convergence had been continued indefinitely.

   IVERIFY = 2   Allocation transferred as above but the current allocation is reevaluated assuming a revised value of the correlation factor which is user-input at the start of the run (CORR2).

451

The details of multiplier adjustment are contained within ALOC.

Salvoed Group Multiplier Adjustment: QUICK forms weapon groups for the mathematical convenience of the allocation process. All weapons contained in a single group are physically identical insofar as their ability to create target damage. This fact is a critical assumption for the Lagrange multiplier process, because a single multiplier, or "price," is associated with each weapon in a group. Thus, after ALOC has run to completion, the weapon/target assignment does not distinguish among the weapons within a single group: this element of detail is added later, in program POSTALOC, (or FOOTPRNT for MIRV weapons).

The use of launch interval timing constraints, however, creates a fundamental change in this assumption, because it now is the case that different weapons from the same group can create different levels of damage on on a time-dependent target. This, in effect, requires that each salvo in the group have a different Lagrange multiplier, because it has a different effectiveness for the destruction of the target.

Consider, for example, the several missiles in a submarine. If these missiles are used to attack bomber bases from which the bombers are departing, then certainly the first salvo can create much greater damage than a later salvo. In the language of the Lagrange multiplier technique, the first weapon has a higher marginal utility and hence a higher multiplier (L). The allocation process will work only if the correct multiplier is assigned to each different weapon. However, for the case of the SLBM, we know there are certain relationships among these weapons and their multipliers. First, all the missiles are physically the same except for time of arrival on target. Second, the value of the weapons decrease with time (assuming the target value decreases in time, or that it is possible to delay a launch). Therefore, the value of the multiplier should decrease with time.

The multiplier for the Nth salvo (L_N) is a function of the multiplier for the first salvo (L_1), the salvo number (n), and the weighted density of salvo allocations (P). That is

$$L_N = \text{LAMGET } (L_1, P, N)$$

LAMGET is a function whose form is

$$L_N = L_1 - (P * (n-1)) * L_1$$

This is a straight-line function. The weighted density of salvo allocation P is also called a balance parameter. It is used to maintain the balance between salvo allocations.

The value of P is set internally by program ALOC according to the number of salvoes in a given group. There is one balance parameter P for each group with LCHINT greater than zero and has the form,

452

$$P = .1/(MAXSAL - 1)$$

where MAXSAL is the maximum salvo number in the group and .1 is a sensitivity parameter to slow the rate of change of P.

Closing Factors -- Premiums: The Lagrange multiplier for each weapon is modified by a premium. This factor is used to force closure of weapon allocations to the available stockpile. It acts as a bonus for using under-allocated weapons and a penalty for using overallocated weapons. The parameters which are used to calculate the premiums are:

SURPWP(G)   An estimate of the number of surplus (or unallocated weapons) in the group. This number is based on estimated allocation rates in the early phase and the actual allocation later.

NWPNS(G)   The actual number of weapons in group G.

CTMULT   The current multiplicity of the target being processed.

LAMEF(G)   The Lagrange multiplier for the group.

The premium depends also on three control parameters: PROGRESS, PRM, and CLOSE.*

The effect of PROGRESS (described earlier) is as follows:

1. If PROGRESS is greater than 1.0, this indicates that a verification allocation is desired to obtain a theoretical upper bound on the payoff without regard to meeting the actual stockpile constraints. For this purpose, the premiums are simply set to zero.

2. If PROGRESS is less than 1.0, a small premium is computed which is intended only to avoid large deviations from the desired allocation rate of small errors in the Lagrange multipliers.

    (Otherwise, a trivial change in the multipliers for two competing weapons could result in a complete change from always allocating one to always allocating the other.)

3. If PROGRESS is equal to 1.0, this is a signal that the closing phase has been reached and the object is to close in on an exact allocation of the available weapons. In this case, a larger step function premium is computed, and the size of the step function is gradually increased until final closure occurs.

---

* PROGRESS is set internally by the module as described in Section 3.

453

During the early allocation phase, superimposed on the actual payoff is a small negative quantity (called a premium) that is proportional to the value of each weapon group and quadratic in the size of the error in allocation. In effect, the actual payoff, $H(X)$, for any allocation, X is adjusted to $H(X)_{ADJ}$:

$$H(X) - PRM* \sum_G \left\{ NWPNS(G)*LAMEF(G)* \left(\frac{SURPWP(G)}{NWPNS(G)} \right)^2 \right\}$$

This quadratic addition to the payoff function has the effect of introducing a preference for allocations where the absolute value of SURPWP is small.

The addition or deletion of a weapon from group G will give rise to a difference in SURPWP equal to the current target multiplicity. Thus, the change in this quantity (per unit multiplicity) with the addition of a weapon G is:

$$PREMIUM(G) = PRM*LAMEF(G)* \frac{SURPWP(G) - .5*CTMULT}{NWPNS(G)}$$
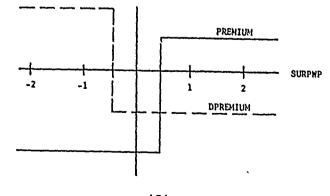
and the change with deletion of a weapon is:

$$DPREMIUM(G) = PRM*LAMEF(G)* \frac{-SURPWP(G) - .5*CTMULT}{NWPNS(G)}$$

The value of PRM is a user-input parameter. The value should be less than 1.0. Otherwise, in cases when no weapons from some group have been used, the premium for allocation of a weapon could exceed the cost of the weapon LAMEF(G) and weapons could be allocated even if the payoff were zero or even negative. Experience has shown that values between .5 and .9 work very well.

When PROGRESS reaches 1.0, PRM is set to .9 by the program to accelerate convergence. In addition, a small step function is added.

The following sketch illustrates the value of these step function premiums as a function of their SURPWP:

454

Notice that when SURPWP is in the desired area, that is SURPWP <.5, the premiums for either addition or deletion of a weapon are negative, making the current allocation seem most desirable. If there is a surplus of weapons (right side of figure), the premium for addition is positive, and the premium for deletion is negative. In the limit, if closure is long delayed, these premiums approach the value of the weapons. In this limit unallocated weapons seem free. The formula for these premiums is approximately:* LAMEF(G)*[1.0 - 1.0/CLOSE] where CLOSE starts at 1.0 and gets larger geometrically. The adjustment of CLOSE is controlled by another user-input parameter. CLOSE is adjusted linearly at a rate such that at the end of one pass it will have increased by the amount CLOSER (which is also a user-input parameter).

On the left-hand side of the figure, where weapons are overallocated, the premium for deletion is positive and the premium for addition is negative. These premiums can grow large without limit to provide incentive if necessary to remove a weapon from a very attractive target. The formula for these premiums is: LAMEF(G)*(CLOSE - 1).

Whereas the first set of premiums is linear and can be thought of as representing a negative quadratic addition to the payoff, these premiums are a step function and can be thought of as an upside down "V"-shaped addition to the payoff, which will strongly favor allocations that exactly match the stockpile.

Closing Factors -- Salvoed Missiles: An additional closing force is applied to missile groups with launch interval times (attribute LCHINT) greater than zero. In the final phase of weapon allocation when PROGRESS = 1.0, a stringent "first come first served" salvo selection method is used. When PROGRESS = 1.0, a salvo will not be allocated unless the salvo is currently underallocated to the extent that the allocation will not cause an overallocation. This closing force produces exact stockpile convergence within the salvoed groups within one pass through the target set at PROGRESS = 1.0.

Single Target Allocation--Targets Without Terminal Ballistic Missile Defenses: The problem is to select the best combination of weapons against each target as the targets are processed. The problem therefore is really a combinatorial problem. However, to calculate the payoff for all possible combinations of weapons and then select the best on each target would clearly be impossible. Consequently the method approaches the problem by adding one weapon at a time. After a weapon is added, the program estimates the additional payoff to be obtained by adding or, where relevant, deleting one weapon from any one of the available weapon groups. A decision must then be made whether to terminate

---

* Actually, it has been found desirable to add a very small quantity equal to ½ the smallest value of LAMEF(G) for any G multiplied by (CLOSE - 1.0). This provides an incentive for [SMALLAM*(CLOSE - 1.0)] using weapons with very low marginal value even if the payoff is essentially zero.

the allocation or whether to add or delete additional weapons. In its effort to maximize profit, the program operates initially on a form of steepest ascent basis. This is, it selects those weapons which provide the highest payoff per unit cost. It also removes any weapon which shows a negative profit after other weapons are added. There is a constraint, however, that every weapon on target destroy a minimum fraction of the target's original value. This minimum fraction is read in with the other control data. Ultimately it works solely on the basis of marginal profit and seeks any change in the allocation that will increase the profit.

Thus in effect the program needs to know the marginal profit for a potential weapon, the efficiency or payoff per unit cost, and the marginal profit of each weapon already on the target so that weapons which become unprofitable after others are added can be recognized.

The data required for these decisions are:

VT	The current surviving target value
VTP(G)	The potential surviving targe value if a weapon from group G were added
VTD(N)	The potential surviving target value if the Nth weapon now on the target were deleted.

The inputs required for their calculation include:

PREMIUM(G)	The current premium for adding a weapon from group G to the target
DPREMIUM(G)	The current premium for deleting from the target a weapon from group G together with the Lagrange multiplier
LAMEF(G)	The current Lagrange multiplier or cost associated with the utilization of a weapon from each group.*

Using these input arrays, the program computes the potential "BENEFIT" associated with the addition of a weapon from any of the weapon groups. The BENEFIT is interpreted simply as the payoff plus the premium; i.e., for potential weapons, BENEFIT = VT-VTP+PREMIUM. Similarly, for each weapon that might be deleted, there is computed the BENEFIT that would be lost if the weapon were deleted, BENEFIT = VTD-VT-DPREMIUM. Notice that if the premiums are small (as they usually are) the benefit is essentially the same as the payoff. It is, therefore, convenient to

---

* For missile groups with a launch interval time (attribute LCHINTVL) greater than zero, the basic multiplier is modified as described in the Salvoed Group Multiplier Adjustment section of this chapter.

think of the BENEFIT as simply a modified payoff that is to be maximized. The PREMIUM is added simply to speed the convergence to the desired stockpile.

The program scans the potential BENEFIT associated with all weapon groups that might be added and finds that group IPPMX for which the "modified potential profit," PP, is greatest; i.e., PPMX, PP = BENEFIT - LAMEF.

Similarly it reports the group IPVRMX for which the "efficiency," PVR is greatest, PVRMX. The "efficiency" is here interpreted as the rate of BENEFIT per unit cost; i.e., PVR = BENEFIT/LAMEF. (It is necessary for the single target allocator to know the "efficiency" of alternative weapons. If it were guided only by "profit" (i.e., (BENEFIT - LAMEF), it would always select those individual weapons showing the largest profit, whereas it is often better (especially on very valuable targets) to select several less costly weapons so long as the benefit per unit cost is higher.)

Finally, the program scans all weapons, already on the target, to determine which weapon IDPMN shows the smallest DPMN marginal modified profit DP where DP = BENEFIT - LAMEF.

These quantities:

VALUE	INDEX	DEFINITION
PPMX	IPPMX	Maximum potential profit
PVRMX	IPVRMX	Maximum potential efficiency
DPMN	IDPMN	Minimum current marginal profit

constitute the primary input for determination of weapon allocation on single targets. There calculation is modified, however, by the minimum and maximum damage constraints placed on each target. MINKILL is the minimum required damage level. MAXKILL is the Maximum desired damage level. MAXCOST is the maximum factor by which value may be multiplied to obtain MINKILL (these three factors are established in the data base: MAXKILL and MINKILL are defined as attributes; MAXCOST is set equal to the attribute MAXFRACV). MINDAMAG, a program user-input parameter, is the minimum fraction of damage required from an individual weapon.

To implement the MINKILL and MAXKILL responsibility, the VT, VTP, and VTD are replaced by effective values VTEF, VTPEF, VTPEF, AND VTDEF. The relationships are:

$$VTEF = ALPHA*MAX1F(VT,VTMIN)$$

$$VTPEF = ALPHA*MAX1F(VTP,VTMIN)$$

$$VTDEF = ALPHA*MAX1F(VTD,VTMIN)$$

(Note: MAX1F implies "Maximum of")

where: VTMIN = VTD*(1.0 - MAXKILL)

ALPHA = Local control variable defined below.

If neither MINKILL nor MAXKILL has been explicitly specified for the
target then the default values apply (ALPHA=1.0 and VTMIN-0.00) and
the effective values of VT, VTP, and VTD are identical with the actual
values. If MAXKILL has been specified as less than 1.0, it implies
there is no value in reducing the target value below VTMIN. This point
of view is built into the payoffs simply by not allowing the effective
value to reflect any surviving target value less than VTMIN.

The variable ALPHA is increased above 1.0 when necessary to motivate
the algorithm to achieve the specified MINKILL (minimum acceptable
fraction of expected value destroyed). A quantity VTMAX is defined

VTMAX = VTO*(1.0 - MINKILL)

which reflects the largest acceptable expected surviving target value.
If the computed surviving target value VT exceeds VTMAX, and at the same
time the output does not show any additional potentially profitable
weapons, then the process will not terminate immediately. It will
instead increase the value of ALPHA above 1.0 by whatever factor nec-
essary to make at least one more weapon seem profitable. It then recy-
cles and reevaluates all the output parameters. Since ALPHA multiplies
all the target values, increasing ALPHA is equivalent to increasing
the value of the target until more weapons can be justified against it.
Once the value has been raised so that the required kill is achieved,
ALPHA remains fixed (for this pass) during the remainder of the alloca-
tion to the target, so that the program automatically proceeds to do a
complete optimum allocation for the revised target value.

There is a protection feature MAXCOST that is designed to prevent exces-
sive waste of warheads against a target where it is simply not practi-
cal to achieve the prescribed destructive level required by MINKILL.
If the current cost (of the allocation to the target) divided by the
total target value already exceeds the ratio prescribed by MAXCOST, the
value of ALPHA will not be increased any further. For the same reason,
if it is necessary to raise the target value by a factor of 100 or more
to justify the specified MINKILL, the ALPHA will not be further increased.

Experience with the allocator has shown that if the efficiency PVR is
used in its pure form, PVR = BENEFIT/LAMEF, the program will sometimes
arrive at its allocation in a very inefficient way. What happens is
that during the initial laydown of weapons on the target it will use
large numbers of very cheap but not very effective weapons. Then as
soon as a more efficient weapon is used, the target value is drastically
reduced and many of the weapons initially allocated cease to be worth-
while and have to be removed. Consequently, the program now incorporates

a revised version of the efficiency PVR'. This is defined as follows:

$$PVR' = \begin{cases} PVR & \text{if } PP<0 \\ \left[1.0 + \dfrac{PP}{LAMEF} * \dfrac{1 + \gamma(VTEF/(VTEF-PP+PREMIUM))}{1+\gamma}\right] & \text{if } PP \geq 0 \end{cases}$$

If γ is zero this gives the pure value of PVR. However if γ is set above zero, as it usually is, then the value of PVR will reflect the magnitude of the profit as well as the efficiency. (This coefficienct, γ, is a user-input parameter.) Notice that as the potential profit PP becomes comparable to the remaining target values, the coefficient of γ in the numerator becomes large and PRV' is increased above PVR. In the limit where the potential profit PP is negligible relative to the remaining target value VTEF, PVR' is equal to PVR. The single target weapon allocation procedure consists of three parts:

1. A set-up and single weapon allocation phase

2. A multiple weapon laydown loop

3. A multiple weapon refinement loop.

The initial laydown operations are handled using the "efficiency" as the criterion for selecting weapons. This is necessary because if the "profit" were used at this stage, effective individual weapons which could produce a large single weapon profit would always be selected in preference to less effective but less expensive weapons where two or three such weapons added in succession might provide a better payoff at lower cost. However, before exiting from the routine, provision is made to test the allocation to determine whether a higher total profit is possible. So, the final refinement of the allocation is always done using total "profit" as the criterion.

An immediate exit is made if there are no potential weapons that show a profit. Otherwise, the weapon which shows the highest "efficiency" is added. A test is then made to determine whether more weapons are needed on the target. If so, control passes to the multiple weapon laydown loop. If not, it is clear that a single weapon allocation is needed. In this case, if the single "efficient" weapon just tested is not also the most profitable weapon, then it is removed and replaced with the most "profitable" single weapon before exiting from the routine.

On the other hand, if several weapons are indicated, the multiple weapon laydown loop takes over. This loop simply keeps adding the most efficient next weapon until there are no more potential weapons that show a profit; i.e., have an efficiency greater than one. (For a profitable weapon, (BENEFIT/COST) must exceed 1.0.) As new weapons are added, however, it often occurs that some of the old weapons cease to be profitable; provision is therefore made to remove any unprofitable weapons after each new weapon is added. When this part of the process is

complete, all weapons on the target must be "profitable" and there must be no potential weapons that would show a profit if added.

At this point, there is a remote possibility that there is again only one weapon in the allocation. If so, it is replaced with the most profitable single weapon. Otherwise, control passes to the allocation refinement loop.

Basically, the allocation refinement loop is intended to start back with the first weapon placed on the target and successively remove each weapon to determine if there is any more profitable weapon that can be substituted. If, in each case, the same weapon proves to be the most profitable the allocation is considered complete. If, in any case, a substitution occurs, the testing of the other warheads starts over again from that weapon until all weapons on the target have been tested.

It is possible during this process, as in the preceding loop, that as more profitable weapons are substituted, some of the other weapons that formerly were profitable will cease to be so. Therefore, after each weapon is added, a check is made and any unprofitable weapons are deleted. If such deletion leaves a situation where some other weapon would be profitable, it is immediately added before reentering the testing loop. Any such change that interrupts the testing process requires that the testing start over again. To avoid unnecessary operations, the pointer which selects successive weapons to be deleted for testing is set to skip over weapons which are from a weapon group that has already been tested.

Single Target Allocation -- Targets With Terminal Ballistic Missile Defenses: The allocator (module ALOC) considers two possibilities for targets with terminal BMD. It first attempts a leakage attack. A force, possibly mixed between bombers and missiles, is allocated without trying to exhaust the missile defense. Any bomber or missile weapons that leak through their respective terminal defenses are considered in evaluating damage. Second, the allocator attempts an exhaustion attack. A force of missiles large enough to exhaust the terminal missile interceptors is allocated. After exhaustion of the defenses, missiles are added until the damage done by each incremental missile is less than the value of the Lagrange multiplier for that missile.[*] The profit from these two attacks is compared and the more profitable allocation is chosen.

The rate of return for a missile against a target with terminal BMD is defined as follows:

$$RATE = (VT - VTDX)/(LAMEF + PREMIUM)$$

---

[*] For missile groups with a launch interval time (attribute LCHINT) greater than zero, the basic multiplier is modified as described in the Salvoed Group Multiplier Adjustment section of this chapter.

VT = Surviving target value prior to latest allocation

VTDX = Surviving target value including latest allocation

LAMEF = Lagrange multiplier*

PREMIUM = Bonus for allocation (see Closing Factors, above)

The surviving target value VTDX is computed as follows. Let PWK be the probability of warhead kill by the terminal defense (PKTX in Bomber and Missile Defenses, above).

Define SSSP(G,J) = Single shot survival probability of the target from group G on hardness component J

NOWEP(G) = Number of weapons allocated from group G

VTOA(N1,J) = Value of target hardness component J at time of arrival index N1

S(G,J) = Probability that target component J survives attack of NOWEP(G) weapons from group G

NWHD(G) = Number of warheads per weapon in group G

NN = Number of weapon groups

M = Number of hardness components

Set: VTOA(L,J) = VO(J) = original value of component J

VTOA(NN+1,J) = 0

Then: $S(G,J)=(SSSP(G,J)+PWK-PWK*SSSP(G,J))(NWHD(G)*NOWEP(G))$

If the weapons are ordered by increasing time of arrival, then

$$VTDX = \sum_{J=1}^{M} \sum_{L=0}^{NN} [VTOA(L,J) - VTOA(L+1,J)] * \prod_{G=1}^{L} S(G,J)$$

The innermost sum over L, must be carried out in order of weapon time of arrival.

Since the payoff function for a defended target is generally not con-

_____

* For missile groups with a launch interval time (attribute (LCHINT) greater than zero, the basic multiplier is modified as described in the Salvoed Group Multiplier Adjustment section of this chapter.

cave, one cannot look at only the rate of return of the next missile to determine whether the target is to be attacked. Rather, it is necessary to allocate weapons beyond the exhaustion point and then search for that allocation which yields the highest average rate of return. If this average rate is greater than one (i.e., a profit is realized by attacking the defended target), then the allocation can actually proceed.

The missile allocation proceeds as follows. First, those missiles with the cheapest terminal objects (warheads and terminal decoys) are allocated until the terminal interceptors are exhausted. Then, each missile type in turn is tried to determine which type has the greatest payoff per unit cost when added to this exhaustion mix of weapons.

If it is determined that saturating the terminal defense does not yield a profit, the leakage allocation is restored. In any event, the more profitable allocation, leakage or saturation, is used.

Other Constraints: Several other constraints may be imposed on the weapon allocation. These constraints will reduce the payoff but allow more realistic modeling of special cases. Weapon groups may be restricted in the set of targets they are allowed to strike in the follow-

Flag Restrictions: The user may restrict the allocation of weapons from any group according to the attribute FLAG. Weapon groups may be permitted or forbidden to strike targets according to the FLAG value for the targets.

Country Location: The user may specify at program execution time the acceptable target country location codes (CNTRYLOC) for weapon allocation by weapon group.

MIRV Restriction: The user may specify at program execution time the acceptable target classes (CLASS) for allocation of MIRV weapons. These constraints are input by MIRV system type name.

Naval Restriction: While naval forces can appear as targets within QUICK, there are specific limitations on the kind of weapons that can attack the aircraft carriers. All the targets which are included under class NAVAL should be moving ships. Certain weapon types can then be designated to attack only NAVAL targets. Since the mechanism of interaction of these naval strategic weapons with the aircraft carriers is essentially different from the normal kill mechanisms used in QUICK, an attribute (PKNAV) is defined for this type of weapon which specifies its single shot kill probability against an aircraft carrier. Thus, in the allocation process if a particular target is class NAVAL, the only weapons which can be allocated against that target are those which have the attribute PKNAV defined to be greater than zero. The kill probability of such a weapon, if successfully delivered through the area defenses against the carrier, is equal to PKNAV. These naval attack aircraft are handled like the tactical aircraft, since they do not pass through penetration corridors.

462

User-Specified Damage Levels (MINKILL/MAXKILL): The QUICK Plan Genera-
tor allows the user to specify the maximum (MAXKILL) and/or minimum
(MINKILL) desired level of damage for any particular target. MINKILL
specifies the minimum level of damage the allocator is to attain (if
not attainable, the user is informed by the message MINKILL Too High).
MAXKILL precludes the assignment of additional weapons once the speci-
fied level of damage is attained. Because only an integral number of
weapons can be assigned to a target, the level of damage specified by
MAXKILL may be slightly exceeded, unless there exists a combination of
weapons which exactly meets the required damage level.

This slightly greater level of damage is intensified when the damage is
evaluated using procedures which ignore the interweapon correlations
and planning factor modifications used in QUICK. In order that the
user can specify whether or not the application of damage constraints
considers these factors, two options are available to the user for
implementing these constraints. As a default option, these constraints
are applied to damage calculations which include degradations for corre-
lations in weapon delivery probabilities and considerations of the time
dependence of target value. Since the evaluation programs to be used
in conjunction with QUICK did not take these factors into account and
since the output of these programs was to be compared to the QUICK-gen-
erated analysis, an optional computational procedure was desirable.
Thus, the user has the option of specifying that the variables MAXKILL
and MINKILL be applied to target damage which was calculated by ignoring
the correlations and weapon delivery probabilities and the time degrada-
tion of value of the target. (User-input parameter IMATCH is used for
this purpose.)

Combined Fixed, Optimum Assignment Capability: In order to provide for
more precise user control of weapon allocations, there is a capability
in the plan generation process to allow the user to specify certain
particular weapon-to-target assignments and then allow the automated
plan generation process to allocate the residual of the weapon stock-
pile so as to maximize destruction of the remaining target value. The
user can specify at his option certain fixed weapon assignments at a
point prior to the actual weapon-to-target automatic (mathematical)
allocation process. This allows the user to examine the output of all
of the preceding modules before committing himself to a particular
fixed assignment. The user must specify the target identifier (target
designator) of each target for which weapons are going to be forced-
assigned. Also, the group of the weapon or weapons which is to be
assigned to each of those targets, as well as the number from those
groups, must be input.

This particular capability is made possible by the flexibility of the
generalized Lagrange multiplier technique for performing optimum weapon
allocations. Since any constraints can be imposed on the allocation to
an individual target without seriously affecting the Lagrange multiplier
allocation procedure, it is necessary only to modify the damage calcula-
tions for each target to reflect the damage created by the user-specified

463

weapons prior to calculating the return for new potential weapons additions. Thus, when the allocator initiates the first pass, the only target value that has to be considered is that which is unaffected by the fixed assigned weapon. Also, the assigned weapons are subtracted from the stockpile available for automatic assignment.

In addition to the fixed assignment capability, the user may also specify the precise impact time of a fixed missile assignment. This allows the user to externally plan a time saturation attack against a BMD installation and be assured that the final QUICK plan will execute the tactic. The only use for this impact time specification is to calculate the correct missile launch time. If an impact time is fixed, this calculation overrides the other factors which would normally determine weapon launch time. However, the use of attribute DELTA for a missile base will modify the launch time in the Simulation subsystem; and the user-input parameters DELMIS or DLMIS (in module PLANOUT) will modify the launch time used in other simulators and damage-assessment systems.

If the target does not have terminal ballistic missile defenses, a maximum of 30 weapons can be assigned. On targets with terminal BMD, weapons from a total of 30 weapon groups may be assigned with no limit on the maximum number of weapons. In this latter case no bomber weapons may be fixed assigned if more than 30 missiles have been fixed assigned.

For missiles with a MIRV capability the assignment and timing of a fixed assignment may be changed by the application of the MIRV footprint parameter constraints.

Selection of Bomber Weapon Allocation: Within the weapon allocation process, the gravity bombs and ASMs of a bomber are treated somewhat differently. The penetration probability of an ASM does not include local attrition effects. The kill probability of the ASM uses the ASM CEP factor (rather than the bomber's factor) and applies the ASM reliability REL. The yield for a gravity bomb used in allocation is the group basic yield. The yield for an ASM is the yield of the ASM warhead.

The allocation process selects the kind of weapon (ASM or gravity bomb) for each target by considering the damage difference between the weapons, the difference between the allocated and actual ratios of ASMs to bombs, and the state of the allocation (i.e., PROGRESS. See the Adjustment of Multipliers section of this chapter.).

The weapon selection method uses a bomber group balance variable which is dynamically maintained for each bomber group. This variable reflects the degree of over or underallocation of ASMs (or bombs) that is currently experienced during the convergence process. The basic group multipliers are not affected in any manner, and determine whether any weapon from that group is to be allocated. This balance variable is updated after the allocation to each target. The calculation and use of this variable involves several variables which are defined as follows:

464

EXPASM = fraction of weapons in a group which are ASMs

EXPBMB = 1 -EXPASM = fraction of weapons which are bombs

DEA   = expected destroyed value of target if ASM used (calculated in program ALOC)

DEB   = expected destroyed value of target if bomb used (calculated in program ALOC)

AVDE   = average (by group) of quantity ABSF (DEA -DEB)

FASM   = current fraction of weapons allocated which are ASMs (calculated in program ALOC)

FBOMB   = 1 -FASM = current fraction of weapons allocated which are bombs

CONPAY = internal program variable between 0. and 1.

Except for CONPAY, all of the above variables are defined for each group composed of bombers. For each bomber group on each target the allocation process selects the type of warhead (ASM or bomb) which is to be used on the target. When the value of PROGRESS is zero or two, the preferred weapon is the weapon with the higher DE (i.e., ASM will be selected if DEA is greater than DEB and vice versa). For values of PROGRESS of .4, .5, .75, and 1.0, the selection process will consider the allocation franctions of the ASMs and bombs, as described in the following paragraphs.

If ASMs are underallocated, ASMs are selected as the preferred weapon unless DEB is greater than DEA and the quantity (EXPASM -FASM)/EXPASM is less than or equal to the quantity CONPAY*(DEB-DEA)/AVDE. If both of these two conditions are met, then the preferred weapon is the bomb.

Note that the quantity (EXPASM -FASM)/EXPASM provides a measure of the size of the allocation imbalance. If ASMs are only slightly underallocated, this quantity will be very small (near zero). If there is a great difference between the actual and allocated fractions, this quantity will approach the value one. The quantity (DEB-DEA)/AVDE is a measure of the magnitude of the damage difference relative to the average damage difference. The quantity ranges from a low of zero to high positive values. A value of one for the quantity represents an average damage difference. The variable CONPAY is used to reflect the importance of the allocation difference relative to the damage difference. Thus, the conditions of the preceding paragraph select the bomb as the preferred weapon if the allocation difference is less than the modified damage difference. Thus, if the allocation is nearly correct, the more damaging weapon is likely to be chosen as preferred. If the allocation is far from correct, the underallocated weapon will be selected on all targets except those where the damage difference is quite large. The

465

same rationale holds for the case of underallocated bombs as described in the next paragraph.

If bombs are underallocated, bombs are selected as the preferred weapon unless DEA is greater than DEB and the quantity (EXPBMB -FBOMB)/EXPBMB is less than or equal to the quantity CONPAY*(DEA -DEB)/AVDE. If both these conditions are met, then the preferred weapon is the ASM.

The value of variable CONPAY lies between zero and one. Lower values of CONPAY tend to increase the importance of the allocation difference. High values of CONPAY increase the importance of the damage difference. In order to provide adequate closing force, the value of CONPAY decreases as the value of PROGRESS increases. Additionally, when PROGRESS equals one, the value of CONPAY continues to decrease.

The selection of ASM or bomb on a particular target allows the allocation process to assess correctly the expected damage effects. Bombs and ASMs usually differ greatly in yield, penetration probability, CEP, and delivery probability. By differentiating between these weapon types at allocation time, the allocation program selects the best weapon to be used when the bomber sorties are generated. The balance between allocation and damage differences provides for maximization of damage while continuing consideration of actual weapon stockpiles.

A.5 Derivation of Lagrange Multiplier Adjustment

Define the following variables:

CURSUM(J)   = sum of the target weights from last Lagrange multiplier update

NOWPS(J)   = number of weapons sharing attribute J

NTGTS   = number of targets

SNSTVTY ⎫
FSNSTVTY ⎭   = user-input parameters which control rate of multiplier adjustment

LAMEF(G)   = Lagrange multiplier for group G

LA(J)   = Lagrange multiplier for attribute J; J = ALL, CLASS, TYPE, etc.

PRM   = local internal control variable which governs size of premiums (closing factors)

NWPNS(G)   = number of weapons in group G

CTMULT   = current target multiplicity

RUNSUM(J)	= product of target weights time number of weapons assigned
WTSUM(J)	= sum of target weights measured from beginning of target list
ALERREST(J)	= error estimate in the allocation rate for attribute J
CORRATE	= rate to correct the weapon allocation rate
MULSTEP	= number of targets processed between corrections
WRATE	= rate of change of the target weights

General Approach: Multipliers are not continuously updated, but rather recomputed based on the internal variable PROGRESS (described later) and various estimates of error allocation rates.

If PROGRESS = 1.0 the change of the local multiplier is omitted so that the same values of the multipliers are retained. If PROGRESS < .75 updates are performed for every two targets and every four targets for PROGRESS = .75. In addition to PROGRESS restrictions each multiplier is changed only if all three estimates of error rate have the same sign. In the early phases of the program (PROGRESS < .75) better stability is achieved by requiring, in addition, that the average allocation rate to the last two to four targets, as computed from CURSUM, show the same sign. This limitation is later removed, since it clearly would not work well for weapon groups with very small numbers of weapons that might only be allocated two to ten times during a pass over the target system.

Upon meeting the mentioned restriction, multipliers are updated. The first step is to recompute all the allocation error estimates, ALERREST. At the same time SURPWP is reevaluated, based on the new value of ALERREST. Although SURPWP is continuously updated by the operating program, it is useful -- especially in the early phases of the program -- to base it on the projected allocation-rate estimates rather than the actual weapons allocated, which at that time could be very misleading. This provides a more rational basis for calculating the premiums at this early stage of the program. An estimate is then made of CORRATE, the rate at which it is desired to correct the allocation rate. Lambda multipliers are now recomputed based on ALERREST, SURPWP, CORRATE, and inputs parameters SNSTVTY, FSNSTVTY. Also, at this time the value of all weapon, VLWPNS, and the summation of the value of the error in weapons allocated, VALERR, are reevaluated along with a recalculation of the integration periods used to estimate allocation rates. After these updates are made, allocation continues.

The following paragraphs show mathematical deviations of methods employed.

467

Adjustment Phase: The adjustment phase processing is determined in part by an internal variable, PROGRESS. This variable is assigned the arbitrary values 0., .4, .5, .75, 1., and 2. by the program as a flag for various stages of the allocation process. PROGRESS is initially set to 0. at the start of processing by program ALOC. When the sum of target weights, WTSUM, exceeds half the number of targets PROGRESS is set to 0.4. When the weight change rate (WRATE, described later in this section) first decreases, PROGRESS is set to 0.5. When the weight change rate decreases to zero value, PROGRESS is set to 0.75. A user-input parameter, SETTLE, determines the next change. SETTLE is the number of passes the process continues with PROGRESS equal to .75. After this time PROGRESS is set to 1.0. PROGRESS remains at this value until one of three conditions is met:

1. More than 1.5 passes over the target set are made while PROGRESS = 1.0;

2. The sum of the Lagrange multipliers for the under- or over-allocated weapons (VALERR) is less than a fraction (ERRCLOS, a user input parameter) of the sum of the Lagrange multipliers for all the weapons in the stockpile (VALWPNS);

3. The sum of the squares of the allocation error estimates (SUMSQUERR, the sum of the squares of ALERREST, described later in this section) is less than $1/(10 * NTGTS^2)$, where NTGTS is the number of targets.

When any of these three conditions is met, the allocation process is complete and PROGRESS is set to 2.0.

Multipliers, when adjusted, are recomputed based on the monitoring of the allocation rates; CORRATE being the allocation rate correction. If the allocation rate is corrected too rapidly there will be a tendency to overcorrect before the effects of the correction become observable in the values of the allocation error estimates. This can produce oscillations. To estimate how rapidly to correct the error, an estimate is made of the number of targets that would have to be observed before an error of the observed size would be statistically significant. Even if the multipliers were exact, and the average allocation rate was correct, statistical fluctuations would be observed in the allocation of each weapon group when the allocation rate was sampled for a small number of targets.

The concept for monitoring allocation rates and, hence, updating multipliers follows.

Let n equal the expected or average number of weapons from a collection available per target; i.e., n = NOWPS(J)/NTGTS. Then in M targets (the size of M is discussed later) the expected number of weapons allocated should be just n(M). Suppose the actual number observed, however, is

468

n'(M). Then our estimate of the error in the allocation rate ALERREST would be

$$ALERREST = n'-n$$

Assuming a Poisson distribution for weapon allocation rates, the statistically expected error in a number of expected value n(M) is equal to $\sqrt{n(M)}$. That is,

$$EXPECTED\ ERROR = \sqrt{n(M)}$$

or, substituting for ALERREST,

$$(n'(M) - n(M))^2 = n(M)$$

$$(n' - n)^2 = n/M$$

Solving for the number of targets M, we have:

$$M = n/[(n'-n)^2]$$

or

$$M = (NOWPS(J)/NTGTS)/[ALERREST(J)]^2$$

as the number of targets we would expect to sample to get a statistical error estimate of size, ALERREST. If we wish to reduce the indicated error by 1 part in M per target, our fractional correction in the allocation rate per target should be:

$$1/M = [ALERREST(J)]^2/(NOWPS(J)/NTGTS)$$

This, multiplied by a sensitivity factor SNSTVTY, is the first term in the value of CORRATE. Therefore, the user-controlled factor SNSTVTY can make the correction more or less sensitive to 'recent' target experience. If SNSTVTY is too high (much above .1) oscillations are more likely to occur. However, if the entire set of targets were observed, the estimate would not be a sample but would be exact. Therefore, even a very small value of ALERREST becomes statistically significant if it is based on a sample of size NTGTS. Therefore, errors should always be corrected at a rate at least equal to one part in NTGTS.

This explains the second term in CORRATE, which is just 1.0/NTGTS multiplied by a sensitivity factor FSNSTVTY (final sensitivity). This factor controls the sensitivity of corrections to the allocation rate in the final phase of the allocation where the errors are small. Thus the desired correction rate is just:

$$CORRATE = [(SNSTVTY)*(ALERREST(J))^2]/(NOWPS(J)/NTGTS) + FSNSTVTY/NTGTS$$

This is multiplied by the number of targets processed between corrections,

MULSTEP, to determine the fraction CORFAC of the error to correct. In addition, a safety limit of $\frac{1}{2}$ is used to avoid ever making a correction larger than $\frac{1}{2}$ the estimated e-ror rate.

However, even when it is known what fraction of the error in the allocation rate we wish to correct, an estimate must be made of the relationship of the allocation rate to changes in the Lagrange multipliers before the size change to make in the multiplier can be estimated. For this purpose it is useful to have a model of the dependence of the allocation rate on the value of the multipliers. We have assumed a dependence as follows:

$$\text{Rate} = k \ \lambda^{-n}$$

Consider now two rates, the current rate R_0 associated with a multiplier λ_0 and a predicted rate R_1 associated with a new multiplier λ_1. Thus we find

$$R_1 \lambda_1^{\ n} = R_0 \lambda_0^{\ n} = k$$

or

$$R_1/R_0 = (\lambda_1/\lambda_0)^{-n}$$

so

$$\frac{\partial(R_1/R_0)}{\partial(\lambda_1/\lambda_0)} = -n$$

For small differences between λ_0 and λ_1 this implies:

$$\frac{R_1 - R_0}{R_0} = -n \ \frac{\lambda_1 - \lambda_0}{\lambda_0}$$

Solving for the new value λ_1 of

$$\lambda_1 = \lambda_0 (1 + \frac{(R_1 - R_0)/(-n)}{R_0}$$

If we now identify a new variable R2 as the ultimately desired allocation rate, R_1 as the new rate we hope to obtain with λ_1, and R_0 as the current allocation rate -- then the above variables can be associated with information already available as follows:

$$R_1 - R_0 = CORFAC*(R_2 - R_0) = CORFACT*ALERREST$$

$$R_0 = ALERREST + (NOWPS/NTGTS)$$

If we now associate the variable PARTIAL with n this gives rise to the following procedure for updating LA:

$$LA_1(J) = LA_0(J) * [1.0 + \frac{CORFAC*ALERREST(J, \ INTPRD)/(- \ PARTIAL)}{ALERREST(J, \ INTPRD) + (NOWPS(J)/NTGTS)}]$$

ALERREST(J) is computed as

$$ALERREST(J) = \frac{RUNSUM(J)}{WTSUM(J)} - \frac{NOWPS(J)}{NTGTS}$$

The formula for $LA_1(J)$ is well-behaved if ALERREST is large and positive but if it is negative and as large as the expected rate (NOWPS(J)/NTGTS) (i.e., if the actual allocation rate is zero), then the denominator goes to zero. In this case an infinite correction would be indicated. To avoid this, the expected rate in the denominator is multiplied by 2 giving:

$$LA_1(J) = LA_0(J) * [1.0 + \frac{CORFAC*ALERREST(J, \ INTPRD)/(-PARTIAL)}{ALERREST(J, \ INTPRD) + 2*(NOWPS(J)/NTGTS)}]$$

This is the function used. The new Lambda's $LA_1(J)$, are recomputed for attribute J (e.g., Jall, Jclass) and for every MULSTEP targets as previously outlined.

In the present version of the program the value of PARTIAL(J) has been set equal to 1.0 for all the local multipliers LA(J). This choice is based on the effect of the return on the sensitivity of the allocation rate to the value of LAMEF or λ. When the multipliers are almost correct, it is usually the case that most weapon groups are in close competition with many other groups with very similar properties. Then a small change in the multiplier LAMEF will produce a very large change in the allocation rates, as the weapon group in question almost totally replaces, or is replaced by, its competitors.

However, such a large error in the allocation rate will not actually occur because as the error builds up the estimated value of the payoff will be automatically changed by the premium. Thus for constant values of LAMEF, when an equilibrium allocation rate is reached, it must be approximately true that the error in LAMEF is compensated by the premium. This is, if λ_0 is the correct value for LAMEF then:

$$LAMEF - PREMIUM \cong \lambda_0$$

Since:

$$PREMIUM = PRM*LAMEF* \frac{SURPWP - .5*CTMULT}{NWPNS}$$

we can define a relation between LAMEF and (SURPWP/NWPNS)

$$\text{LAMEF}*(1 - \text{PRM}* \frac{\text{SURPWP} - .5*\text{CTMULT}}{\text{NWPNS}}) \cong \lambda_0$$

Since this relationship is the same for all groups it is reasonable simply to use the same value 1.0 of partial derivative for all local multipliers.

The values of LAMEF(G), where G is the group index, are recomputed using the new values of the local multipliers (LA(J) accordingly,

$$\text{LAMEF}(G) = \text{LA}(\text{Jall})*\text{LA}(\text{Jclass})*\text{LA}(\text{Jreg})*\text{LA}(\text{Jalert})*\text{LA}(\text{Jgroup})$$

At the same time it is necessary to reevaluate the summation of the value of all the weapons VALWPNS = Σ LAMEF(G)*NWPNS(G) and the summation of the value of the error in weapons allocated

$$\text{VALERR} = \Sigma \text{ LAMEF}(G)*\text{ABSF}(\text{SURPWP}(G))$$

using the updated values of LAMEF.

Target Weight Change Rate & Integration Period: The above explained how multipliers are recomputed by monitoring allocation rates. The remaining discussion addresses how the target weight change rate and integration period is computed.

The average number of targets over which allocation rates are averaged (the integration period) is determined by the rate at which the target weights are increased.

In estimating the rate with which to correct multipliers, it was computed on a statistical basis that even if the allocation rates were correct an estimated error of size ALERREST would be expected if the allocation rates were monitored only over a small sample of M targets where:

$$M = (\text{NOWPS}(J)/\text{NTGTS})/(\text{ALERREST}(J))^2$$

Thus if separate integration periods could be used for each local multiplier, M as defined above might provide a reasonable basis for determining the period. However, in fact, the same periods must be used for all local multipliers LA(J). Currently three periods are maintained (INTPRD=1, 2, 3). Consequently the value of the integration period used must be based on an estimate of overall error rate. The corresponding relation is:

$$M = (\underset{G}{\Sigma} \text{ NOWPS}(J)/\text{NTGTS})/\underset{G}{\Sigma} (\text{ALERREST}(J))^2$$

where the summations are taken over all weapon groups. The quantity $\underset{G}{\Sigma}$NOWPS(J), is identical with NOWPS(1) (Note: LA(J) for J = 1 is used

for all weapon groups) and so for efficiency the variable NOWPS(2) is used. While the expected value of $(ALERREST(1))^2$ is the same as $\sum_G (ALERREST(J))^2$, the variance of the latter version is much less, and it is therefore preferable as an estimator of the expected integration period, EXPINTPD and is:

$$EXPINTPD = NOWPS(1)(ALERREST(1)^2)*NTGTS)$$

To allow the possibility of using integration periods either longer or shorter than the theoretical EXPINTPD, a desired longest integration period DESINTPD is defined:

$$DESINTPD = EXPINTPD*RATIOINT$$

where RATIOINT is an adjustable input parameter. A low value allows higher sensitivity without oscillations in the values of the Lagrange multipliers but too low a value makes convergence to the correct stockpile sensitive to statistics of the target list. If the target list contains targets with heavy ballistic missile defenses or if a large fraction of the weapons are assigned by the fixed assignment capability, this parameter value should be increased (to 4.0 or above if necessary). If this period were used exactly in setting the rate of change of the target weight (i.e., WRATE = 1.0/DESINTPD), the WRATE would never become exactly zero as is required for a constant target weight. Obviously when the change in the target weight becomes small over a full pass, the WRATE should be allowed to go to zero. Therefore in:

$$WRATE = (1.0/DESINTPD) -(2.0/(NTGTS*RATIOINT))$$

the term (2./NTGTS*RATIOINT)) is subtracted, and if the resulting WRATE is negative it is set to zero. To avoid a situation where large errors cause the integration period to become ridiculously small, a limit that WRATE $\leq$ .07 is set.

Moreover, after the allocation is well under way, PROGRESS $\geq$ .5, the value of WRATE is not allowed to increase. In the program WTRATE(INTPRD) is used as a multiplier of the target weight; therefore we add 1.0 to WTRATE to obtain a suitable multiplier for the longest period NINTPRD.

The values for the three WTRATE variables are:

$$WTRATE(3) = 1 + WRATE$$

$$WTRATE(2) = 1 + WRATE + \frac{RINTPRD-1}{NTGTS} + RINTPRD$$

$$WTRATE(1) = 1 + WRATE + \frac{RINTPRD-1}{NTGTS} + 2.*RINTPRD$$

Input parameter RINTPRD is an approximate ratio between rate of change of target weights between different integration periods. An increase in

this parameter increases the sensitivity of the multiplier adjustment to recent target experience.

To restate, Lagrange multipliers are recomputed based on variable PRO-GRESS and after a specific number of targets have been processed. The adjustment is based on maintaining statistics of weapon allocation rates. The differences in true and observed rates, along with input sensitivity parameters, make up the formula for multiplier adjustment.

A.6 Derivation of Formula for Correlations in Weapon Delivery Probability

An exact calculation of the probability of target survival when it is subject to attack by correlated weapons is very lengthy. Both the conventional statistical analysis and the bayesian incremental information approach have been examined. Both approaches for each time and hardness require the calculation component of the interaction terms between each weapon to be added with all possible combinations of the weapons already on the target. Thus the completely rigorous calculation would be impractical in a rapid response allocator. The method used here is based on an approximation derived from the properties of the log-gamma distribution.

When a group of weapons share a common failure risk the probability of success is likely to be either high or low for all weapons collectively. Thus the probability of success can itself be thought of as a random variable. For any chance value of this overall random variable there will exist the usual independent probabilities for individual weapons. However, on one trial the overall success probability for the group of weapons may be 90%, while in another trial it may be 50% depending on the particular success probability drawn for the trial.

The following mathematical model has been developed to deal with this type of problem. We assume that the _probability_ of survival of a target with respect to the ith weapon is itself a random variable S of the form

$$S_i = e^{-X_i}$$

where the X_i are random variables drawn from a known distribution.

If two weapons are involved, then the probability of survival with respect to both can be represented by the random variable S_T:

$$S_T = S_i S_j = e^{-(X_i + X_j)}$$

However, the random variables X_i and X_j may or may not be independent. If they are not independent then of course

474

$$<S_i S_j> \quad <S_i><S_j>$$

If the X_i are independently drawn from a known two-parameter family of distribution with a convolution property,[*] then the distribution of $X_i + X_j$ will of course be a member of the same distribution family. Moreover, since any probability distribution for the X_i implies a distribution for the corresponding S_i, the distribution for $S_i S_j$ can be calculated and the value for $<S_i S_j>$ can be computed.

The gamma distribution given by:

$$P(X)dx = \frac{X^a e^{-X/b}}{b^{a+1}\Gamma(a+1)} \ dx \text{ for } X \geq 0$$

$$P(X) \quad = 0 \text{ for } X \leq 0$$

is a well known two-parameter distribution with the required convolution property.

The gamma distribution is unique among convolving two-parameter distributions in that the expected value of e^{-X} is easily computed. This property is particularly important for QUICK since the damage function performs a computation of this value many times during the allocation. The expected value of e^{-X} is given by:

$$<e^{-X}> = \int_0^\infty P(X)e^{-X}dX$$

which can be written

$$<S> = <e^{-X}> = \left(\frac{1}{b+1}\right)^{a+1}$$

This distribution is valid for $b > 0$ and $a > -1$. It has a mean $\mu = b(a+1)$ and a variance $\sigma^2 = b^2(a+1)$.

Since this distribution is completely defined by the mean and variance, the actual probability distribution of S can be computed at any time so long as a record of the mean and variance of the distribution is maintained. We now observe that:

$$a + 1 = \mu^2/\sigma^2$$

---

[*] A probability distribution is said to "convolve" when the convolution of any two distributions in the family (i.e., the distribution of the sum of the two random variables) is itself a member of the same family.

and

$$b = \sigma^2/\mu$$

so the expected value of S can be written

$$\langle S \rangle = \left(\frac{1}{\frac{\sigma^2}{\mu} + 1} \right)^{\mu^2/\sigma^2}$$

or

$$-\ln \langle S \rangle = \frac{\mu^2}{\sigma^2} \ln \left(\frac{\sigma^2}{\mu} + 1 \right)$$

This distribution is sufficiently flexible to include almost any shape distribution of interest. For σ small the distribution in S approximates a gaussian centering on some specific survival probability. As the σ is increased the distribution widens, so that it can approximate a uniform probability from zero to one, or a sloping probability with more weight on zero or one. In the limit of very large σ the distribution consists essentially of spikes of different weight at zero and one.

If we were dealing with $\underline{independent}$ weapons we could calculate the parameters for the multiple weapon distribution from those for the single weapon distributions simply by making use of the additivity of the mean and the variance. Specifically the mean, μ_T, for the new distribution and the variance σ_T^2 would be given by:

$$\mu_T = \sum_i \mu_i$$

$$\sigma_T^2 = \sum_i \sigma_i^2$$

The expected value of target survivability S_T for the new distribution would then be obtainable through the equation:

$$- \ln \langle S_T \rangle = \frac{\mu_T^2}{\sigma_T^2} \ln \left[\left(\frac{\sigma_T^2}{\mu_T} \right) + 1 \right]$$

However, the variance is directly additive as above only if the weapons are really independent. To introduce the possibility of correlations we will write the variance as follows:

$$\sigma_T^2 = \sum_i \sum_j \sigma_i \Gamma_{ij} \sigma_j$$

476

where the quantity Γ_{ij} represents the correlation between the weapons. In the special case of uncorrelated weapons, $\Gamma_{ij} = 0$ for $i \neq j$ and 1 for $i = j$, which is identical with the previous form.

This approach of arbitrarily introducing the cross terms in this formulation to approximate the actual correlations is exact so long as the correlations are of such a form that the distribution of X remains a gamma distribution. To the extent that the actual correlations cause departures from the Γ distribution the approximation is in error. The correlation model thus amounts to the assumption that correlations can be adequately modeled without going outside the log-gamma distribution.

For implementation it seems appropriate to introduce an additional simplification. In the foregoing formuation the magnitude of the penalty for using correlated weapons will depend not only on the size of the correlation and the kill probability for the correlated weapons, but also on the shape of the distribution for the success probability for each weapon. This shape dependence introduces a complicating variable which undoubtedly exists, but for which it would not be easy to get data. It therefore seems desirable to eliminate this factor.

This can be done by standardizing on a single shape factor for all calculations of the effects of correlations. It is easiest to do this by considering only distributions with a very large σ, which are essentially spikes on zero and one. This choice tends to exaggerate the importance of correlations (and this fact should be borne in mind in assigning the correlations for the war game) but it significantly simplifies the data required, as well as the computation of the payoff.

In the limit of large σ the quantity σ_i^2/μ_i approaches infinity while the quantity μ_i^2/σ_i^2 compensates to maintain the correct value of $-\ln \langle S_i \rangle$

To illustrate the transition to this limit we let $b_i = \sigma_i^2/\mu_i$ and define

$$\beta_i = b_i/\ln(b_i + 1)$$

Then

$$-\ln \langle S_i \rangle = \mu_i/\beta_i$$

so:

$$\mu_i = \beta_i \left[-\ln \langle S_i \rangle \right]$$

and

$$\sigma_i^2 = \mu_i b_i = b_i \beta_i \left[-\ln \langle S_i \rangle \right]$$

477

The formula for obtaining the expected value of S_T can now be written

$$-\ln \langle S_T \rangle = \frac{\mu_T^2}{\sigma_T^2} \ln(b_T + 1)$$

and substituting,

$$\mu_T = \Sigma \mu_i \text{ and } \sigma_T^2 = \sum_{ij} \sigma_i \Gamma_{ij} \sigma_j$$

we obtain:

$$-\ln \langle S_T \rangle = \frac{\left(\Sigma \beta_i \left[-\ln \langle S_i \rangle \right] \right)^2 \left(\ln(b_T + 1) \right)}{\sum_i \sum_j b_i \beta_i \left[-\ln \langle S_i \rangle \right]^{1/2} \Gamma_{ij} \left[-\ln \langle S_j \rangle \right]^{1/2}}$$

We now assign to all weapons the same value of b_i, so that all b_i are equal and all β_i are equal and we obtain:

$$-\ln \langle S_T \rangle = \frac{\ln(b_T + 1)}{\ln(b_i + 1)} \frac{\left[\Sigma (-\ln \langle S_i \rangle) \right]^2}{\sum \sum (-\ln \langle S_i \rangle)^{1/2} \Gamma_{ij} (-\ln \langle S_j \rangle)^{1/2}}$$

If we now let b_i approach infinity the ratio of the two logarithmic quantities will approach 1. Note that

$$b_T = \frac{\sigma_T^2}{\mu_T}, \qquad \text{so } b_T = \frac{\sum \sum \sigma_i \Gamma_{ij} \sigma_j}{\Sigma \mu_i}$$

It follows that $b_T \geq b_i$ and $b_T \leq \eta^2 b_i$, where η is the number of weapons. The limiting case $b_T = \eta^2 b_i$ occurs when all $\Gamma_{ij} = 1$ and all μ_i are equal. Therefore so long as $b_i \gg \eta^2$ the ratio of the logarithms will be essentially 1, and in the limit as b_i approaches infinity we obtain simply:

$$-\ln \langle S_T \rangle = \frac{\left[\Sigma \Sigma -\ln \langle S_i \rangle \right]^2}{\sum \sum \left(-\ln \langle S_i \rangle \right)^{1/2} \Gamma_{ij} \left(-\ln \langle S_j \rangle \right)^{1/2}}$$

For compactness of notation let us identify the quantities

$$\mu_i = (-\ell n \langle S_i \rangle) \quad \text{and} \quad \mu_T = (-\ell n \langle S_T \rangle)$$

Then since $\delta_{ij} = 1$ if $i = j$ we obtain

$$\mu_T = \frac{\left[\sum_i \mu_i \right]^2}{\sum_i \mu_i + \sum_i \sum_{j=i} \left(\mu_i \right)^{1/2} \Gamma_{ij} \left(\mu_j \right)^{1/2}}$$

or equivalently

$$\mu_T = \frac{\left[\sum_i \mu_i \right]^2}{\sum_i \mu_i + \sum_i \sum_{j<i} \left(\mu_i \right)^{1/2} 2 \Gamma_{ij} \left(\mu_j \right)^{1/2}}$$

This form has the basic properties desired. Notice there is only one interaction term between each pair of weapons. In addition, only two sums need to be maintained to compute μ_T. These are:

$$MU = \sum_i \mu_i$$

$$SIG = \sum_i \sum_{j<i} \left(\mu_i \right)^{1/2} 2 \Gamma_{ij} \left(\mu_j \right)^{1/2}$$

From these the value μ_T is given simply:

$$\mu_T = (MU)^2 / (MU + SIG)$$

The addition of any new weapon adds one term to the MU sum, and several terms to the SIG sum.

The computation of the first sum is trivial; however, before the second one can be used it is necessary to provide a practical method of estimating Γ_{ij}.

We recall that the array RISK (A,G,J) was computed as an estimate of shared risk, and that:

$$RISK(A,G,J) = \sum_{L=1,5} SM(L) * SMAT(A,L)$$

For a particular weapon G and hardness component J, this relation might look as follows: (A is a weapon attribute index; L is a failure mode index.)

		SMAT(A,L)						
L	SM(L)	A = 1 All	2 Group	3 Reg	4 Class	5 Type	6 Alert	Independent Risk
1	-LOGF(DBL) = .20	.00	.10	.10	.10	.10	.40	.20
2	-LOGF(CC) = .00	.00	.10	.30	.10	.10	.30	.10
3	-LOGF(REL) = .05	.00	.05	.00	.10	.20	.00	.65
4	-LOGF(PEX) = .20	.00	.00	.10	.20	.20	.00	.50
5	-LOGF(STK) = .02	.00	.00	.00	.00	.00	.00	1.00
	RISK(A,G,J)	.000	.0225	.040	.065	.070	.08	.1925

Thus the SMAT array, a user input estimate of shared risk, is used simply to divide the five types of risk SM(L) between the independent weapon risk, and the six factors A that any two weapons might have in common. The total RISK over all A plus the independent risk is of course equal to the sum of SM(L). We are now interested in using the RISK array to derive reasonable values for the correlation coefficients Γ_{ij}.

The RISK array thus represents the amount of the risk for each weapon that is likely to be correlated with other weapons of the same class, type, etc.

The correlation coefficients should reflect the _shared_ risk. If two weapons have only two attributes A in common then the shared risk should come only from these two common attributes. Moreover, the amount of risk that can be shared on the basis of one attribute cannot exceed the minimum risk associated with that attribute for either weapon. Therefore, to estimate the maximum risk, γ_{ij}, that can be shared by two weapons, i and j, we define:

$$\gamma_{ij} \text{ or GAM}(k,j) = \sum_{A} \delta\ (A_i, A_j) * \text{Min RISK}(A_i, G_i, J) \text{RISK}(A_j, G_j, J)$$

where $\delta = 0$ if $A_i \neq A_j$ and $\delta = 1$ if $A_i = A_j$.

The coefficients Γ_{ij} however must never exceed 1.0. Therefore it is appropriate to divide the shared risk GAM(i,j) by $\sum_L$ SM to obtain a normalized fraction guaranteed to be less than 1.0.

Thus the form of the second summation

$$\text{SIG} = \sum_{i} \sum_{j<i} 2(\mu_i)^{1/2}\ \Gamma_{ij}(\mu_j)^{1/2}$$

would become

$$\text{SIG} = \sum_{i} \sum_{j<i} 2(\mu_i)^{1/2}\ \frac{\text{GAM}(i,j)}{\sum_L \text{SM}}(\mu_j)^{1/2}$$

However, this form involves square roots which are inconvenient. Moreover, it represents an upper limit of correlation. We can reduce the size of the overestimate by using the largest (or maximum) $\sum_L SM$; i.e., using the least reliable weapon for normalization. In addition, we can simplify the form and provide for the removal of square roots if we also multiply by $(\mu_{min}/\mu_{max})^{1/2}$. (This is a factor less than 1.0 that has the effect of reducing slightly the assumed correlation between weapons of very different overall effectiveness.)

With these changes, the equation for SIG takes the form of

$$SIG = \sum_i \sum_{j<i} 2(\mu_i)^{1/2} \left\{ \frac{GAM(i,j)}{\text{Max } \sum_L SM} * \left(\frac{\mu_{min}}{\mu_{max}}\right)^{1/2} \right\} (\mu_j)^{1/2}$$

The form in braces is still guaranteed to fall between zero and 1.0. It represents the actual form for Γ_{ij} used in the present version of the Allocator. This form has a computational advantage in that it simplifies the calculation of SIG. Assume that $\mu_i \leq \mu_j$. Then

$$\sum_L SM_i > \sum_L SM_j \qquad \text{and so}$$

$$SIG = \sum_i \sum_{j<i} 2(\mu_i)^{1/2} \left\{ \frac{GAM(i,j)}{\sum_L SM_i} * \left(\frac{\mu_i}{\mu_j}\right)^{1/2} \right\} (\mu_j)^{1/2}$$

This reduces to:

$$SIG = \sum_i \sum_{j<i} 2*GAM(k,j)*\underset{i,j}{\text{MIN}} \left\{ \frac{\mu_i}{\sum_L SM_i} \right\}$$

This is the actual form used computationally. (For each weapon group G the quantity $\mu/\sum_L SM$ is identified in the FORTRAN as SSIG(G,J).)

The specific formula used for the terms in SIG is of heuristic origin and is obviously somewhat arbitrary. It is justified, in the final analysis, by the fact it is fairly simple and that it works. The resulting kill probabilities produce realistic cross targeting, and in cases where these probabilities can be compared with a rigorous statistical model of correlations, it produces a satisfactory approximation to the kill probability.

481

In summary, the mathematics is as follows:[*]

For a single weapon let

SSK = single shot kill probability, and let

SSS = single shot target survival probability

then SSK is given by

$$-\text{LOGF(SSK)} = \sum_L \text{SM(L)}$$

As usual, SSS = 1.0 - SSK, and we define μ_i or MUP for group G_i relative to hardness component J as:

$$\underline{\text{MUP(G,J)} = -\text{LOGF(SSS)}}$$

We also define SSIG(G,J) as:

$$\underline{\text{SSIG(G,J)} = \text{LOGF(SSS)}/\text{LOGF(SSK)} = \text{MUP(G,J)}/ \sum_L \text{SM(L)}}$$

Finally we define RISK(A,G,J) as:

$$\underline{\text{RISK(A,G,J)} = \sum_{L=1,5} \text{SM(L)}*\text{SMAT(A,L)}}$$

The preceding three arrays (underlined for emphasis) are the main input for the estimation of kill probabilities.

The target survivability relative to multiple weapons S_T is given by

$$S_T = e^{-\mu^T}$$

where $\quad \mu T = (MU)^2 / (MU + SIG)$

and where $\quad MU = \sum_i \mu_i = \sum_i \text{MUP}(G_1, J)$

and $\quad SIG = \sum_i \sum_{j<i} 2(\mu_i)^{1/2} \Gamma_{ij} (\mu_j)^{1/2}$

The individual terms in SIG for specific i and j can be thought of as:

$$\text{DSIG(i,j)} = 2(\mu_i)^{1/2} \Gamma_{i,j} (\mu_j)^{1/2}$$

---

[*] The displayed mathematics for the calculation of MUP are for the exponential damage law. The derivation of the quantity, MUP, required for use of the square root damage law is discussed in the Derivation of Square Root Damage Function section of this chapter and are not of any importance in this discussion of correlation effects.

which we identify computationally as

$$DSIG(k,j) = 2*GAM(i,j)*\underset{k=i,j}{\text{Min}} \left\{ SSIG(G_k,J) \right\}$$

where $GAM(i,j)$, the maximum risk shared by i and j, is estimated as

$$GAM(i,j) = \sum_A \delta(A_i,A_j)*\text{Min} \left\{ RISK(A_i,G_i,J),\ RISK(A_i,G_j,J) \right\}$$

where δ, the Kroniker δ, is 0 if $A_i \neq A_j$, and 1 if $A_i = A_j$.

The simple form used for DSIG above implies that Γ_{ij} has the form:

$$\Gamma_{i,j} = \frac{GAM(i,j)}{\underset{i,j}{\text{Max}}\left[\sum_L SM(L)\right]} * \left(\frac{\mu\text{Min}}{\mu\text{Max}}\right)^{1/2}$$

however, this form never enters explicitly into the calculations.

To combine this treatment for the analysis of weapon correlations with the preceding treatment of time dependent target values we simply use the S_T evaluated above to supply the $S(NI,J)$ required in the formula

$$VT = \sum_{J=1}^{J=M} \sum_{NI=0}^{I=NN} [V(NI,J) - V(NI+1,J)] * S(NI,J)$$

The weapons to be included in the evaluation S_T for any NI are of course those on the target up to and including the time NI.

This, of course, requires that separate sums for MU and SIG be maintained for each relevant time interval, NI, and each hardness component J. Thus these variables are actually two dimensional arrays $MU(NI,J)$ and $SIG(NI,J)$. Moreover, every potential payoff estimate (both for each weapon that might be added, and for each that might be deleted) requires a separate complete set of sums.

Derivation of Damage Functions

A Universal Damage Function: Consider the situation for which the lethal radius and CEP of a single weapon are small compared to the target dimensions. This case becomes quite pertinent under any of the following circumstances:

Very large cities

Targets whose uncertainty of location is larger than the area of influence of a weapon

Employment of large numbers of small weapons (e.g., cluster war-
heads)

Hardening which reduces effective weapon radius below target size
(e.g., blast shelters for urban population).

In such a situation, where the value <u>density</u> of the target does not vary
significantly over the area of effect of a single weapon, one can use-
fully employ the concept of weapon density (weapons targeted per unit
area) and seek the weapon density as a function of value density which
optimizes the total target destruction for a given total number of
weapons.

Before such an optimization can be effected, however, it is necessary to
obtain the relationship between the weapon density applied to a sub-
region, expressed for convenience as the fraction of the original value
surviving. In the most general case, this function can vary with posi-
tion in the target, reflecting the possibility of varying degrees of
vulnerability over the target.

We introduce the following notation:

X	Position within target (x, y coordinates)
$\omega(X)$	Density of weapons targeted in vicinity of X (number/unit area)
$V(X)$	Target value density in vicinity of X (value/unit area)
$F(\omega)$	Fraction of destruction produced by weapon density ω, in the absence of hardening
$\mu(X)$	Vulnerability (hardening) factor ($0 \leq \mu \leq 1$) expressed as effective degradation of weapon density
W	Total number of weapons intended against target.

The total payoff for a given weapon density distribution is then given
by:

$$H = \int_A VF(\mu\omega)\,dA \tag{1}$$

where the integration is understood to be over the whole target area,
and dA is the area element.

Similarly, the total number of planned weapons is given by:

$$W = \int_A \omega\,dA$$

We seek now the weapon density distribution which maximizes the payoff for a given W. Introducing a Lagrange multiplier ≤ 0, and applying the generalized method described above,* we seek the weapon density function which maximizes the unconstrained Lagrangian.

$$L = H - \lambda W \tag{3}$$

This is equivalent to maximizing:

$$L = \int_A [VF(\omega\mu) - \lambda\omega]\,dA \tag{4}$$

The density function ω_λ which maximizes this Lagrangian for a given λ is obtained simply by maximizing the expression inside the integral at each point (see cell problem discussion in Everett's paper, appendix C). The optimum density at any point is therefore a solution of:

$$\underset{\omega}{MAX} = \left\{ VF(\mu\omega) - \lambda\omega \right\} \tag{5}$$

For the case where F is monotone increasing, concave (diminishing returns), and differentiable, an internal maximum of (5) can be sought by zeroing its derivative:

$$\frac{d}{d\omega}\,[VF(\mu\omega) - \lambda\omega] = VF'(\mu\omega_\lambda)\ \mu - \omega = 0 \tag{6}$$

Letting $G = (F')^{-1}$ stand for the inverse function of the derivative of F leads to:

$$\omega_\lambda = \frac{1}{\mu}\,G\ \frac{\lambda}{V\mu} \tag{7}$$

Equation (7) gives the internal maximization of (5). To complete the solution we must account for the constraint $\omega^* \geq 0$ (negative densities are not allowed). Thus the optimum is given by (5) only if $\omega_\lambda^* \geq 0$ and if $VF(\mu\omega) - \lambda\omega \geq 0$, since otherwise (5) is maximized by $\omega = 0$. The complete solution can therefore be stated:

$$\omega_\lambda = \begin{cases} \dfrac{1}{\mu}\,G\,\dfrac{\lambda}{V\mu} & \text{if } \omega_\lambda \geq 0 \text{ and } VF(\mu\omega) - \lambda\omega \geq 0 \\[3em] 0 & \text{otherwise} \end{cases} \tag{8}$$

(This solution is also valid even if F is not concave -- a situation in which G may be multivalued -- provided that one uses that value of $G(\lambda/V\mu)$ for which $VF(\mu\omega) - \lambda\omega$ is a maximum.)

Observe that the optimum density given by (8) is a function only of V and H, and is explicitly independent of position. If we can further assume that the vulnerability H is a function only of the value density V and is otherwise independent of position,* then we can simplify the formulation and solution somewhat. In this case, all pertinent target characteristics are summarized by two functions:

A(V) = total area of those areas whose value density is greater than V

μ(V) = vulnerability factor as a function of value density

The optimum weapon density ω_λ given by (8) becomes then a function only of the value density V:

$$
\omega_\lambda \ (V) = \begin{cases} \dfrac{1}{\mu(V)} \ G\left(\dfrac{\lambda}{V\mu(V)}\right) \text{ if } \omega_\lambda \ \geq 0 \text{ and } VF(\mu\omega) \ - \lambda\omega \geq 0 \\ \\ 0 \qquad \text{otherwise} \end{cases} \tag{9}
$$

and the total payoff and total weapons are given in the simple form of Stieltjes integrals:

$$
H_\lambda \ = - \int_0^\omega VF(\omega_\lambda \ \mu(V))dA(V)
$$

$$
\tag{10}
$$

$$
W_\lambda \ = - \int_0^\omega \omega_\lambda \ dA(V)
$$

This completes the general optimization of weapon density. For explicit solutions we require specific functions for the target value distribution function A(V), the destruction function F(ω), and the vulnerability distribution μ(V). We shall now consider several pertinent cases.

---

*Which seems generally quite plausible, and is in any case certainly true if the variation of H arises from optimization of shelter deployment, for example.

<u>Locally Random Impact Model</u>: When the CEP is not significantly smaller than the lethal radius, or when the delivery probability of individual weapons is low, the situation over any homogeneous part of the target can be closely approximated by regarding the weapons as having been dropped uniformly at random over that part.

Consider, therefore, a region of area A (large compared to the lethal area of a single weapon) into which N weapons each with lethal area πR_K^2 and delivery probability P are delivered uniformly and independently at random. The probability that any given point in the region will survive one weapon is:

$$S(1) = 1 - \frac{P\pi R_K^2}{A} \tag{11}$$

and, since weapon arrivals are independent events, the probability of surviving N is:

$$S(N) = \left(1 - \frac{P\pi R_K^2}{A}\right)^N \tag{12}$$

Introducing the parameters K and ω:

$$K = P\pi R_K^2 = \text{expected lethal area of one weapon}$$

$$\tag{13}$$

$$\omega = N/A \quad = \text{weapon density}$$

allows (12) to be written as:

$$S(\omega) = \left(1 - \frac{K\omega}{N}\right)^N \tag{14}$$

This gives for the destruction function:

$$F_N(\omega) = 1 - S(\omega) = 1 - \left(1 - \frac{K\omega}{N}\right)^N \tag{15}$$

487

Equation (15) still contains an extra parameter, N, which is the number of weapons in the area A used to derive (12)--presumed large compared to the effects of a single weapon and small compared to the total target size. We are currently interested in the limit as this area A becomes infinite compared to the effects of a single weapon, hence in the limit as $N \longrightarrow \infty$:

$$F_{\infty}(\omega) = \lim_{N \longrightarrow \infty} F_N(\omega) = 1 - e^{-K\omega} \tag{16}$$

which becomes our final destruction function for the locally random impact model.

"Perfect" Weapon Model: At the other extreme from the locally random impact model is the hypothetical situation where the weapons have zero CEP, delivery probability of unity, and completely destroy a hexagonal region of area K with no damage outside the region.

This situation closely resembles the case of "cookie-cutter" weapons of zero CEP and unit delivery probability, and deviates from the latter only when the area covered is so densely packed that the "cookie-cutter" circles begin to overlap--which does not occur until the fractional coverage exceeds $\pi/(2\sqrt{3})$ or about .91.

For such "perfect" weapons the destruction fraction is given by:

$$F = \begin{cases} K\omega & \omega < 1/K \\ 1 & \omega \geq 1/K \end{cases} \tag{17}$$

Intermediate Cases: We have considered two extremes, locally random impact, and perfect weapons. For actual situations, the targeting will not be random, but some optimum pattern of DGZs.

As the CEP becomes larger than the lethal radius, or the delivery probability becomes small, the situation--even though based on a pattern of DGZs--approaches a situation described by the random impact model. On the other hand, for high delivery probability and small CEP, the situation begins to approach the "perfect" weapon case--particularly as the weapon effect radius becomes sharp (close to "cookie-cutter"--e.g., the conventional σ_{20} model).

Returning to the destruction function given by (15) containing the extra parameter N (from which the random model was obtained by letting $N \longrightarrow \infty$), we observe the remarkable fact that for N=1, this function is precisely the damage function (17).

Since this function contains, for the extreme values of N, the two limits we have considered, it seems reasonable to suppose that any actual intermediate case could be adequately approximated by this function for some intermediate value of N.

We shall accordingly adopt this general function as our destruction function, subject to subsequent empirical verification.

The general law therefore becomes:

$$F_N(\omega) = \begin{cases} 1 - \left(1 - \dfrac{K\omega}{N}\right)^N & \omega < \dfrac{N}{K} \\ \\ 1 & \omega \geq \dfrac{N}{K} \end{cases} \tag{18}$$

For purposes of determining the optimum distribution of weapon density over a target of varying value density we wish to employ Eq. (9), for which we require the function $G = (F')^{-1}$. Accordingly,

$$F'_N(\omega) = \frac{d}{d\omega} F_N(\omega) = \begin{cases} K\left(1 - \dfrac{K\omega}{N}\right)^{N-1} & \omega < \dfrac{N}{K} \\ \\ 0 & \omega \geq \dfrac{N}{K} \end{cases} \tag{19}$$

for which the inverse function is easily determined to be:

$$G_N(X) = \frac{N}{K}\left[1 - \left(\frac{X}{K}\right)^{1/(N-1)}\right] \tag{20}$$

Thus from (9), the optimum weapon density is given by:

$$\omega_\lambda(V) = \begin{cases} \dfrac{1}{\mu(V)} \dfrac{N}{K} \left[1 - \left(\dfrac{\lambda}{KV\mu(V)}\right)^{\frac{1}{N-1}}\right] & \dfrac{\lambda}{KV\mu} < 1 \\ \\ 0 & \dfrac{\lambda}{KV\mu} \geq 1 \end{cases} \tag{21}$$

and for which the destruction fraction is easily calculated:

$$F_N(\omega_\lambda, \mu) = \begin{cases} 1 - \left(\dfrac{\lambda}{KV\mu(V)}\right)^{N/N-1} & \dfrac{\lambda}{KV\mu} < 1 \\ \\ 1 & \dfrac{\lambda}{KV\mu} \geq 1 \end{cases} \tag{22}$$

This completes the general treatment for arbitrary target value distributions.

Gaussian Targets: A particularly important special case is that of a Gaussian target, for which the value density distribution is given by:

$$V(x,y) = \frac{1}{2\pi\sigma^2} e^{-r^2/2\sigma^2} \tag{23}$$

(The total value is here normalized to unity.) From (23) we determine the relationship between radius and value to be:

$$r^2(V) = -2\sigma^2 \ln(2\pi\sigma^2 V) \tag{24}$$

and hence the cumulative area distribution function to be:

$$A(V) = \pi r^2(V) = -2\pi\sigma^2 \ln(2\pi\sigma^2 V) \quad \text{for } V \leq \frac{1}{2\pi\sigma^2} \tag{25}$$

and the differential element is:

$$dA(V) = -\frac{2\pi\sigma^2}{V} dV \tag{26}$$

Solution For Constant Vulnerability: Combining Eq. (10) with (26) and (22), and letting $\mu = 1$:

490

$$H_\lambda = \int_{\lambda/K}^{1/(2\pi\sigma^2)} V \left[1 - \left(\frac{\lambda}{KV} \right)^{\frac{N}{N-1}} \right] \left(\frac{2\pi\sigma^2}{V} \right) dV$$

$$= 1 - \frac{2\pi\sigma^2\lambda}{K} - (N-1) \left[\left(\frac{2\pi\sigma^2\lambda}{K} \right)^{\frac{N}{N-1}} - \frac{2\pi\sigma^2\lambda}{K} \right] \qquad (27)$$

Transforming the Lagrange multiplier λ to a new multiplier β:

$$\beta = \left[\frac{2\pi\sigma^2\lambda}{K} \right]^{1/(N-1)} \qquad (28)$$

we can rewrite (27) as:

$$H_\beta = 1 - \beta^{N-1} \left[1 + (N-1) \cdot (1 - \beta) \right] \qquad (29)$$

The total number of weapons as given by (10), (21), and (26):

$$W_\lambda = \int_{\lambda/K}^{1/(2\pi\sigma^2)} \frac{N}{K} \left[1 - \left(\frac{\lambda}{KV} \right)^{\frac{1}{N-1}} \right] \left(\frac{2\pi\sigma^2}{V} \right) dV \qquad (30)$$

leads, in terms of β, to:

$$W_\beta = \frac{N(N-1)2\pi\sigma^2}{K} \left[\beta - \ell n(\beta - 1) \right] \qquad (31)$$

In order to permit explicit exhibition of payoff as a function of number of weapons, it is necessary to define a new function, γ, which is the inverse of

$$y - \ell ny - 1 = x \qquad (32)$$

491

that is, $y = \gamma(x)$. It is defined for all nonnegative arguments, with values on the interval zero-one. With this function, (29) and (31) can be rewritten, in terms of surviving value:

$$S = \beta^{N-1} \left[1 + (N-1)(1 - \beta) \right]$$

$$\beta = \gamma\left(\frac{KW}{2\pi\sigma^2 N(N - 1)} \right) \tag{33}$$

Equations (33) summarize the relationship between surviving fraction, S, and number of weapons targeted, W, for Gaussian targets, and with a model parameter N, which can range from 1 to ∞.

The two limiting forms of (33), corresponding to $N = 1$ and $N \to \infty$ are interesting and important, and are easily shown to be:

$$S_1 = \exp(-KW/2\pi\sigma^2)$$

$$S_\infty = \left(1 + \frac{\sqrt{KW}}{\pi\sigma^2} \right) \exp\left(- \frac{\sqrt{KW}}{\pi\sigma^2} \right)$$

These are often termed the power law (or exponential law) and the square root law, respectively.

Derivation of Kill Probability Function

A variety of kill probability functions are in general use. The "normal model" employs a function of the form:

$$P_K(r) = e^{-r^2/2\sigma_K^2} \tag{34}$$

The "cookie-cutter" model employs a discontinuous function:

$$P_K(r) = \begin{cases} 1 & R_K \geq r \geq 0 \\ \\ 0 & r > R_K \end{cases} \tag{35}$$

492

where R_K is the so-called "lethal radius." The relation between R_K and σ_K is obtained by equating lethal areas

$$\pi R_K^2 = \int_0^{2\pi} \int_0^\infty e^{-r^2/2\sigma_K^2} \, r\,dr\,d\theta \tag{36}$$

leading to the relation

$$\sigma_K^2 = .5 R_K^2 \tag{37}$$

Other functions have often been used and, indeed, it has occasionally been found convenient to employ a generalized kill function of the form:

$$G_K(r) = e^{-K} \sum_{j=0}^{W-1} \frac{K_j}{j!} \tag{38}$$

where

$$K = \frac{Wr^2}{a^2}$$

Again, we can equate lethal areas to relate a with R_K:

$$\pi R_K^2 = \int_0^{2\pi} \int_0^\infty G_K(r) \, r\,dr\,d\theta \tag{39}$$

so that

$$R_K^2 = a^2 \text{ for all } W \tag{40}$$

The parameter W serves to alter the shape of this kill probability curve. Thus, $G_K(r)$ reduces to the normal curve for $W = 1$ and the cookie-cutter for $W \longrightarrow \infty$. Standard kill curves, such as the σ_{20} and σ_{30} curves of AFM 200-8, representing, respectively, ground burst and optimal air burst blast damage probabilities as a function of distance, can readily be approximated. $W = 6$ approximates closely the σ_{20} curve, and $W = 3$ approximates the σ_{30} curve.

493

Integration of a kill probability function over appropriate density functions allows the representation of such factors as delivery error, geodetic error, extended targets, etc.

Assume an extended target with the Gaussian normal value distribution as follows:

$$V(r) = \frac{1}{2\pi\sigma_{Tgt}^2} e^{-r^2/2\sigma_{Tgt}^2} \qquad (41)$$

$V(r)$ = value per unit area at distance r from center

σ_{Tgt} = standard deviation of value distribution

Clearly:

$$1.0 = \frac{1}{2\pi\sigma_{Tgt}^2} \int_{o}^{\infty} e^{-r^2/2\sigma_{Tgt}^2} \, dr \qquad (41)$$

Define a radius, R95, such at 95% of the value of the target is contained within this distance of the target center. (This R95 is the target radius used in the QUICK system.)

Then

$$\int_{o}^{R95} e^{-r^2/2\sigma_{Tgt}^2} \, dr = .95 \int_{o}^{\infty} e^{-r^2/2\sigma_{Tgt}^2} \, dr \qquad (43)$$

Solving this equation for σ_{Tgt} in terms of R95, we get:

$$\sigma_{Tgt} = R_{95}/2.448$$

Assume a CEP, the radius of a circle with center at an aiming point which will contain 50% of the centers of impact of weapons aimed at the aiming point. Assuming a circular normal (Gaussian) distribution of the aiming errors:

$$p(r) = \frac{r}{\sigma_{CEP}^2} e^{-r^2/2\sigma_{CEP}^2}$$

494

where

$p(r)$ = probability aiming error is r

σ_{CEP} = standard deviation of aiming errors

By definition of CEP

$$\int_0^{CEP} p(r)\ dr = 0.5$$

Solving for σ_{CEP} in terms CEP

$$\sigma_{CEP} = .8493 * CEP$$

Assume a weapon is aimed at the center of the target. From the nature of the Gaussian distribution we can define a standard deviation $\sigma_D^2 = \sigma_{CEP}^2 + \sigma_{Tgt}^2$ such that the circular normal distribution characterized by σ_D^2 is the convolution of the distributions characterized by σ_{CEP}^2 and σ_{Tgt}^2.

Therefore, if

$P_K(W)$ = probability of target kill

W = kill function parameter

$G_K(r)$ = kill function from Eq. (38)

then

$$P_K(W) = \frac{1}{2\pi\sigma_D^2} \int_0^\infty \int_0^{2\pi} \exp\left[-\frac{r^2}{2\sigma_D^2}\right] G_K(r)\ rd\theta \tag{46}$$

495

Evaluating the integrals

$$P_K(W) = 1 - \left(\frac{2WX^2}{1 + 2WX^2}\right)^W$$ (47)

where $X = \sigma_D/R_K$

or

$$P_K(W) = 1 - \left(\frac{\sigma_D^2}{\sigma_D^2 + \frac{1}{2W} R_K^2}\right)^W$$

which is the function used in QUICK.

APPENDIX B

OPTIMIZATION OF DGZs FOR COMPLEX TARGETS

Module ALOCOUT is responsible for selecting optimum desired ground zeros DGZs) for weapons allocated to complex targets. The complex target may contain several component target elements, each with specific coordinates, hardness, and some given time dependence of value. To place this diverse target element information on a commensurate basis for efficient DGZ selection, each target component of the complex is represented as a series of simple point value elements. Complex elements with more than one hardness component generate more than one such target element, and area targets generate several elements, spread over the area of the target, to represent a value spread over the area. A (DGZ) Desired Ground Zero Selector then uses the data to select optimum aim points within the target complex.
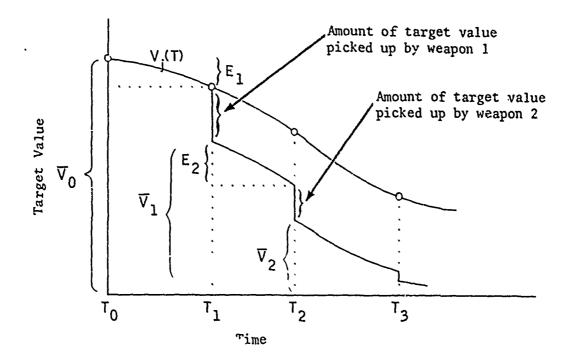
The selection of DGZs is a two-step process. First, the prescribed warheads are assigned initial coordinates through a "lay-down" process in which each successive warhead is targeted directly against the target element where the highest payoff is achieved, taking into account collateral damage to all other target elements. Second, a general-purpose function optimizer, FINDMIN, calculates the derivatives of the payoff as a function of x and y coordinates of each weapon and adjusts the coordinates to minimize the surviving target value. FINDMIN terminates either after a maximum number of iterations (which can be specified by the analyst) or after it finds that it can no longer make significant improvements in the payoff.

The mathematical representation used is as follows.

The weapons allocated to a complex target are to be placed in a manner which attempts to minimize the total escaping target value. To simplify discussion, the notation below is introduced. A second subscript, j, referencing the jth target element, is used when needed.

$\overline{V}_j$ = value of jth target element remaining immediately following arrival of the ith weapon

S_j = probability of survival of jth target element associated with weapon i

E_j = amount of value of jth target element that "escapes" between arrival of weapons $i - 1$ and i

T_i = time of arrival of weapon i (T_0 is an initial time when the full target value is applied) ($T_i \leq T_{i + 1}$ all i)

$V_j(T_i)$ = value of jth target, at time T_i

N = number of weapons

NT = number of targets

The following sketch illustrates the treatment of the time-dependent values of the jth target.

From this sketch, the following relationships should be apparent. The equations immediately below refer to a single target (j), but for simplicity the j subscript is omitted.

$$\overline{V}_i = V(T_i)S_i\overline{V}_{i-1}/V(T_{i-1}) \qquad (i = 1,2, \ldots, N)$$

$$E_i = \overline{V}_{i-1}\left[1 - V(T_i)/V(T_{i-1})\right] \quad (i = 1,2, \ldots, N+1)$$

From the previous equations,

$$\bar{V}_i = \left[\prod_{k=1}^{i} S_k\right] V(T_i) \quad \text{and} \quad E_i = \left[\prod_{k=1}^{i} S_k\right]\left[V(T_{i-1}) - V(T_i)\right]$$

(For $i = 1$, the product $\left(\prod_{k=1}^{i-1} S_k\right)$ is understood $= 1$. Also $V(T_{N+1}) = 0$.)
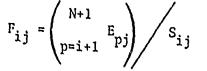
The total escaping value associated with target j is

$$\sum_{i=1}^{N+1} E_{ij} = \sum_{i=1}^{N+1}\left(\left[\prod_{k=1}^{i-1} S_{kj}\right]\left[V_j(T_{i-1}) - V_j(T_i)\right]\right)$$

The value on target j which escapes after arrival of weapon i is given by

$$\sum_{p=i+1}^{N+1} E_{pj}$$

The effective value of target j associated with weapon i defined by

$$F_{ij} = \left(\sum_{p=i+1}^{N+1} E_{pj}\right) \bigg/ S_{ij}$$

This value is introduced for computational efficiency and may be thought of as the total value available for weapon i, the effect of all other weapons having been taken into account.

The marginal value picked up on target j due to weapon i is given by

$$F_{ij}(1 - S_{ij})$$

where S_{ij} is a function of, among other things, the position of weapon i. For a fixed weapon configuration, weapon i can be moved from (x,y) to (x',y') and the marginal escaped value is given by:

$$\sum_{j=1}^{NT} F_{ij}(S_{ij} - S'_{ij})$$

To establish an initial weapon configuration, a lay-down is performed as follows. Initially, set $S_{ij} = 1$ for all i, j. Denote by S_{ik}^j the survival probability of the kth target, relative to the ith weapon, when this weapon is placed on the jth target. Now the ith weapon is placed on that target, j, which yields a maximum value for the expression

$$\sum_{k=1}^{NT} F_{ik}(S_{ik} - S_{ik}^j)$$

The S_{ik} are now set to equal to S_{ik}^j (k = 1, 2...., NT) the F_{ik} (all i, k) are redetermined, i is increased by one, and the process repeated until all weapons have been allocated.

This weapon configuration can now be input as the initial position to a "hill climber" routine, based on a steepest descent algorithm, which attempts to optimize further by replacing the discrete set of possible weapon positions with the two-dimensional continuum. The function to be minimized is:

$$\sum_{j=1}^{NT} \sum_{i=1}^{N+1} E_{ij}$$

Processing by the optimizer will be terminated either when the optimum has been achieved or when a specified number of iterations have been completed. In either case, to insure that the local optimum obtained cannot be further improved, the value of removing, in sequence, each of the weapons from its final location and placing it on one of the target points is explored. If the results obtained by this method are better than those achieved with the previous configuration, this new assignment will be used as an initial one for a second utilization of subroutine FINDMIN. If not, the results of the first use of subroutine FINDMIN will be kept.

APPENDIX C

GENERALIZED LAGRANGE MULTIPLIER METHOD FOR SOLVING PROBLEMS OF OPTIMUM ALLOCATION OF RESOURCES

Hugh Everett III

Weapons Systems Evaluation Division, Institute for Defense Analyses, Washington, D. C.

The usefulness of Lagrange multipliers for optimization in the presence of constraints is not limited to differentiable functions. They can be applied to problems of maximizing an arbitrary real valued objective function over any set whatever, subject to bounds on the values of any other finite collection of real valued functions defined on the same set. While the use of the Lagrange multipliers does not guarantee that a solution will necessarily be found for all problems, it is 'fail-safe' in the sense that any solution found by their use is a true solution. Since the method is so simple compared to other available methods it is often worth trying first, and succeeds in a surprising fraction of cases. They are particularly well suited to the solution of problems of allocating limited resources among a set of independent activities.

IN MOST textbook treatments, Lagrange multipliers are introduced in a context of differentiable functions, and are used to produce constrained stationary points. Their validity or usefulness often appears to be connected with differentiation of the functions to be optimized. Many typical operations-research problems, however, involve discontinuous or nondifferentiable functions (integral valued functions, for example), which must be optimized subject to constraints.

We shall show that with a different viewpoint the use of Lagrange multipliers constitutes a technique whose goal is *maximization* (rather than location of stationary points) of a function with constraints, and that in this light there are no restrictions (such as continuity or differentiability) on the functions to be maximized. Indeed, the domain of the function to be maximized can be any set (of any cardinal number) whatever.

The basic theorems upon which the techniques to be presented depend are quite simple and elementary, and it seems likely that some of them may have been employed previously. However, their generality and applicability do not seem to be well understood at present (to operations analysts at least). The presentation will consequently place primary emphasis on the implications and applications of the basic theorems, as well as

501

discussion of a number of techniques for extending the usefulness of the methods.

FORMULATION

FOR CLARITY of presentation, we shall develop the subject in a language of problems concerning the optimal allocation of resources. Other applications of the theorems will suggest themselves.

Let us suppose that there is a set S (completely arbitrary) that is interpreted as the set of possible strategies or actions. Defined on this strategy set is a real valued function H, called a *payoff function*. $H(x)$ is interpreted as the payoff (or utility) which accrues from employing the strategy $x \epsilon S$. In addition, there are n real valued functions $C^k(k=1\cdots n)$ defined on S, which are called *Resource* functions. The interpretation of these functions is that employment of the strategy $x \epsilon S$ will require the expenditure of an amount $C^k(x)$ of the kth resource.

The problem to be solved is the maximization of the payoff subject to given constraints $c^k, k=1\cdots n$, on each resource; i.e., to find

$$\max_{x \epsilon S} H(x)$$

subject to $C^k(x) \leqq c^k$, all k.

A particular subclass of this general problem with wide application is what will be called a *cell problem* (or separable problem) in which there are a number, m, of independent areas into which the resources may be committed, and for which the over-all payoff that accrues is simply the sum of the payoffs that accrue from each independent venture (cell). In this type of problem we have as before, for each cell, a strategy S_i, a payoff function H_i defined on S_i, and n resource functions C_i^k defined on S_i. $H_i(x_i)$ is the payoff in the ith cell for employing strategy $x_i \epsilon S_i$, and for each k, $C_i^k(x_i)$ is the amount of the kth resource expended in the ith cell by employing strategy x_i in that cell. In this case the problem to be solved is to find a strategy set, one element for each cell, which maximizes the total payoff subject to constraints c^k on the total resources expended; i.e.,

$$\max_{\substack{\text{all choices of } \{x_i\} \\ x_i \epsilon S_i}} \sum_{i=1}^{i=m} H_i(x_i)$$

subject to $\qquad \sum_{i=1}^{i=m} C_i^k(x_i) \leqq c^k \quad$ for all k.

This type of problem is simply a subclass of the previous general problem since it can be translated to the previous problem by the following identifications:

$$S = \prod_{i=1}^{i=m} S_i \text{ (direct product set)},$$

[where a strategy $x \epsilon S$ consists of an ordered m-tuple $(x_1, \cdots, x_n)$ of strategies, one for each S_i]

$$H(x) = \sum_{i=1}^{i=m} H_i(x_i),$$

$$C^k(x) = \sum_{i=1}^{i=m} C_i^k(x_i), \quad \text{all } k$$

MAIN THEOREM AND SOME OF ITS IMPLICATIONS

WE NOW present the main theorem concerning the use of Lagrange multipliers, and discuss its meaning and implications. The proof will be supplied in a later section.

THEOREM 1

1. λ^k, $k = 1, n$ are nonnegative real numbers,
2. $x^* \epsilon S$ maximizes the function

$$H(x) - \sum_{k=1}^{k=n} \lambda^k C^k(x) \text{ over all } x \epsilon S,$$

→3. x^* maximizes $H(x)$ over all those $x \epsilon S$ such that $C^k \leq C^k(x^*)$ for all k.

Discussion

This theorem says, for any choice of nonnegative λ^k, $k = 1, n$, that if an *unconstrained* maximum of the new (Lagrangian) function

$$H(x) - \sum_{k=1}^{k=n} \lambda^k C^k(x)$$

can be found (were x^*, say, is a strategy which produces the maximum), then this solution is a solution to that *constrained* maximization problem whose constraints are, in fact, the amount of each resource expended in achieving the unconstrained solution. Thus if x^* produced the unconstrained maximum, and required resources $C^k(x^*)$, then x^* itself produces the greatest payoff which can be achieved without using more of any resource than x^* does.

According to Theorem 1, one can simply choose an arbitrary set of nonnegative λ's, find an unconstrained maximum of the modified function, $H(x) - \sum_{k=1}^{k=n} \lambda^k C^k(x)$, and one has as a result a solution to a constrained problem. Notice, however, that the particular constrained problem which is solved is not known in advance, but arises in the course of solution and is, in fact, the problem whose constraints equal the resources expended by the strategy that solved the unconstrained problem.

In general, different choices of the λ^k's lead to different resource levels, and it may be necessary to adjust them by trial and error to achieve any given set of constraints stated in advance.

However, it is noteworthy that in most operations-research work one is not simply interested in achieving the optimum payoff for some given resource levels, but rather in exploring the entire range of what can be

obtained as a function of the resource commitments. In this case it matters little whether this function is produced by solving a spectrum of problems with constraints stated in advance, or by simply sweeping through the λ^k's to solve a spectrum of problems whose constraint levels are produced in the course of solution. The method when applicable is therefore quite efficient if the whole spectrum of constraints is to be investigated. Even in the case where only a single constraint set is of interest the use of this method, and adjustment of the λ^k's until the constraint set is achieved, is often more efficient than alternative procedures.

A limitation of the Lagrange multiplier method arises from the fact that it does not guarantee that an answer can be found in every case. It simply asserts that if an answer can be found it will indeed be optimum.

In cases where multiple constraints are involved that are not completely independent it may not be possible to simultaneously utilize all resources to the full allowance of the constraints. This can happen if the utilization of one resource requires the utilization of others, or equivalently in cases where some constraints may involve various combinations of others. These cases are analogous to problems in linear programming where certain constraints prove to be irrelevant in the optimum solution.

In such cases one might actually find the optimum solution but be unable to establish the optimality of the result because of incompletely utilized resources. Nevertheless, there is a large class of allocation problems in which the constraints really are independent (i.e., the resources can be consumed independently in the region of interest). In such cases solutions can usually be obtained that give consumption values adequately close to the constraint values. The existence of optimum solutions that can be found by this method actually depends upon an approximate concavity requirement in the region of the solution that will be discussed more carefully later.

At this point we wish to remind the reader of the generality of Theorem 1. *There are no restrictions whatever on the nature of the strategy set S, nor on the functions H and C^k other than real-valuedness.* The strategy set may therefore be a discrete finite set, or an infinite set of any cardinality. Furthermore, the payoff function and the resource functions can take on negative as well as positive values. [$C^k(x)$ negative may be interpreted as *production* rather than expenditure of the kth resource.]

Application to Cell Problem

One of the most important applications of Theorem 1 is in the solution of cell problems. As shown in the Formulation Section, these problems are a subclass of the general problem to which Theorem 1 is applicable. In this case, maximizing the unconstrained Lagrangian function

$$H(x) - \sum_{k=1}^{k=m} \lambda^k \, C^k(x)$$

is equivalent to finding

$$\max_{x, n \Vert_{i=1}^m s_i} [\sum_{i=1}^{i=m} H_i(x_i)] - \sum_{k=1}^{k=n} \lambda^k [\sum_{i=1}^{i=m} C_i^k(x_i)],$$

which (interchanging summation order) is the same as:

$$\max_{x, n \Vert_{i=1}^m s_i} \sum_{i=1}^{i=m} [H_i(x_i) - \sum_{k=1}^{k=n} \lambda^k C_i^k(x_i)].$$

But, since the choices x_i may be made independently in each cell as a consequence of $S = \prod_{i=1}^{i=m} S_i$, the sum is obviously maximized by simply maximizing

$$H_i(x_i) - \sum_{k=1}^{k=n} \lambda^k C_i^k(x_i)$$

in each cell independently of strategy choices in other cells, and summing the payoffs and resources expended for each cell (for the strategy that maximized the Lagrangian for that cell) to get the total payoff and resource expenditures. Theorem 1 then assures us that the result of this process is a solution to the over-all constrained problem with constraints equal to the total resources expended by the strategy produced by this procedure.

Observe that there is no possibility that just a local maximum to the over-all problem has been obtained. If the Lagrangian in each cell has been correctly maximized (i.e., is not itself merely locally maximized), then theorem 1 guarantees that the result is a *global* maximum to the over-all problem.

Theorem 1 says nothing about the manner in which one obtains the maxima of the unconstrained Lagrangian functions, but simply asserts that if one can find them, then one can also have maxima of a problem with constraints. The Lagrange multipliers therefore are not a way in themselves of finding maxima, but a technique for converting optimization problems with constrained resources into unconstrained maximization problems.

This conversion is especially crucial for cell problems with constraints on total resource expenditures, where the conversion to unconstrained maximization of the Lagrangian function uncouples what was an essentially combinatorial problem (because of the interaction of choices in each cell through total resource constraints) into a vastly simpler problem involving independent strategy selections in each cell.

The present treatment of Lagrange multipliers was motivated, in fact, by a cell problem involving continuous, differentiable payoff functions, the solution of which was attempted by a classical Lagrange multiplier approach. In this case, the resulting (transcendental) equations had in many circumstances a multiplicity of solutions, and the embarrassing problem arose as to which of several solutions to select for each cell. It appeared as though it might be necessary to try all combinations of choices of solutions—an impossible task in this case which involved several hun-

dred cells. As a result of this difficulty, a closer look was taken at the role of Lagrange multipliers, and the present treatment is the result. The original problem of multiple solutions is, of course, easily solved by simply selecting that solution in each cell which gives the largest value for the Lagrangian.

It is the recognition that the objective is to maximize the Lagrangian, by whatever means, not to zero its derivative, which is decisive. In many cases it *is* expeditious to maximize the Lagrangian by finding zeroes of its derivative. One can then easily select a final value by testing each solution (if there is more than one) to find which gives the largest (global) maximum. This procedure automatically excludes any solutions that correspond to minima or saddle values, and also facilitates taking into account any boundary conditions (such as nonnegative resource constraints) by testing the boundary cases as well.†

In other cases (particularly cases of nonnumerical strategies, or discrete strategy sets such as integers), the Lagrangian may best be maximized by trial and error procedures, or even direct computer scanning of all possibilities.

Another possibility is illustrated by cases wherein resources may be applied only in integral numbers. Often in such cases one can define a continuous differentiable payoff function that attains its correct value on the integers. A useful trick applicable to many such cases is to maximize analytically the Lagrangian based upon the continuous function, and then test the integer on each side of the solution, selecting the one that maximizes the Lagrangian.

PROOF OF MAIN THEOREM

THE PROOF of the main theorem presented and discussed in the previous section is quite elementary and direct:

Proof of Main Theorem. By assumptions (1) and (2) of Theorem 1, λ^k, $k = 1 \cdots n$, are nonnegative real numbers, and $x^* \epsilon S$ maximizes

$$H(x) - \sum_{k=1}^{k=n} \lambda^k C^k(x)$$

over all $x \epsilon S$ (the x^* producing the maximum may very well not be unique— all that we require is that x^* be *some* element that maximizes the Lagrangian). This means that, for all $x \epsilon S$,

$$H(x^*) - \sum_{k=1}^{k=n} \lambda^k C^k(x^*) \geqq H(x) - \sum_{k=1}^{k=n} \lambda^k C^k(x),$$

† This type of constraint (e.g., nonnegativity of resources), which holds independently for each cell rather than over-all as with total resources, is handled by simply restricting the strategy set for the cell appropriately. The Lagrange multipliers are reserved for over-all constraints.

506

and hence, that

$$H(x^*) \geqq H(x) + \sum_{k=1}^{k=n} \lambda^k [C^k(x^*) - C^k(x)]$$

for all $x \epsilon S$. But if the latter inequality is true for all $x \epsilon S$, it is necessarily true for any subset of S, and hence true on that subset S^* of S for which the resources never exceed the resources $C^k(x^*)$. Notationally: $x \epsilon S^* \Leftrightarrow$ for all k, $C^k(x) \leqq C^k(x^*)$. However, on the subset S^* the term

$$\sum_{k=1}^{k=n} \lambda^k [C^k(x^*) - C^k(x)]$$

is nonnegative by definition of the subset and the nonnegativity of the λ^k's, hence our inequality reduces to $H(x^*) \geqq H(x)$ for all $x \epsilon S^*$, and the theorem is proved.

LAMBDA THEOREM

THEOREM 2

1. Let $\{\lambda_1{}^k\}, \{\lambda_2{}^k\} k = 1 \cdots n$ be two sets of λ^k's that produce solutions $x_1{}^*$ and $x_2{}^*$, respectively. Furthermore, assume that the resource expenditures of these two solutions differ in only the jth resource.

$$C^k(x_1{}^*) = C^k(x_2{}^*) \text{ for } k \neq j$$

and that $C^j(x_1{}^*) > C^j(x_2{}^*)$.

2. Then: $\lambda_2{}' \geqq [H(x_1{}^*) - H(x_2{}^*)] / [C^j(x_1{}^*) - C^j(x_2{}^*)] \geqq \lambda_1{}'$.

This theorem states that, given two optimum solutions produced by Lagrange multipliers for which only one resource expenditure differs, the ratio of the change in optimum payoff to the change in that resource expenditure is bounded between the two multipliers that correspond to the changed resource.

Thus the Lagrange multipliers, which were introduced in order to constrain the resource expenditures, in fact give some information concerning the effect of relaxing the constraints.

In particular, if the set of solutions produced by Lagrange multipliers results in an optimum payoff that is a differentiable function of the resources expended at some point, then it follows from Theorem 2 that the λ's at this point are in fact the partial derivatives (or total derivative in case of one resource) of the optimum payoff with respect to each resource (all other resources kept constant):

$$[\partial H^* / \partial C^j]_{\substack{C_k \text{ constant} \\ k \neq j}} = \lambda^j.$$

Proof. The proof of Theorem 2 is also quite elementary. By hypothesis $x_1{}^*$ is the solution produced by $\{\lambda_1{}^k\}$, hence $x_1{}^*$ maximizes the Lagrangian for $\{\lambda_1{}^k\}$, which implies:

$$H(x_1{}^*) \geqq H(x) + \lambda_1{}^j [C^j(x_1{}^*) - C^j(x)] + \sum_{k \neq j} \lambda_1{}^k [C^k(x_1{}^*) - C^k(x)]$$

507

holds for all $x \epsilon S$, and hence in particular holds for $x_2{}^*$. But since by hypothesis $C^k(x_1{}^*) = C^k(x_2{}^*)$ for $k \neq j$, we can deduce that

$$H(x_1{}^*) \gtreqless H(x_2{}^*) + \lambda_1{}^j [C^j(x_1{}^*) - C^j(x_2{}^*)],$$

which, since by hypothesis $C^j(x_1{}^*) > C^j(x_2{}^*)$, implies that:

$$[H(x_1{}^*) - H(x_2{}^*)] / [C^j(x_1{}^*) - C^j(x_2{}^*)] \gtreqless \lambda_1{}^j,$$

which proves one side of the conclusion of Theorem 2. Interchanging the roles of $x_1{}^*$ and $x_2{}^*$ [and observing the reversal of the sign of

$$C^j(x_1{}^*) - C^j(x_2{}^*)]$$

produces the other side of the inequality to complete the proof of Theorem 2.

An obvious consequence of Theorem 2 is the fact that, if all but one resource level is held constant, the resource that changes is a monotone *decreasing* function of its associated multiplier. This fact indicates the direction to make changes when employing a trial and error method of adjusting the multipliers in order to achieve some given constraints on the resources.

The Lambda Theorem also suggests a potentially useful technique for choosing a starting set of multipliers for such a trial-and-error method of achieving given constraint levels in a cell problem. Beginning with any reasonably good allocation of the given resources, one can often calculate easily what the effect on the payoff is for a small additional increment of each resource, optimally placed within the cells. The differential payoff divided by the increment of resource is then taken as the starting λ for that resource. The λ's are then adjusted by trial and error until the Lagrange solution corresponds to the given constraints, producing the optimum allocation.

THE EPSILON THEOREM

A NATURAL question with respect to the practical application of the Lagrange method concerns its stability—supposing that as a result of methods of calculation or approximation one cannot precisely maximize the Lagrangian, but can only guarantee to achieve a value close to the maximum. Such a solution can very well be at a drastically different resource level and payoff than that which actually achieves the maximum, and yet produce a value of the Lagrangian very near to the maximum. For the method to be practical, it is required that in this situation a solution that nearly maximizes the Lagrangian must be a solution that also nearly maximizes the payoff for the resource levels that *it itself* produces (which may be quite different than those of the solution that actually

508

maximizes the Lagrangian). Only in such a circumstance would it be safe to assert that the solutions produced by any nonexact procedures (such as numerical computation with finite accuracy, or methods based upon approximations) were in fact approximately optimal solutions to the constrained problem. Such required assurance of insensitivity is supplied by the following ('epsilon') theorem.

THEOREM 3

1. $\bar{x}$ comes within ϵ of maximizing the Lagrangian, i.e., for all $x \epsilon S$:

$$H(\bar{x}) - \sum \lambda^k C^k(\bar{x}) > H(x) - \sum \lambda^k C^k(x) - \epsilon.$$

→2. $\bar{x}$ is a solution of the constrained problem with constraints $c^k = C^k(\bar{x})$ that is itself within ϵ of the maximum for those constraints.

The proof of this theorem, which is a simple extension of Theorem 1, exactly parallels the proof of Theorem 1 (with an added ϵ) and will not be repeated.

ADDITIONAL REMARKS, CONCLUSIONS, AND COMPUTATIONAL PLOYS

Gaps or Inaccessible Regions

Theorem 1 assures us that any maximum of the Lagrangian necessarily is a solution of the constrained maximum problem for constraints equal to the resource levels expended in maximizing the Lagrangian.

The Lagrange multiplier method therefore generates a mapping of the space of lambda vectors (components λ^k, $k = 1, \cdots, n$) into the space of constraint vectors (components c^k, $k = 1 \cdots n$). There is no a priori guarantee, however, that this mapping is onto—for a given problem there may be inaccessible regions (called *gaps*) consisting of constraint vectors that are not generated by any λ vectors. Optimum payoffs for constraints inside such inaccessible regions can therefore not be discovered by straight-forward application of the Lagrange multiplier method, and must hence be sought by other means.

The basic cause of an inaccessible region is nonconcavity in the function of optimum payoff vs. resource constraints (convexities in the envelope of the set of achievable payoff points in the space of payoff vs. constraint levels). This possibility, and several methods for dealing with it, will now be investigated.

Before beginning this investigation, however, we wish to point out that even though the Lagrange multiplier method is not certain to obtain the desired solutions in all cases, any solutions that it does yield are guaranteed by Theorem 1 to be true solutions. The procedure is therefore 'fail-safe,' a very reassuring property. It has been our experience over the last several years, which includes application of this method to a variety

509

of production and military allocation problems, that the method has been extremely successful, and nearly always has directly yielded all solutions of interest. The few situations in which the direct method failed were readily solved by simple modifications to the procedure, some of which will now be mentioned.

Source of Gaps

Consider the $(n+1)$ dimensional space of payoff vs. resource expenditures. This space will be called PR space for brevity. Every strategy $x \epsilon S$ maps into a point in this space corresponding to $H(x), C^k(x) (k=1 \cdots n)$. The entire problem is therefore represented by this set of accessible points in PR space. The problem of finding the maximum of H subject to constraints c^k, $k=1 \cdots n$, is simply the problem of selecting that point of our set in PR space of maximum H that is contained in the subspace of PR space where the resources are bounded by the c^k's. The set of all such points (corresponding to all sets of values in the c^k's) will be called the *envelope*, and constitutes the entire set of solutions for all possible constraint levels.

Consider now any solution x^* produced by a set of Lagrange multipliers (λ^k). By definition x^* maximizes the Lagrangian; consequently we have that

$$H(x^*) - \sum \lambda^k C^k(x^*) \geqq H(x) - \sum \lambda^k C^k(x)$$

for all $x \epsilon S$. Rearranging terms slightly, we have:

$$H(x) \leqq H(x^*) - \sum \lambda^k C^k(x^*) + \sum \lambda^k C^k(x)$$

for all $x \epsilon S$. If we consider now the hyperplane in PR space defined by $H = \sum \lambda^k C^k + \alpha$ where $\alpha = H(x^*) - \sum \lambda^k C^k(x^*)$, we see that, because of the previous inequality, none of the accessible points in PR space lies above this hyperplane, and at least one point, $H(x^*), C^k(x^*)$ $k=1 \cdots n$, lies on it.

Each solution produced by Lagrange multipliers therefore defines a bounding hyperplane that is tangent to the set of accessible points in PR space at the point corresponding to the solution (hence tangent to the envelope), and which constitutes an upper bound to the entire set of accessible points. It is clear that, since no such tangent bounding hyperplanes exist in regions where the envelope of accessible points in PR space is not concave, the Lagrange multiplier method cannot produce solutions in such a region. Conversely, for any point on the envelope (solution) where a tangent bounding hyperplane *does* exist (envelope concave at the point), it is obvious that there exists a set of multipliers (namely the slopes of the hyperplane) for which the strategy corresponding to the point in question maximizes the Lagrangian.

Thus the Lagrange method will succeed in producing all solutions that correspond to concave regions of the envelope (optimized payoff vs. constraint level), and fail in all nonconcave regions.

A fortunate feature of cell problems with many cells is the fact that, even though there may be large convexities in the envelope in the PR space for each cell, the result of over-all optimization is an envelope in the PR space for the total problem in which the convexities are vastly reduced in significance.[†] This property is the major reason for the general success of the Lagrange method in solving cell problems.

Some Methods for Handling Gaps

Despite the general success of Lagrange multipliers (at least for the problems we have encountered), occasions may arise where gaps occur in regions of critical interest. Under such circumstances there are several useful techniques that can be attempted before abandoning the procedure altogether.

First, all solutions that can be obtained outside the gaps contribute a good deal of information and can be used to bound the solution in the gap region. As was previously shown, each solution that can be obtained by Lagrange multipliers defines a bounding hyperplane that gives an upper bound to the maximum payoff at all points, and hence inside the gap as well. For any point inside a gap, therefore, an upper bound can be obtained by finding the minimum payoff for that point over the set of bounding hyperplanes corresponding to the solutions that one could calculate.

On the other hand, every solution that can be obtained that has the property that none of its resource expenditures exceeds the resources of a point in a gap for which one is seeking bounds, obviously constitutes a *lower bound* to the optimum payoff at the point in question, and the maximum of these lower bounds can be selected as a lower bound to the payoff in question. Thus the set of solutions that *can* be obtained by Lagrange multipliers can be used to obtain bounds on the optimum payoff for inaccessible regions.

There is another technique that is often successful in reducing gaps in instances where the bounds one can compute leave too large a region of uncertainty, and where the gap is caused by degeneracy in which a number of cells have gaps corresponding to the same multiplier. A gap is char-

[†] In fact, the gap structure for the over-all problem obviously simply reflects faithfully the gap structure in the individual cells, with each gap in a cell corresponding to a given multiplier value occurring with the same magnitude (same jump in payoff and resources) in the over-all optimization at precisely the same multiplier value. Only degeneracies in which several cells have gaps corresponding to the same multiplier can cause a larger gap in the over-all problem, and such degeneracy is easily removed by techniques to be discussed in the following section.

acterized by the behavior that, as the λ's are continuously varied, there are abrupt discontinuities in the resource levels generated. These discontinuities can often be filled in cell problems by the following technique.

Given two sets of λ's, $(\lambda_1{}^k)$, $(\lambda_2{}^k)$, which are very close, but for which the generated resource levels markedly differ, one can make a *mixed* calculation in a cell problem using the set $(\lambda_1{}^k)$ in some cells and the set $(\lambda_2{}^k)$ in the others. If the two sets of λ's are close together, maximizing the Lagrangian in any cell for one set will necessarily result in a solution that nearly maximizes the Lagrangian for the other set, hence by the Epsilon Theorem will yield a result that is guaranteed to be nearly optimum.

Somewhat more generally, one can simply exploit the Epsilon Theorem directly in a cell problem, working with a given set of λ's but deliberately modifying the choices in some or all cells in a way which moves in the direction of the desired expenditure of resources. By summing the deviations from maximum of the Lagrangian in each cell (epsilons) in which the strategies are so modified, a bound on the error of the result is obtained (which can be kept quite small in most cases by judicious choice of deviations). This appears to be a quite powerful stratagem.

.
.
.

512

DISTRIBUTION

Addressee	Copies

CCTC Codes

Technical Library (C124)	3
C124 (Stock)	6
C126	2
C313	1
C314	15
C600	1

DCA Code

205	1

EXTERNAL

Chief, Studies, Analysis and Gaming Agency, OJCS
ATTN: SFD, Room 1D957, Pentagon, Washington, DC
20301   2

Chief of Naval Operations, ATTN: OP-96C4, Room 4A478,
Pentagon, Washington, DC 20350   2

Commander-in-Chief, North American Air Defense Command
ATTN: NPXYA, Ent Air Force Base, CO 80912   2

Commander, U. S. Air Force Weapons Laboratory (AFSC)
ATTN: AFWL/SUL (Technical Library),
Kirtland Air Force Base, NM 87117   1

Director, Strategic Target Planning, ATTN: (JPS), Offutt
Air Force Base, NE 68113   2

Defense Documentation Center, Cameron Station,
Alexandria, VA 22314   12
                                                                              50

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CSM MM 9-77, Volume III	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* THE CCTC QUICK-REACTING GENERAL WAR GAMING SYSTEM (QUICK), Users Manual, Weapon Allocation Subsystem		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)* Dale J. Sanders Paul F. M. Maykrantz Jim M. Herron		8. CONTRACT OR GRANT NUMBER*(s)* DCA 100-75-C-0019
9. PERFORMING ORGANIZATION NAME AND ADDRESS System Sciences, Incorporated 4720 Montgomery Lane Bethesda, Maryland 20014		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Command and Control Technical Center Room BE-685, The Pentagon, Washington, DC 20301		12. REPORT DATE 15 April 1978
		13. NUMBER OF PAGES 526
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*
Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

PRECEDING PAGE BLANK

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

War Gaming, Resource Allocation

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*
The computerized Quick-Reacting General War Gaming System (QUICK) will accept input data, automatically generate global strategic nuclear war plans, provide statistical output summaries, and produce input tapes to simulator subsystems external to QUICK.
The Program Maintenance Manual consists of four volumes which facilitate maintenance of the war gaming system. This volume, Volume III, provides the programmer/analyst with a technical description of the purpose, functions, general procedures, and programming techniques applicable to the modules and subroutines

DD FORM
1 JAN 73 1473   EDITION OF 1 NOV 65 IS OBSOLETE

20. ABSTRACT (Continued)

of the Weapon Allocation Subsystem.

The Program Maintenance Manual complements the other QUICK Manuals to facilitate application of the war gaming system. These manuals Series 9-77 are published by the Command and Control Technical Center (CCTC), Defense Communications Agency (DCA), The Pentagon, Washington, DC 20301.