

**NATIONAL
MILITARY
COMMAND
SYSTEM
SUPPORT
CENTER**



**DEFENSE
COMMUNICATIONS
AGENCY**

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

AD 742783

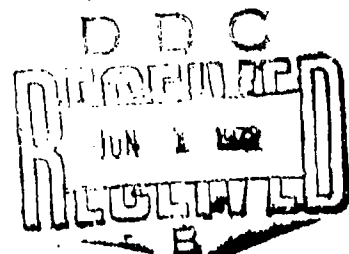
COMPUTER SYSTEM MANUAL
CSM PSM 9A-67
VOLUME I, PART A
29 FEBRUARY 1972

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

DATA INPUT SUBSYSTEM

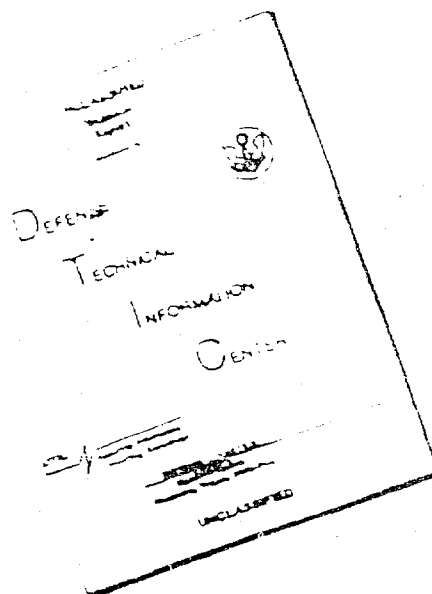
**PROGRAMMING SPECIFICATIONS
MANUAL**

NATIONAL TECHNICAL
INFORMATION SERVICE



485

DISCLAIMER NOTICE



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

THIS DOCUMENT CONTAINED
BLANK PAGES THAT HAVE
BEEN DELETED

REPRODUCED FROM
BEST AVAILABLE COPY

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D		
(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)		
1. ORIGINATING ACTIVITY (Corporate author) National Military Command System Support Center (NMCSSC) Defense Communications Agency (DCA) The Pentagon Washington, DC 20301		2a. REPORT SECURITY CLASSIFICATION
		2b. GROUP
3. REPORT TITLE The NMCSSC Quick-Reacting General War Gaming System (QUICK) Programming Specifications Manual, Volume I, Data Input Subsystem		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) N/A		
5. AUTHOR(S) (First name, middle initial, last name) NMCSSC: Yvonne Mapily Donald F. Webb		Lambda Corp: Betty J. Ellis Jack A. Sasseen
6. REPORT DATE 29 February 1972	7a. TOTAL NO. OF PAGES 486	7b. NO. OF REFS 4
8a. CONTRACT OR GRANT NO. DCA 100-70-C-0065	8b. ORIGINATOR'S REPORT NUMBER(S) NMCSSC COMPUTER SYSTEM MANUAL CSM PSM 9A-67	
8c. PROJECT NO. NMCSSC Project 631		
8d.	8d. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None	
10. DISTRIBUTION STATEMENT This document is approved for public release; its distribution is unlimited.		
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY National Military Command System Support Center/Defense Communications Agency The Pentagon, Washington, DC 20301
13. ABSTRACT This is one of three volumes describing the computer programming specifications for the Quick-Reacting General War Gaming System (QUICK). This volume addresses computer programs of the QUICK Data Input Subsystem. It is intended to serve as the basis for program maintenance activities. Accordingly, it describes the program functions and contains flow charts for each program and subprogram of the Data Input Subsystem. Based upon suitable data base and user control parameters, QUICK will generate individual bomber and missile plans suitable for war gaming, and simulate the planned events. The generated plans are of a form suitable for independent review and revision. Subsequently, the planned events are simulated; various statistical summaries are produced to reflect the results of the war game. A variety of force postures and strategies can be accommodated. QUICK is documented extensively in a set of Computer System Manuals (series 9-67) published by the National Military Command System Support Center (NMCSSC), Defense Communications Agency (DCA), The Pentagon, Washington, DC 20301.		

DD FORM 1473

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

479

Security Classification

NATIONAL MILITARY COMMAND SYSTEM SUPPORT CENTER

Computer System Manual Number CSM PSM 9A-67

29 February 1972

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

Programming Specifications Manual

Volume I - Data Input Subsystem

Part A

Submitted by:

Donald F. Webb

DONALD F. WEBB
Major, USAF
Project Officer

REVIEWED BY:

R. E. Harshbarger

R. E. HARSHBARGER
Technical Director
NMCSSC

APPROVED BY:

Bruce Merritt

BRUCE MERRITT
Colonel, USA
Commander, NMCSSC

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

This document has been approved for public release; distribution unlimited.

ACKNOWLEDGMENT

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

CONTENTS

Part A

Chapter		Page
	ACKNOWLEDGMENT	ii
	ABSTRACT	xiii
1	INTRODUCTION	1
2	QUICK SYSTEM FILEHANDLER	5
	Purpose	5
	Concept of Operation	5
	Functional Description	8
	Initialization and Directory Maintenance	8
	Output Data Transfer	11
	Input Data Transfer	13
	Dump Facility	14
	Error Detection and Recovery	14
	Common Blocks	15
	Subroutine FILEHNR	21
	(Entry ALOCDIR)	
	(Entry DEACTIV)	
	(Entry INITAPE (INITAP))	
	(Entry PRNTLAB)	
	(Entry RDWORD)	
	(Entry SETWRITE (SETWRIT))	
	(Entry WRWORD)	
	Subroutine RDARRAY	35
	(Entry RDARRAY)	
	Subroutine SETREAD	43
	Subroutine TERMTAP	50
	(Entry TERMTAPE)	
	(Entry TERMTPE)	
3	SPECIAL-PURPOSE UTILITY ROUTINES	56
	Program OUTFILE	57
	Subroutine CLOSPIL	67
	Subroutine CLRCMON	69
	Subroutine GETDF	71
	Subroutine INBUFDK	77
	Subroutine INERRDK	80
	Subroutine INLABEL	83

Chapter		Page
	Subroutine NODIRC	85
	Subroutine OPENSPL	87
	Subroutine OUTBFTP	90
	Subroutine OUTERTP	92
	Subroutine SETHEAD	96
	Program RELOADF	98
	Subroutine ENDTAPE	102
	Subroutine GETLOC	104
	Subroutine INBUFTP	106
	Subroutine INERRTP	108
	Subroutine NEXTAPE	113
	Subroutine OUTBFDK	116
	Subroutine OUTDF	119
	Subroutine OUTERDK	126
	Program DECLARES	129
	Subroutine EQUIV	135
	Program FILEDUMP	137
4	GENERAL UTILITIES	140
	Subroutine ABORT	141
	(Entry WARNING)	
	Subroutine ANOTHER (COMPASS)	143
	Function ATN2PI	144
	Subroutine CHANGE	146
	Function DELONG	148
	Function DIFFLONG	150
	(Entry DIFFLNG)	
	Function DISTF	152
	Function DSTF	154
	Subroutine ENDDATA	156
	Subroutine ERAZE (COMPASS)	157
	Function GETCLOCK	158
	(Entry GETCLK)	
	Function GETDATE (COMPASS)	159
	Subroutine GETLIMIT (COMPASS)	160
	Subroutine GETVALU	162
	Function IGET	166
	Subroutine INITEDIT	168
	(Entry INITEDT)	
	Subroutine INPITEM	170
	(Entry NEXTITEM)	
	(Entry NEXTITM)	

Chapter

Page

Subroutine INTERP	173
Subroutine INTERPGC	176
(Entry INTRPGC)	
Subroutine IPUT	181
Function ITLE	183
Function IWANT	185
Function KEYMAKE	187
Function LOCF (COMPASS)	189
Subroutine LOCREAD	190
(Entry LOCWRIT)	
(Entry LOCWRITE)	
Subroutine NEWUNIT (COMPASS)	192
Subroutine NEXTFILE	193
Function NUMGET	195
Subroutine ORDER	198
Subroutine OUTITEM	200
Subroutine OUTWORDS	202
(Entry OUTWRDS)	
Subroutine PAGESKP	204
(Entry PAGESKIP)	
Subroutine PRITEM	205
Subroutine PRNTBASE	207
(Entry PRNTBAS)	
(Entry PRNTBSE)	
Subroutine PRNTDIRC	209
(Entry PRNTDRC)	
Subroutine PRNTDTA	213
(Entry PRNTDATA)	
Subroutine PRNTPGE	216
(Entry PRNTPAGE)	
Subroutine READDR	218
Subroutine REORDER	221
Subroutine SKIP	224
Subroutine SKIPFILE	226
(Entry BACKFILE)	
Function SSKPC	227
Subroutine STORAGE	231
Function TIMEDAY (COMPASS)	233
Subroutine TIMEME	234
Subroutine WRITEDIR	237
(Entry WRITEDR)	
5 PROGRAM QUIKBASE	240
Purpose	240

Chapter		Page
	Input	240
	Program Option Cards	240
	Data Library File	240
	Update Command Cards	241
	Item Update Files	242
	Output	242
	Concept of Operation	243
	Creation of a Data Library File (SETID) Option	243
	Updating a Data Library File (UPDATE Option)	245
	Data Base Generation (QUIKDBG Option)	246
	Printing of a Data Base File (PRINTDB) Option	246
	Identification of Subroutine Functions	247
	SETID Option (Creation of Data Library File)	247
	UPDATE Option (Updating of Data Library File)	247
	QUIKDBG Option (Data Base File Creation)	247
	PRINTDB Option (Printing of a Data Base File)	247
	Common Block Definition	248
	External Common Blocks	248
	Internal Common Blocks	248
	Subroutine ADDSET	258
	Subroutine BUFFIT	260
	Subroutine CARDCK	263
	Subroutine COPYDB	265
	Subroutine COUNTDS	267
	Subroutine FASTSET	269
	Function ILOOK	271
	Subroutine INITFAST	273
	Subroutine INPRCTL	275
	Function IPRINT	277
	Subroutine MAKEBAS	279
	Subroutine MAKEIT	281
	Subroutine MOVEIT	283
	Subroutine NEWBASE	285
	Subroutine NEWDATA	293
	(Entry NEWCARDS)	
	Subroutine NEWDIR	302
	Subroutine OUT	307
	Subroutine PRONLY	309
	Subroutine PRTCONT	311
	Subroutine SETID	313
6	PROGRAM BASEMOD	322
	Purpose	322
	Input Files	322
	Output Files	323

Chapter		Page
	Concept of Operation	323
	Identification of Subroutine Functions	323
	Post-QUIKBASE Operation	324
	Post-INDEXER Operation	324
	Common Block Definition	324
	External Common Blocks	324
	Internal Common Blocks	324
	Program BASEMOD	330
	Subroutine ADDVAL	332
	(Entry PRNTVAL)	
	Subroutine COUNTDES	334
	Subroutine DBMOD	336
	Function INDEXTYP	340
	Subroutine INDMOD	342
	Function MYZONE	345
	Subroutine NUMDEL	347
	Subroutine PRINTIT	349
	Subroutine PRTCOUNT	351
	Subroutine RDTYPES	353
	Subroutine STKRIN	355
	Subroutine TARDEFS	357
7	PROGRAM INDEXER	360
	Purpose	360
	Input	360
	Output	360
	Concept of Operation	361
	General	361
	Pass 1 Processing (Sheets 1 to 6, Figure 111).	368
	Pass 2 Processing (Sheets 6 to 11, Figure 111)	370
	Pass 3 Processing (Sheets 11 to 18, Figure 111).	372
	Common Block Definition	376
	Subroutine AROVRFL	404
	Subroutine COLOCATE	406
	(Entry COLOCAT)	
	Subroutine FINDIT	416
	Function ICPL	418
	Function IDXF	420
	Subroutine INITIND	422
	Subroutine READIN	423
	Subroutine TDEFSTT	426
	(Entry TDEFSTAT)	
	Subroutine VLRADI	429
	Subroutine WRPRNT	431
	Subroutine WRSIMT	433

Chapter		Page
8	PROGRAM BASESUM	436
	Method	436
	Common Block Definition	437

APPENDIXES

A.	Utility Routine Common Blocks	442
B.	QUICK Attribute Names and Descriptions	447
C.	Entry Points for QUICK Utility Routines	458
D.	Utilization of General Utility Routines	462
	DISTRIBUTION	471
	DD Form 1473	472

PART B

QUICK Utility Program/Subroutine Listings

The QUICK Filehandler	473
Special Utility Routines	538
General Utility Routines	676

PART C

Data Input Subsystem Program Listings

Program QUIKBASE	841
Program BASEMOD	966
Program INDEXER	1063
Program BASESUM	1205

ILLUSTRATIONS

Number		Page
1	The Data Input Subsystem	2
2	Physical File Format	9
3	Subroutine FILEHNR	25
4	Subroutine RDARRAY	38
5	Subroutine SETREAD	45
6	Subroutine TERMTAP	52
7	Program OUTFILE	64
8	Subroutine CLOSPIL	68
9	Subroutine CLRCMON	70
10	Subroutine GETDF	73
11	Subroutine INBUFDK	78
12	Subroutine INERRDK	81
13	Subroutine INLABEL	84
14	Subroutine NODIRC	86
15	Subroutine OPENSPL	88
16	Subroutine OUTBFTP	91
17	Subroutine OUTERTP	94
18	Subroutine SETHEAD	97
19	Program RELOADF	100
20	Subroutine ENDTAPE	103
21	Subroutine GETLOC	105
22	Subroutine INBUFTP	107
23	Subroutine INERRTP	110
24	Subroutine NEXTAPE	114
25	Subroutine OUTBFDK	117
26	Subroutine OUTDF	121
27	Subroutine OUTERDK	127
28	Program DECLARES	132
29	Subroutine EQUIV	136
30	Program FILEDUMP	139
31	Subroutine ABORT	142
32	Subroutine ANOTHER	143
33	Function ATN2PI	145
34	Subroutine CHANGE	147
35	Function DELLONG	149
36	Function DIFFLONG	151
37	Function DISTF	153
38	Function DSTF	155
39	Subroutine ENDDATA	156
40	Subroutine ERAZE	157
41	Function GETCLOCK	158
42	Function GETDATE	159
43	Subroutine GETLIMIT	161
44	Subroutine GETVALU	165

Number		Page
45	Function IGET	167
46	Subroutine INITEDIT	169
47	Subroutine INPITEM	171
48	Subroutine INTERP	175
49	Coordinate System for INTERPGC	177
50	Subroutine INTERPGC	180
51	Subroutine IPUT	182
52	Function ITLE	184
53	Function IWANT	186
54	Function KEYMAKE	188
55	Function LOCF	189
56	Subroutine LOCREAD	191
57	Subroutine NEWUNIT	192
58	Subroutine NEXTFILE	194
59	Function NUMGET	197
60	Subroutine ORDER	199
61	Subroutine OUTITEM	201
62	Subroutine OUTWORDS	203
63	Subroutine PAGESKP	204
64	Subroutine PRITEM	206
65	Subroutine PRNTBASE	208
66	Subroutine PRNTDIRC	212
67	Subroutine PRNTDTA	214
68	Subroutine PRNTPGE	217
69	Subroutine READDIR	220
70	Subroutine REORDER	222
71	Subroutine SKIP	225
72	Function SSKPC	229
73	Subroutine STORAGE	232
74	Function TIMEDAY	233
75	Subroutine TIMEME	236
76	Subroutine WRITEDIR	239
77	Program QUIKBASE	254
78	Subroutine ADDSET	259
79	Subroutine BUFFIT	261
80	Subroutine CARDCK	264
81	Subroutine COPYDB	266
82	Subroutine COUNTDS	268
83	Subroutine FASTSET	270
84	Function ILOOK	272
85	Subroutine INITFAST	274
86	Subroutine INPRTCL	276
87	Function IPRINT	278
88	Subroutine MAKEBAS	280
89	Subroutine MAKEIT	282
90	Subroutine MOVEIT	284

Number		Page
91	Subroutine NEWBASE	287
92	Subroutine NEWDATA	295
93	Subroutine NEWDIR	304
94	Subroutine OUT	308
95	Subroutine PRONLY	310
96	Subroutine PRTCONT	312
97	Subroutine SETID	315
98	Program BASEMOD	331
99	Subroutine ADDVAL	333
100	Subroutine COUNTDES	335
101	Subroutine DBMOD	338
102	Function INDEXTYP	341
103	Subroutine INDMOD	343
104	Function MYZONE	346
105	Subroutine NUMDEL	348
106	Subroutine PRINTIT	350
107	Subroutine PRTCOUNT	352
108	Subroutine RDTYPES	354
109	Subroutine STKRIN	356
110	Subroutine TARDEFS	359
111	Program INDEXER	386
112	Subroutine AROVRFL	405
113	Subroutine COLOCATE	408
114	Subroutine FINDIT	417
115	Function JCPL	419
116	Function IDXF	421
117	Subroutine INITIND	422
118	Subroutine READIN	425
119	Subroutine TDEFSTT	428
120	Subroutine VLRADI	430
121	Subroutine WRPRNT	432
122	Subroutine WRSIMT	434
123	Program BASESUM	439

TABLES

Number		Page
1	Filehandler Label Content	7
2	Filehandler Common Blocks	17
3	Common Blocks (Programs OUTFILE and RELOADF)	59
4	Data Block Format for DATADB File	243
5	Data Base File (Logical Record Format)	244
6	Program QUIKBASE Common Blocks	249
7	Program BASEMOD Common Blocks	325
8	Indexed Data Base File (Logical Record Format)	362
9	SIMTAPE Format	364
10	Warhead, ASM, Payload, and DBL Data	373
11	Structure of a Word in the Array STATUS	375
12	Structure of a Word in the Array IOVERLAP	375
13	Program INDEXER Common Blocks	377
14	Description of COLOCATE Arrays	407
15	Program BASESUM Common Blocks	438

ABSTRACT

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

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

1. GENERAL DESCRIPTION
Computer System Manual CSM GD 9A-67
A nontechnical description for senior management personnel
2. ANALYTICAL MANUAL
Computer System Manual CSM AM 9A-67 (three volumes)
Provides a description of the system methodology for the non-programmer analysts
3. USER'S MANUAL
Computer System Manual CSM UM 9-67 (two volumes)
Provides detailed instructions for applications of the system
4. OPERATOR'S MANUAL
Computer System Manual CSM OM 9A-67
Provides instructions and procedures for the computer operators

CHAPTER 1 INTRODUCTION

The QUICK system consists of four functional subsystems: the Data Input, Plan Generation, Simulation, and Data Output subsystems. In addition, QUICK employs a general-purpose utility package. This utility package consists of programs, subroutines, and functions which perform a variety of support tasks common to two or more system programs.

This volume of the Programming Specifications Manual describes the programs which make up the QUICK Data Input subsystem and provides a description of the programs, subroutines, and functions which comprise the QUICK utility package. The organization of this volume is described below.

Since the utility programs performed are common to all of the programs of the Data Input subsystem, these programs are presented first in Chapters 2-4. Chapter 2 is a description of the file handling system (subroutine FILEHNR) used by all QUICK programs for tape and disk operations to speed read/write operations. Chapter 3 provides a description of the purpose, functions, and operation of the special-purpose programs listed below.

1. OUTFILE and RELOADF. These programs provide a restart capability used in conjunction with the Plan Generation subsystem. They enable the user to interrupt processing in the middle of the subsystem and either repeat or resume processing at a later time.
2. DECLARES. This program is used to assist the NMCSSC analyst in writing and maintaining programs which process QUICK data base tapes. It places into programs the proper FORTRAN COMMON, EQUIVALENCE, and TYPE statements required for the data base being processed.
3. FILEDUMP. This program provides the capability to print specific portions of a magnetic tape, written in binary mode, or a disk file.

Chapter 4 describes the remaining programs, subroutines, and functions of the utility package, which perform a variety of tasks through the system.

The subsequent chapters provide detailed descriptions of the programs of the Data Input subsystem. A description of the data flow within this subsystem, shown in figure 1, and the programs involved are summarized below.

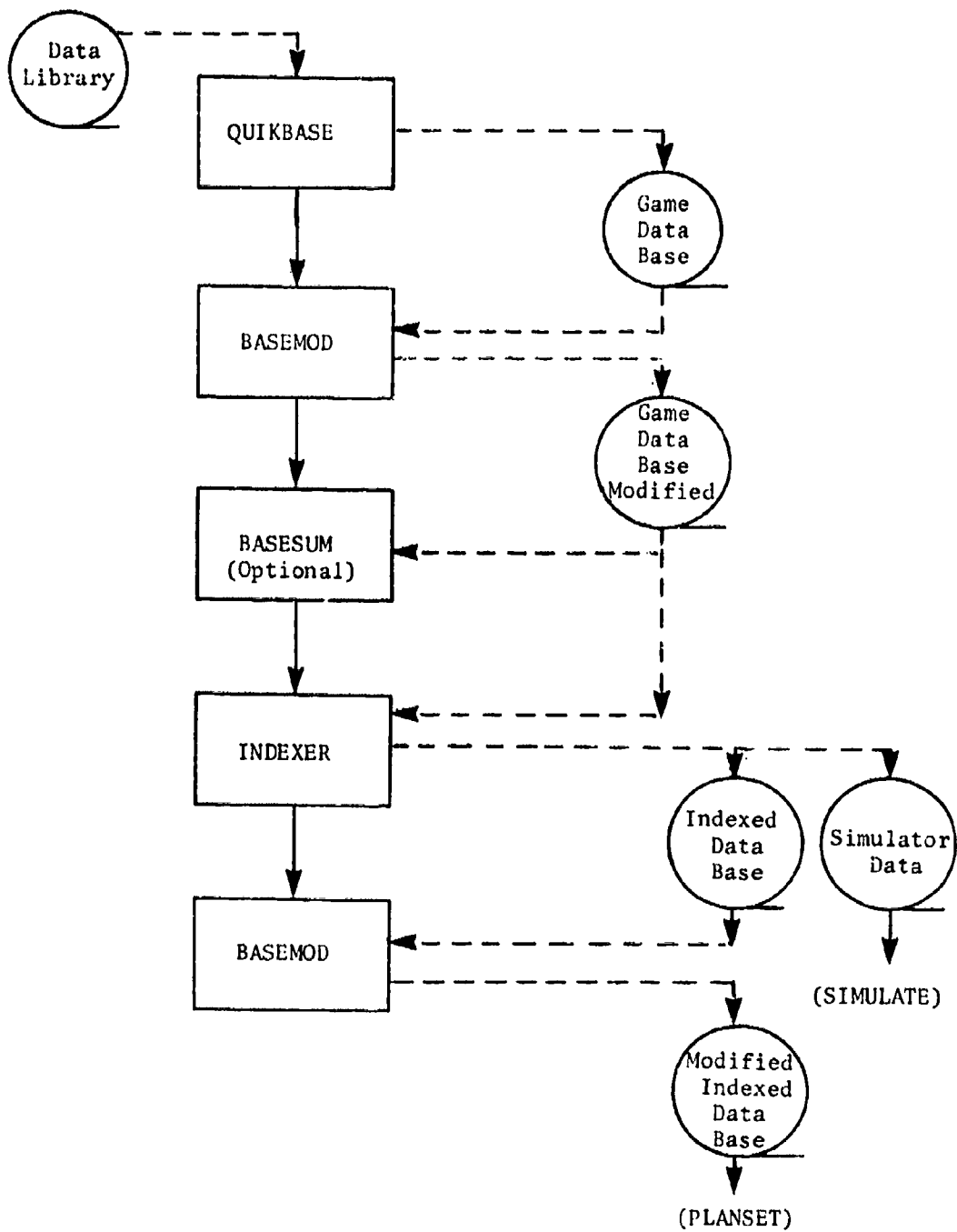


Fig. 1. The Data Input Subsystem

1. Program QUIKBASE performs the primary function of creating a game base file which defines the general data to be used by succeeding programs of the Data Input subsystem. This program accepts as input the data library (in tape or card form) and, based on user-input parameters, modifies the file to create a game base file (QUIKDB). As an optional feature, this program provides the facilities to print the game base file (QUIKDB) in a format meaningful to the user.
2. Program BASEMOD was developed in response to a specific NMCSSC support task. As described herein, the main function of program BASEMOD is to alter the content or characteristics of a data base in order to adapt it to the specific scenario for which a plan or set of plans is being developed. As indicated in figure 1, this program may be used after program QUIKBASE and after program INDEXER to introduce user-desired modifications in the game file.
3. The purpose of program BASESUM is to summarize a game base file and to print these summaries in tabular form. While figure 1 reflects the program operating after program BASEMOD, the program may be used to summarize the data base contained on the output tapes produced by programs QUIKBASE, BASEMOD, or INDEXER. This program, while not a part of the data flow, provides a means for further checking the input data and is a record of the information contained on any game base tape.
4. To provide for efficient data handling and communications between the programs of the QUICK system, it is necessary to assign index numbers to various kinds of data in the base. Program INDEXER is designed to perform this important task and is also responsible for forming complex targets and collocation islands. While the program is capable of accepting a game base as created by program QUIKBASE (QUIKDB) in the NMCSSC mode of operation, the modified data base output by BASEMOD (QKMODDB) serves as the primary input. In a like manner, the output game base (INDEXDB) is suitable for input to the Plan Generation and Data Output subsystems. However, in the NMCSSC mode, INDEXDB may be input to program BASEMOD for additional processing and/or modification. An additional output of INDEXER is the SIMTAPE, which contains selected weapon and target data and is subsequently input to the Simulation subsystem.

Appendix A contains a description of the common blocks associated with all utility programs except OUTFILE and RELOADP. For the filehandler (subroutine FILEHNR), only those common blocks used by the calling programs for data transfer are included in this appendix. Appendix B contains a list of QUICK attribute names and their descriptions. Appendix C contains a list of the entry points within the utility programs. Appendix D contains a list of the programs/subroutines which call the general utility routines described in chapter 4.

Users of this manual are encouraged to submit comments or recommendations for changes to improve the manual. Comments should be forwarded to the Commander, National Military Command System Support Center, Defense Communications Agency, Attn: B220, The Pentagon, Washington, D.C. 20301.

CHAPTER 2 QUICK SYSTEM FILEHANDLER

PURPOSE

The QUICK system filehandler is a set of subroutines which provides for data transfer from core memory to either magnetic tape or peripheral disk storage (CDC 814 disk unit). The filehandler provides for fully buffered read/write operations with a maximum of 10 files active at any one time. The filehandler provides input/output (I/O) error checking and correction service to the calling programs. The data are received by the filehandler as a word stream consisting of blocks of words of varying lengths. The word stream is blocked to a convenient record size and output to the peripheral device as a number of physical records. A data dump facility is provided, whereby the filehandler will print the data as the data are being transferred. In addition, a list of files on the disk which are required for the restart capability (programs OUTFILE and RELOADF, discussed elsewhere in this manual) is maintained on the disk by the filehandler.

CONCEPT OF OPERATION

The I/O capabilities of FORTRAN are the most complex and cumbersome feature of the language. The programmer must provide consistency in record lengths and data lists, as well as insert all error detection statements for each data transfer operation. In addition, I/O is very slow compared to the speed of the arithmetic operations. The QUICK system filehandler provides a simplified I/O capability to the programmer.

The input to the filehandler for writing operations consists of a word stream. This stream consists of any number of blocks of words contiguous in the memory. The blocks may be as short as one word or as long as 32,767 words. For reading operations, the output is the same word stream. The blocks transferred from the filehandler, however, do not have to have the same length as the input blocks. The word stream can be segmented by the programmer in any desired way.

The filehandler provides for full double buffering for each file. The two buffers provide for temporary storage. At any one time, one buffer is being used for data transfer to or from core memory; the other buffer is being used for data transfer to or from the peripheral device. When

the core memory transfer buffer is filled (for writing, empty for reading), the status of the I/O operation on the peripheral data transfer buffer is checked. If there are no errors, the roles of the buffers are reversed. In this way, the I/O operations proceed in parallel with the arithmetic operations, which improves the job execution efficiency.

There are three kinds of files available to the user of the CDC 3800 system under the SCOPE PFS Operating System. These types are magnetic tape, permanent disk files, and scratch disk files. Permanent disk files remain intact from job to job. Scratch disk files are destroyed at the end of each job. The QUICK system uses no permanent disk files; all QUICK disk files are of the scratch type. Within that type, however, the QUICK system discriminates between two file types: active and scratch. The active files are required by more than one program in the system. If job termination procedures intervene between executions of two programs which require a specific active file, the restart capability (programs OUTFILE and RELOADF) must be used to save the active file on magnetic tape and restore it at the start of the later job. Scratch files are those files required by only one program of the QUICK system for temporary storage. Therefore, no procedure is necessary to save these files between jobs.

NOTE: In all QUICK documentation, the term scratch file refers to those files which are:

1. Used by only one QUICK program for temporary storage
2. Considered by the SCOPE PFS Operating System to be scratch files.

Although the operating system considers the active files to be scratch files, they are never referred to in that manner in the QUICK documentation.

The QUICK system uses no permanent disk files. The scratch file space on the disk, however, is used and reused by the programs of the QUICK system.

The physical format of any filehandler file consists of a label, data records, padding records, and a terminating end-of-file. The filehandler label is a 32-word physical record containing tape structure information. Table 1 displays the contents of the file label. The label is the first record on the tape following the operating system label. On magnetic tape files, the label is followed by an end-of-file mark, a dummy BCD record, and a second end-of-file mark. These marks and the record are not put on the tape by the filehandler. The operating system places them there after the tape label has been rewritten after the tape is written. (This rewrite is necessary, since the label contains the file length.) On disk files, no end-of-file marks are between the label and the data.

Table 1. Filehandler Label Content

<u>WORD(S)</u>	<u>DESCRIPTION</u>
1	File logical name
2	LNxx5162 - where xx is buffer number used on write operation
3	Number of words written in file
4	Record length (96 words)
5	Run number (currently blank)
6	Date written (mm/dd/yy)
7	Number of words requested for disk file (zero for tape file)
8	Security classification (currently set always to NOT SET)
9	Format identifier (programmer's option)
10	First disk sector address (zero for tape)
11	Number of disk tracks allocated (zero for tape)
12	Number of tracks used (zero for tape)
13	Time of run (hhmmss)
14	Last sector address used (zero for tape)
15-27	Reserved for future use
28-32	User comments

The data records are 96-word physical records on both tape and disk. (The sequential disk read/write routines of the operating system are used for disk I/O.) The first word of each record contains a record number. The first record is numbered one, the second is two, and so on. This numbering is used in the error checking portions of the filehandler to detect the dropping of a record by the operating system and to correct for that error. The remaining 95 words of each record contain the data transferred to or from the program. This is the word stream. (The buffers, of course, are each 96 words long.) The last record in the file is a padding record. It consists of 96 words, each containing the word PADFILE in internal BCD code. This padding file is required by the double buffering operation. In a double buffer mode of input, the peripheral device is reading the record following the record currently being

transferred to core. Thus, the last data record must be followed by a padding record to prevent an end-of-file error when the file is read. On disk files, bad sectors on the disk may prevent the correct output of a record. In order to salvage the file, an error record is written over the bad sector (each sector is 32 words long). This record contains 96 words of the word NODATA in BCD code. When the file is read, all errors which occur in reading a NODATA record are ignored.

NOTE: All filehandler files are written in binary (odd) parity.

Both tape and disk files are terminated with an end-of-file mark. Figure 2 displays the physical format of the files.

The filehandler maintains a directory file on the disk. This file is required by the restart programs, OUTFILE and RELOADF. It has a logical file name of DIRC5162. Its format is one 64-word record. The 64-word record consists of common block /DRC5162/. This file contains the number of active files, their logical file names, their lengths, the number of scratch files, and the length of a scratch file (one million words). This directory governs the files spilled by OUTFILE to the spill tapes. The number of scratch files is saved, so that later programs can reuse space allocated to scratch storage in a previous program.

FUNCTIONAL DESCRIPTION

Initialization and Directory Maintenance

The first program in any job which uses the filehandler should call subroutine ALOCDIR. This routine allocates disk space to the file directory and initializes that directory.* Every program which uses the filehandler must call subroutine INITAPE prior to a call on any other filehandler subroutine (except ALOCDIR).* INITAPE initializes the filehandler variables and reads the file directory. (If INITAPE is called before ALOCDIR, the filehandler corrects the error by calling ALOCDIR on its own.) Subroutine ALOCDIR has no input necessary for its functioning. INITAPE requires two inputs, MYIDENT (in block /MYIDENT/) and NOPRINT (in block /NOPRINT/). MYIDENT is the program name which will be printed on the console typewriter and the standard output. NOPRINT is a print switch. If it has the value one, all file initiations will print a message on the typewriter and standard output. A different value of

*Note that ALOCDIR should be called only once each job. INITAPE should be called once each program.

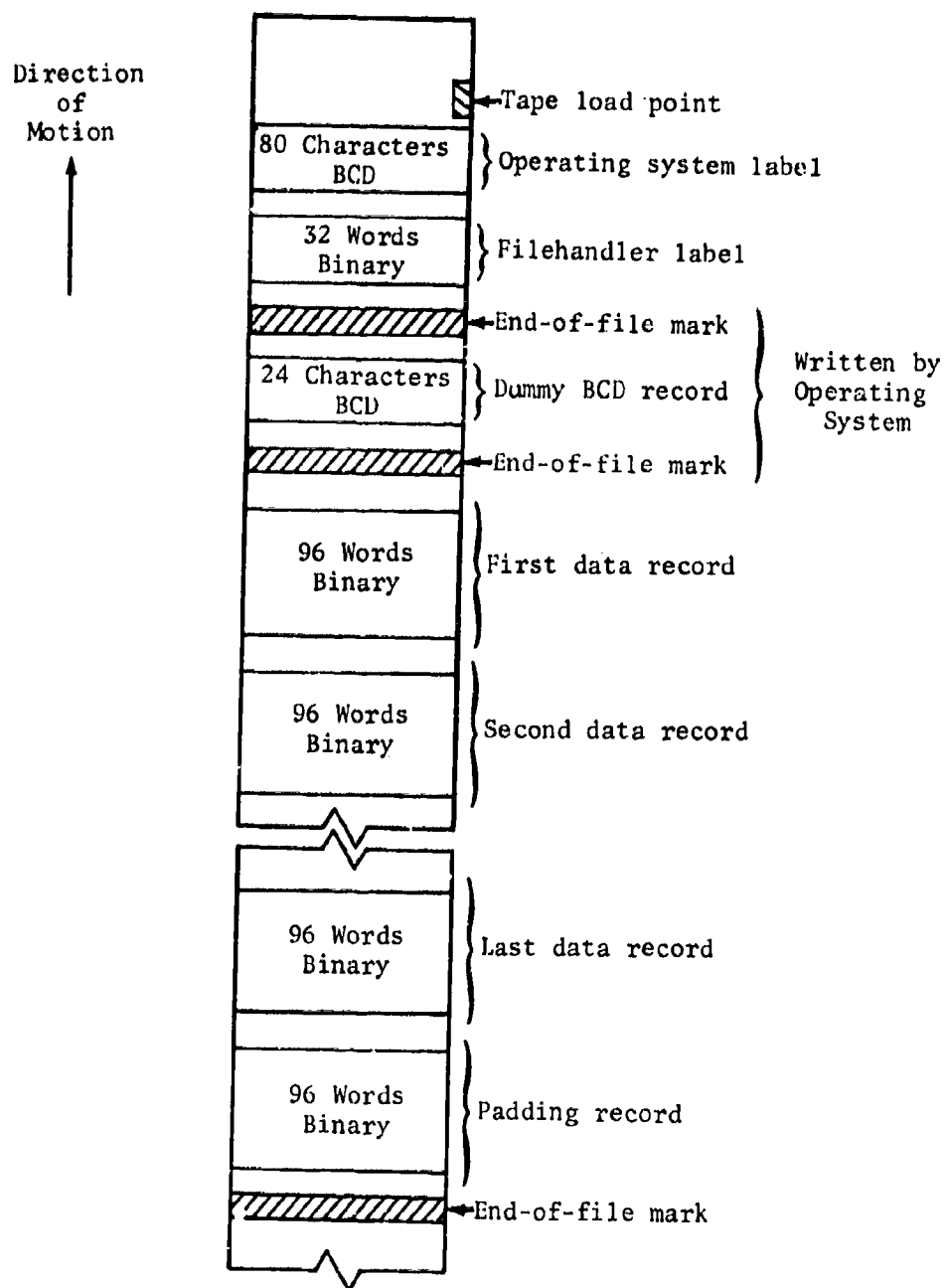


Fig. 2. Physical File Format
Part 1 of 2: Magnetic Tape File

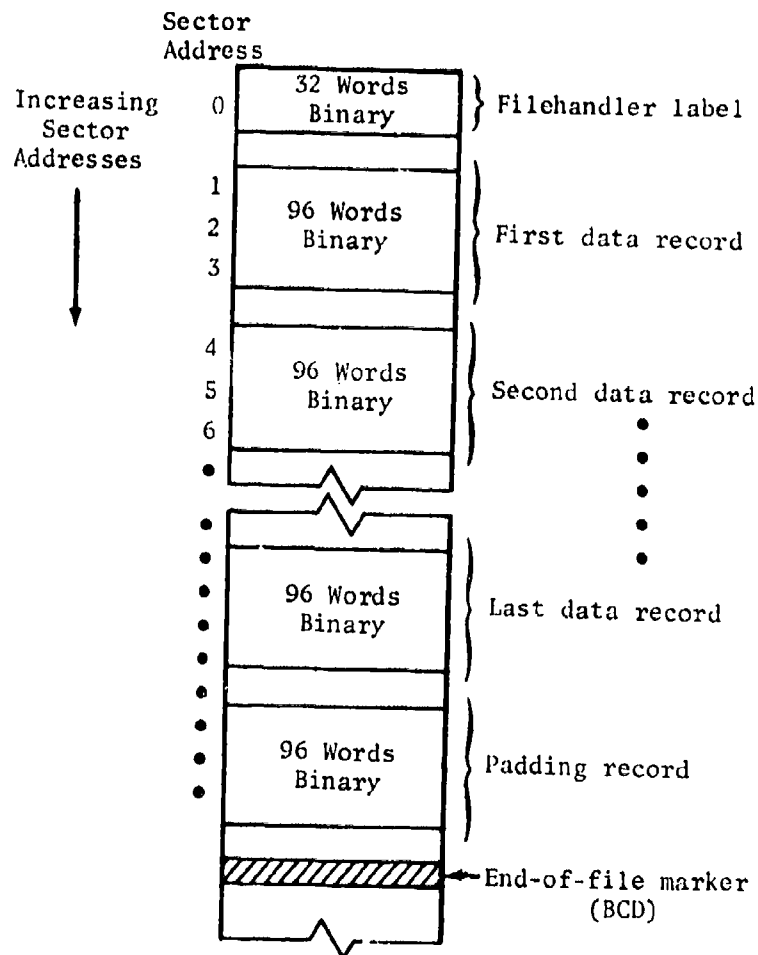


Fig. 2. (cont.)
Part 2 of 2: Disk File

NOPRINT will suppress these messages. Therefore, the standard calling sequence* for INITAPE is:

```
MYIDENT = 8HMYPROGRM
NOPRINT = 1
CALL INITAPE.
```

The value of NOPRINT can be changed during a run to request or delete the messages.

Active files are automatically entered in the directory by subroutine SETWRITE. They are removed by subroutine DEACTIV. This routine removes the file whose name is contained in MYIDENT. Thus, the standard calling sequence for DEACTIV is:

```
MYIDENT = 8HMYFILE
CALL DEACTIV.
```

Output Data Transfer

Four subroutines are used for output data transfer: SETWRITE, WRWORD, WRARRAY, and TERMTAP. Subroutine SETWRITE prepares a file for writing. Buffers are allocated and pointers initialized by this routine. If an active disk file is to be created, the file name is entered in the file directory. Three variables are input to SETWRITE to determine the mode of the file. The three variables are MYIDENT (block /MYIDENT/), ITP (block /ITP/), and MYLENGTH (block /MYLABEL/). The use of these variables is as follows:

1. Tape files

ITP - Is the buffer number and the logical tape unit number. It must lie between 1 and 10 (inclusive).
MYIDENT - Filehandler label name.
MYLENGTH - Not used.

2. Active disk files

ITP - Must be negative. Absolute value is buffer number. It must lie between -10 and -1 (inclusive).
MYIDENT - Logical file name; cannot be SCRATCH. If left blank, a default value of UNAMEDxx is used, where xx is an arbitrary number assigned by the filehandler.
MYLENGTH - Number of words to be allocated to the file. If the file length exceeds this, the operating system will abort the job.

*MYIDENT and NOPRINT cannot be set by data statements. They must be set at execution time.

3. Scratch disk files

- ITP - May be positive or negative. Absolute value is buffer number. Absolute value must lie between 1 and 10.
- MYIDENT - Must be SCRATCH. On the disk, the logical file name will be SCRTCHxx, where xx is a number assigned by the filehandler.
- MYLENGTH - Ignored. Length is one million words.

Once a file has been initialized by a call to SETWRITE, further references to the file can be made through the use of ITP alone. Since each buffer can be used by only one file, the buffer number is a unique file identifier. Note that for tape files, every subsequent initialization of the file must use the same value for ITP, since this is used as the logical unit number. Similarly, for active disk files, subsequent initializations of the file must use the same value for MYIDENT, the logical file name. Scratch disk files must use the same ITP on subsequent initializations.

The standard SETWRITE calling sequence is:

```
ITP = NBUFF
MYIDENT = NAME
MYLENGTH = LONG )   Active disk files only
CALL SETWRITE
```

Note that SETWRITE transfers all the data from common /MYLABEL/ to the file label. If the program fills this block prior to the call on SETWRITE, a call on SETREAD will retrieve this information and pass it to common /FILABEL/.

If one word is to be transferred from core to the file, subroutine WRWORD is used. This routine transfers the value of the variable ITWORD (equivalenced to TWORD) in common /TWORL/ to the output buffer. The standard WRWORD calling sequence is:

```
ITP = NBUFF
ITWORD = IOUT
CALL WRWORD
```

If more than one word is to be transferred, subroutine WRARRAY should be used. This routine transfers a contiguous block of words from core to the output buffer. This routine has two formal parameters. The first is the first word to be transferred; the second is the number of words to be transferred. The standard WRARRAY calling sequence is:

```
ITP = NBUFF
CALL WRARRAY (ORIGIN, LENGTH)
```

A write file is terminated by a call on subroutine TERMtAP. This routine clears the buffers, adds the padding record, rewrites the label, and

resets the buffer pointers. The standard calling sequence is:

```
ITP = NBUFF  
CALL TERMTAP
```

Input Data Transfer

There are four routines used for input data transfer, SETREAD, RDWORD, RDARRAY, and TERMTAP.

A file is initialized for reading by a call on subroutine SETREAD. This subroutine allocates the buffers and initiates reading from the file into the buffers. Two variables are input to SETREAD: ITP (block /ITP/) and MYIDENT (block /MYIDENT/). The value of these variables determines the type of file which is initialized by SETREAD.

1. Tape files
ITP - Must be positive and in the range 1 to 10 (inclusive). This is the logical unit number and buffer number.
MYIDENT - Filehandler label name.
2. Active disk files
ITP - Must be negative. Absolute value is buffer number. Range is -10 to -1.
MYIDENT - Logical file name. Cannot be SCRATCH nor can the first six characters be SCRTCH.
3. Scratch disk files
ITP - May be positive or negative. Absolute value is buffer number. Absolute value range is 1 to 10.
MYIDENT - Must be SCRATCH or the first six characters must be SCRTCH.

For scratch files, SETREAD reads the scratch file created using the same buffer number requested in ITP. Common /FILABEL/ is filled from the filehandler label. SETREAD reads the first buffer of data and initiates the reading of the second buffer before returning control. The standard SETREAD calling sequence is:

```
ITP = NBUFF  
MYIDENT = NAME  
CALL SETREAD
```

After the file is initialized, it may be referred to with only the buffer number. Since only one file can be attached to a single buffer, the buffer number ITP is a unique file identifier after file initialization.

To read one word, subroutine RDWORD is used. The next word in the stream (buffer) is transferred to variable ITWORD in common /TWORD/. (ITWORD is usually equivalenced to TWORD for variable type compatability.) The standard RDWORD calling sequence is:

```
ITP = NBUFF
CALL RDWORD
IN = ITWORD
```

If more than one word is to be transferred, subroutine RDARRAY should be used. This routine transfers a block of words from the buffer(s) to a contiguous storage area. There are two formal parameters for subroutine RDARRAY. The first is the first word of core into which the data are to be transferred; the second parameter is the block length. The standard RDARRAY calling sequence is:

```
ITP = NBUFF
CALL RDARRAY (DESTIN, LENGTH)
```

A read file is terminated by a call on subroutine TERMTAP. For this type of file (read), TERMTAP rewinds the file if on tape or positions the sector pointer to the first sector (sector 0).

Dump Facility

The program can request a print of the data being transferred through the filehandler. The array IFTPRNT in common /IFTPRNT/ is used for this purpose. There is one position in the array for each buffer. The value of IFTPRNT is used as follows: if zero or negative, no print is produced; if positive, data are printed. On calls to RDWORD or WRWORD, only the one word transferred is printed. On calls to RDARRAY or WRARRAY, the value of IFTPRNT (ITP) is interpreted as the number of words from each end of the block transferred to be printed. For example, if IFTPRNT(3)=20 and the sequence of code is as follows:

```
ITP = 3
CALL RDARRAY (ARRAY, 100)
```

the values of ARRAY(1) through ARRAY(20) and ARRAY(81) through ARRAY(100) will be printed.

Error Detection and Recovery

The filehandler performs two types of error detection and recovery. It checks for program errors in filehandler use, such as attempting to read from a write file. The recovery procedures in these cases are either warning messages, simple corrective actions (such as calling TERMTAP and SETREAD in the above case), or abort with a memory dump. The other

error checks are concerned with data transmission errors. End-of-file, end-of-tape, or parity errors are all detected, and some recovery is attempted. After an error has occurred, the filehandler will first repeat the operation several (five) times to determine if the error was of a transient nature. If the error persists, various recovery procedures are performed as described below.

1. Tape Output Error
Subroutine ERAZE is called to erase six inches of tape and skip over the bad spot.
2. Tape Input Error
The tape is skipped forward to the first record following the troubled record. If this new record has the correct number, recovery is successful. If the record number is wrong, there is no way to recover, and the job aborts. Note that a tape produced with output errors as corrected by the filehandler can be read back by the filehandler.
3. Disk Output Error
A variable LOCNOW in block /FILE/ contains the sector address of the first sector of the troubled record. The file is "backspaced" by positioning the file at that point. A dummy record containing 96 words of the word NODATA is written over the troubled sectors. The data record is then written starting with the next group of three sectors.
4. Disk Input Error
If the troubled record contains the NODATA value, the file is skipped forward to the next data record. If this record has the correct record number, recovery is successful. Otherwise, the job aborts. In addition, if the original error did not occur in a NODATA record, the job aborts.

COMMON BLOCKS

Table 2 lists the filehandler common blocks. All subroutines of the filehandler use all these blocks for their operation. The following blocks are also used by the calling programs for data transfer:

ITP	}	All programs
MYIDENT		
NOPRINT		
TWORD		
MYLABEL		
FILABEL		
IFTPRNT		
TODAY		(Program PLANSET)
DRCS162		(Programs OUTFILE, RELOADF, ALOC)
INTFILE		(Program ALOC).

The following blocks are used internally by the filehandler:

FILE
22642626
FILELAB
FORFLS
LOCVAR.

Table 2. Filehandler Common Blocks
(Sheet 1 of 4)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY *</u>	<u>DESCRIPTION</u>
DRC5162	LENLIST	Length of file directory
	NOWACTV	Number of active disk files
	LFNAME(30)	Logical file names of active files
	NUMFWDS(30)	Number of words written on each active file
	NINTRNL	Number of scratch disk files allocated to disk
	LINTRNL	Length (words) of scratch disk files
INTFILE	INTIDNT	First six characters of scratch file name (=SCRATCH)
	LINTFL	Length (words) of scratch disk files
	NOWALOC	Number of scratch files used by current program
	MYITP(10)	Buffer number used for each scratch file written in current program
FILE	ACTIVE(10)	Logical array; TRUE if buffer in use
	READST(10)	Logical array; TRUE if file is in read status
	IAUX(10)	First word in auxiliary buffer for each file (this is device transfer buffer)
	IBEG(10)	First word in core transfer buffer for each file
	IDISKFL(10)	Negative for disk files
	IEND(10)	Last word available in data transfer buffer
	IPT(10)	Pointer to next word to be transferred
	NAME(10)	Logical file name
	NUMRECW(10)	Number of words written on file

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

Table 2. (cont.)
(Sheet 2 of 4)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
FILE (cont.)	NEWT(10)	Pointer to last good buffer transferred to/from file
	LOCNOW(10)	Sector address after last good disk read/write
22642626*	IBUFF(1920)	Filehandler buffers
FILELAB	LNDIR	Length of file label
	NDIROUT(32)	File label (see table)
	NDUMMY(33)	Label temporary storage
FORFLS	IBADGE	=5162
	LNGREC	Physical record length (=96)
	LUN	=LN
	MODE	Data parity (=1)
	NPAUL	=DIRC (for file directory name)
	NSECPRT	Number of sectors per track on CDC 814 disk (=32)
	NSECTOR	Number of words per sector on CDC 814 disk (=32)
ITP	ITP	Filehandler buffer number
MYIDENT	MYIDENT	File name
TWORD	ITWORD	Single word transfer medium

*This number is the word BUFF in internal BCD code. The block is a
numbered common block to take advantage of the loader overlap feature
of the SCOPE PFS loader.

Table 2. (cont.)
(Sheet 3 of 4)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
FILABEL	INIDENT	Incoming name (word 1 of label)
	INRUNNO	Incoming run number (word 5 of label)
	INDATE	Incoming date (word 6 of label)
	INFORM	Incoming format (word 9 of label)
	INSECR	Incoming security (word 8 of label)
	INTIME	Incoming time (word 13 of label)
	INLNTH	Incoming length (word 3 of label)
	INCOMM(5)	Incoming comments (words 28-32 of label)
IFTPRNT	IFTPRNT(10)	Debug print switches
MYLABEL	MYFORM	Outgoing format (word 9 of label)
	MYSECR	Outgoing security (word 8 of label)
	MYLNTH	Number of words requested for file (word 7 of label)
	MYCOMM(5)	Outgoing user comments (words 28-32 of label)
NOPRINT	NOPRINT	File initiation print switch
TODAY	NOWRUNO	Current run number (word 5 of label)
	NOWDATE	Current date (word 6 of label)
	NOWTIME	Current time (word 13 of label)
LOCVAR	IPOINT	Pointer to next word transferred
	MESS(8)	Abort/warning message
	MFAKE	Used for blank file names (UNAM:D)
	NDIR	Directory length
	NTX	Error code
	NT	Logical unit number for disk

Table 2. (cont.)
(Sheet 4 of 4)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
LOCVAR (cont.)	NTR	Number of tracks allocated to disk file
	NVERS	Blank
	NWAIT	Disk I/O completion code
	NWRDSAL	Number of words allocated to disk file
	NTEMP	Entry point name
	MTEMP	TAPE or DISK: file type identifier
	INITFST	INITAPE call check variable
	KREC(10)	Record counter for each file
	NULL(96)	Storage for NODATA record
	NODATA	=NODATA
	NTIMES	Error correction attempt counter

SUBROUTINE FILEHNR

PURPOSE:

This FORTRAN subprogram contains entry points ALOCDIR, INITAPE, DEACTIV, PRNTLAB, SETWRITE, RDWORD, and WRWORD. The purposes of these entries are as follows:

ALOCDIR allocates the disk file directory to the disk and initializes that directory

INITAPE initializes the filehandler variables

DEACTIV removes disk logical file names from the active list in the file directory

PRNTLAB prints file labels

SETWRITE initializes a file for writing; the file is activated and the buffers are allocated

RDWORD transfers one word from the buffer to common /TWORD/

WRWORD transfers one word from common /TWORD/ to the buffer

ENTRY POINTS:

FILEHNR (not used), ALOCDIR, INITAPE (INITAP), DEACTIV, PRNTLAB, SETWRITE (SETWRIT), RDWORD, WRWORD

FORMAL PARAMETERS:

None

COMMON BLOCKS:

DRC5162, INTFILE, FILE, 22642626, FILELAB, FORPLS, ITP, MYIDENT, TWORD, FILABEL, IFTPRNT, MYLABEL, NOPRINT, TODAY, LOCVAR

SUBROUTINES CALLED:

GETDATE, TIMEDAY, ERAZE, RDARRAY, WRARRAY, ABORT, WARNING, Disk I/O subroutines of SCOPE PFS Operating System (ALLOCATE, LOCATE, SEEK, LSTATUS, DREAD, DWRITE)

CALLED BY:

All programs of the QUICK system

Method:

Entry ALOCDIR: This entry uses operating system routine ALLOCATE to allocate disk storage for the active disk file directory. The directory is initialized to show no active files other than itself. ALOCDIR should be called only once, at the beginning of each job. The directory name is DIRC5162. If the directory cannot be written without errors, the job will abort.

Entry INITAPE: This entry initializes filehandler variables. It should be called once per program, at the beginning of execution. INITAPE clears the buffers and retrieves the current date (subroutine GETDATE) and time (subroutine TIMEDAY). The variable MYIDENT is assumed to be the name of the calling program. INITAPE prints a program initiation message on the console typewriter and also on the standard output, if requested. If ALOCDIR has not yet been called in the job when INITAPE is called, INITAPE transfers control to ALOCDIR before exiting. Otherwise, the file directory is read into common block /DRC5162/.

Entry DEACTIV: This entry removes a file name from the list of active disk files. The name to be removed is placed in variable MYIDENT (block /MYIDENT/) by the calling program. The file directory DIRC5162 cannot be deactivated by DEACTIV. Since scratch files are not entered in the directory, any attempt to deactivate a scratch file is ignored. If DEACTIV cannot find the requested file in the active list, the routine returns control to the calling program without further processing. If the file is found in the list, the name and length are removed from the list and the list compressed.

Entry PRNTLAB: This entry prints file labels. The input is the buffer number ITP and the file name MYIDENT of the file whose label is to be printed. Since PRNTLAB must move the file to retrieve the label, a call on PRNTLAB while the file is being read or written (i.e., active) will be considered an error. Since PRNTLAB is a convenience utility routine, all errors, including I/O errors, in its processing terminate PRNTLAB processing and return to the calling program after printing a warning message. The format of the file label print is described in the User's Manual. The print is also produced on every call to SETREAD if the program requests it.

Entry SETWRITE: This entry initializes a file using buffer ITP and name MYIDENT. (The Functional Description section of this chapter describes the interpretation of these variables.) SETWRITE allocates two buffers, each 96 words in length, to the file. For all scratch files, SETWRITE generates a logical file name. The scratch files are named SCRTCHyy, where yy is a number assigned by SETWRITE. The first scratch file is SCRTCH 1; the second SCRTCH 2; and so on. If the program calls SETWRITE with a blank file name, SETWRITE assigns a name UNAMEDzz, where zz is a

number assigned in the same fashion as yy for scratch files. Various procedural error checks are performed to see that the file should be written on. For example, a call to SETWRITE on a file currently in read status will abort the job. For disk files, variable MYLENGTH in /MYLABEL/ is used to determine the number of tracks on the disk to allocate for the file. MYLENGTH is the maximum number of words expected on the file. If it is less than one or greater than one million, it is set to one million (977 tracks). Scratch files are always given lengths of one million words, since the number of words on the file changes with each use.

SETWRITE also fills the filehandler label from block /MYLABEL/ and internal filehandler variables. The label is then written on the file. For tape files, subroutine ERAZE is then called three times on the file. This routine erases six inches of tape. (An end-of-file mark is approximately six inches long.) This 18-inch blank stretch of tape is required by the operating system. When TERMTAP is called on this file, the label will be rewritten so as to contain the number of words written on the tape. The tape will then be rewound. If a program writes on a tape and then rewinds the tape, the operating system places on the tape, following the last write, an end-of-file mark, a dummy BCD record, and a second end-of-file mark. The dummy record is 24 characters long, of which the first seven are EOSQUAT and the remaining 17 are undetermined. The erasure of 18 inches of tape between the label and the first data record allows space for placement of these marks written by the operating system.

Entry RDWORD: This entry transfers the next word in the buffer for file ITP to the variable comprising common block /TWORD/. ITP is the buffer number for the file. Entry RDWORD uses subroutine RDARRAY to read the word. This procedure is used so that the error correction procedures coded in subroutine RDARRAY do not have to be repeated in RDWORD. Note that if the data debug dump facility of the filehandler is activated (i.e., IFTPRNT(ITP)>0), then it is deactivated before RDARRAY is called. The dump facility produces different print formats for calls on RDWORD/WRWORD and RDARRAY/WRARRAY. Thus, if RDWORD calls RDARRAY, the facility must be off during the data transfer. When control returns to RDWORD, the value of IFTPRNT is restored. If the original value is greater than zero, then the RDWORD format for the debug print is produced.

Entry WRWORD: This routine is very similar to RDWORD. A call on WRWORD transfers the data in common block /TWORD/ (one word) to the next location in the buffer for file ITP. Subroutine WRARRAY is used for the transfer in the same fashion as RDARRAY is used by RDWORD. The use of the filehandler dump facility in WRWORD is identical to that in RDWORD.

The flow of operations in FILEHNR is shown in figure 3. The flowchart consists of seven parts. Parts I to V pertain to entries ALOCDIR, DEACTIV, INITAPE, PRNTLAB, SETWRITE, respectively. Part VI shows entries WRWORD and RDWORD while part VII illustrates the error processing operations.

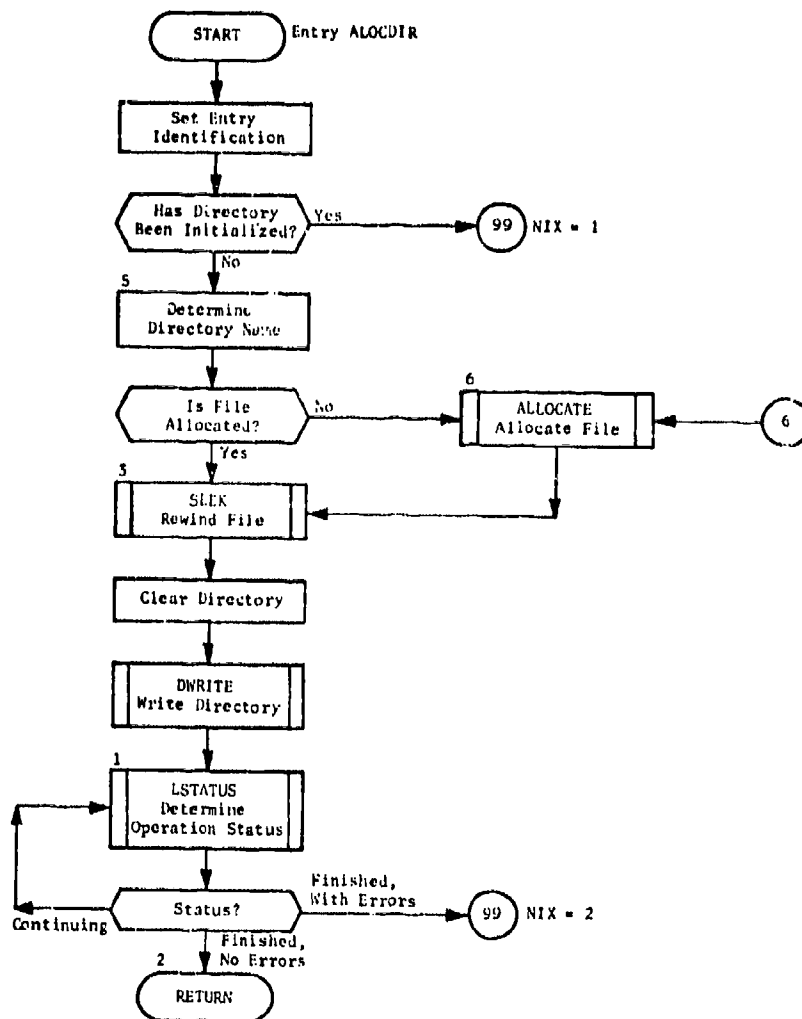


Fig. 3. Subroutine FILEHNR
Part I: Entry ALOCDIR
25

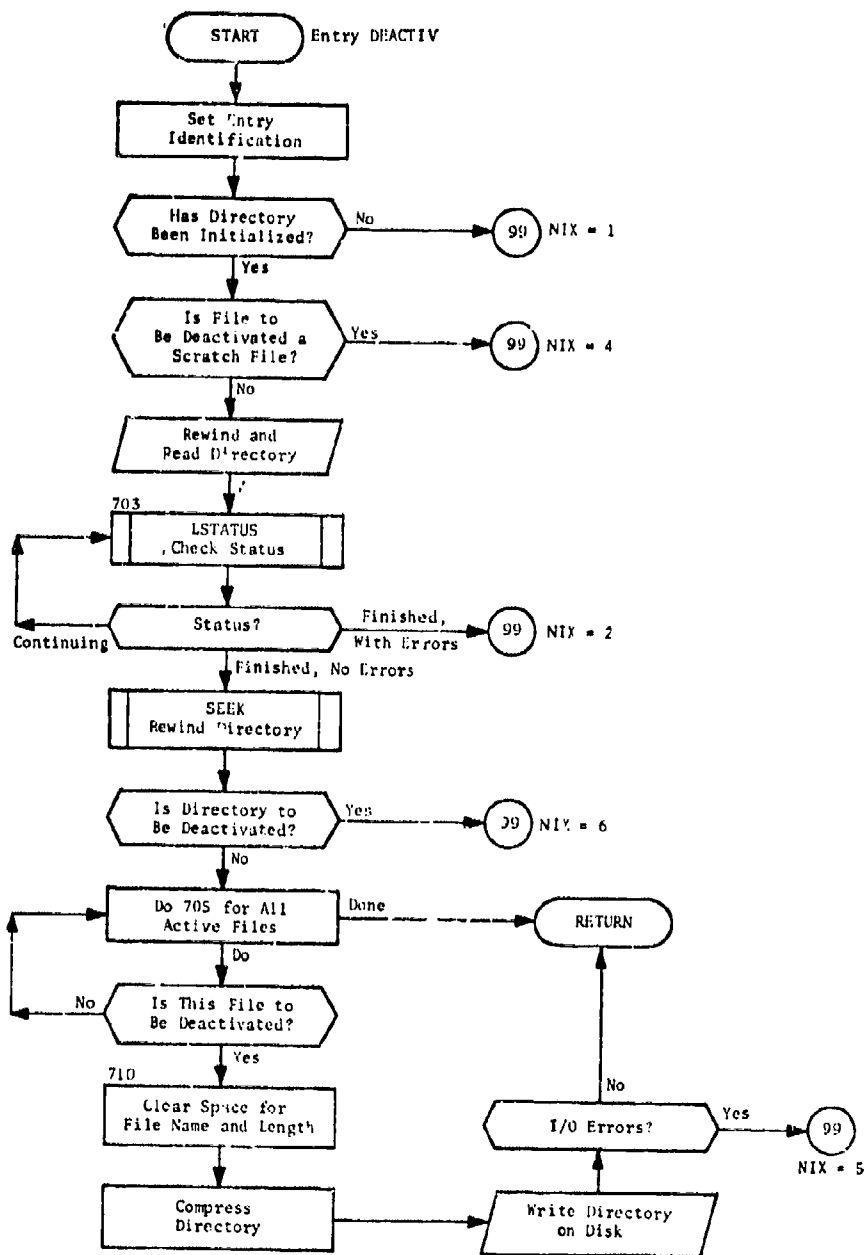


Fig. 3. (cont.)
Part II: Entry DEACTIV

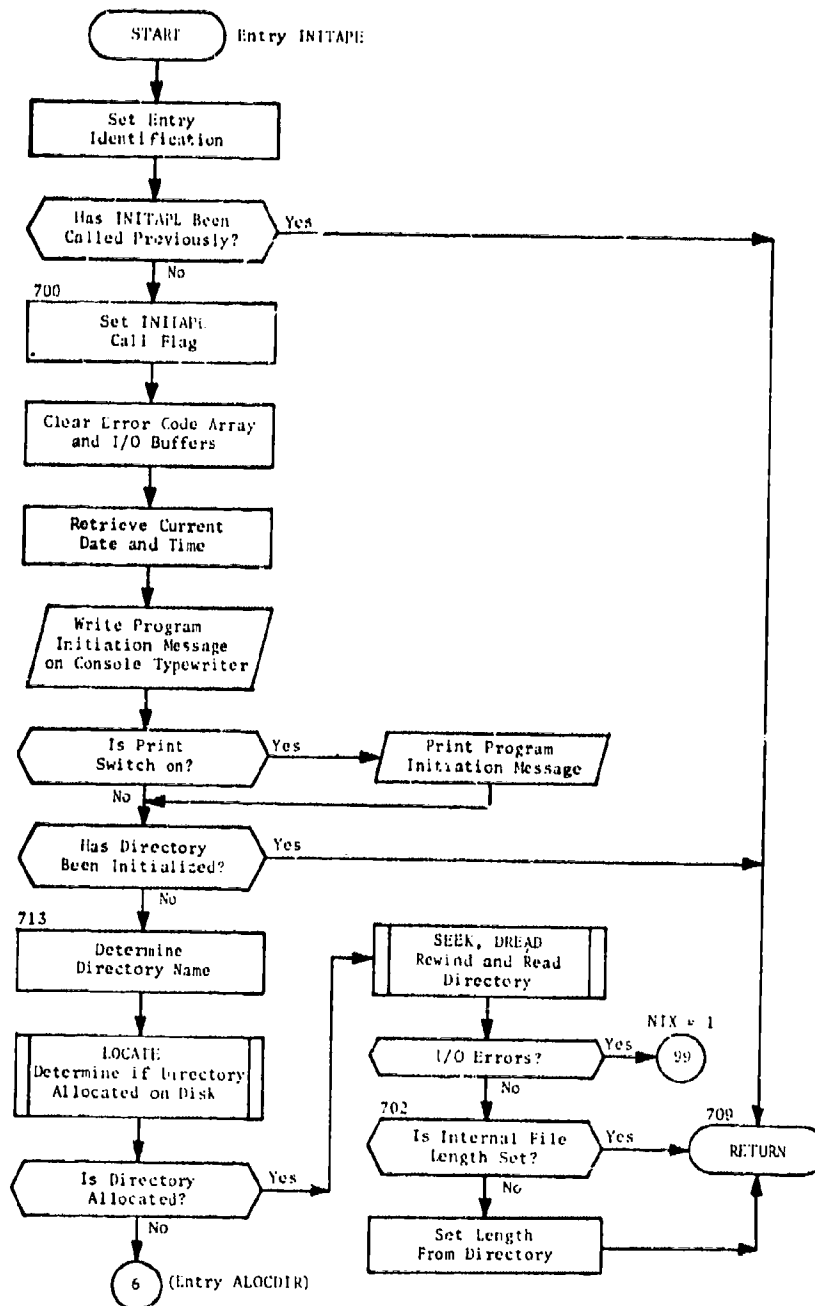


Fig. 3. (cont.)
Part III: Entry INITAPE

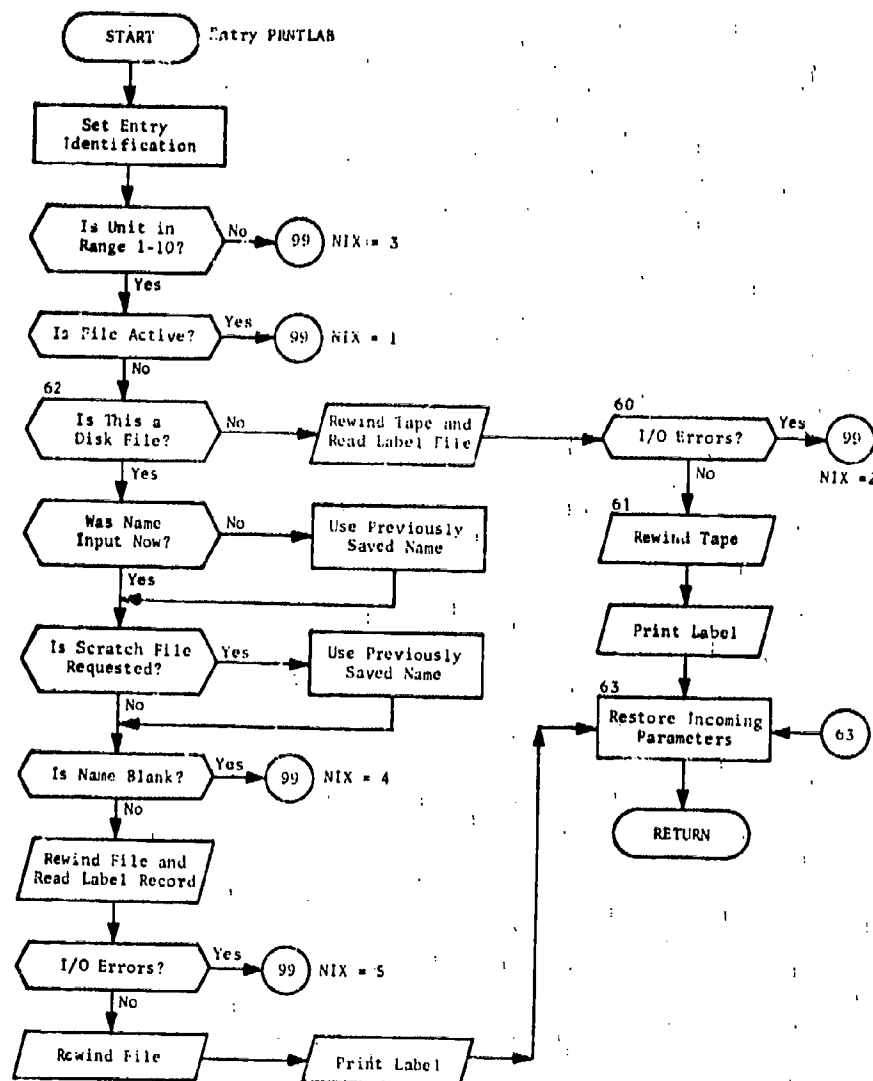


Fig. 3. (cont.)
Part IV: Entry PRNTLAB

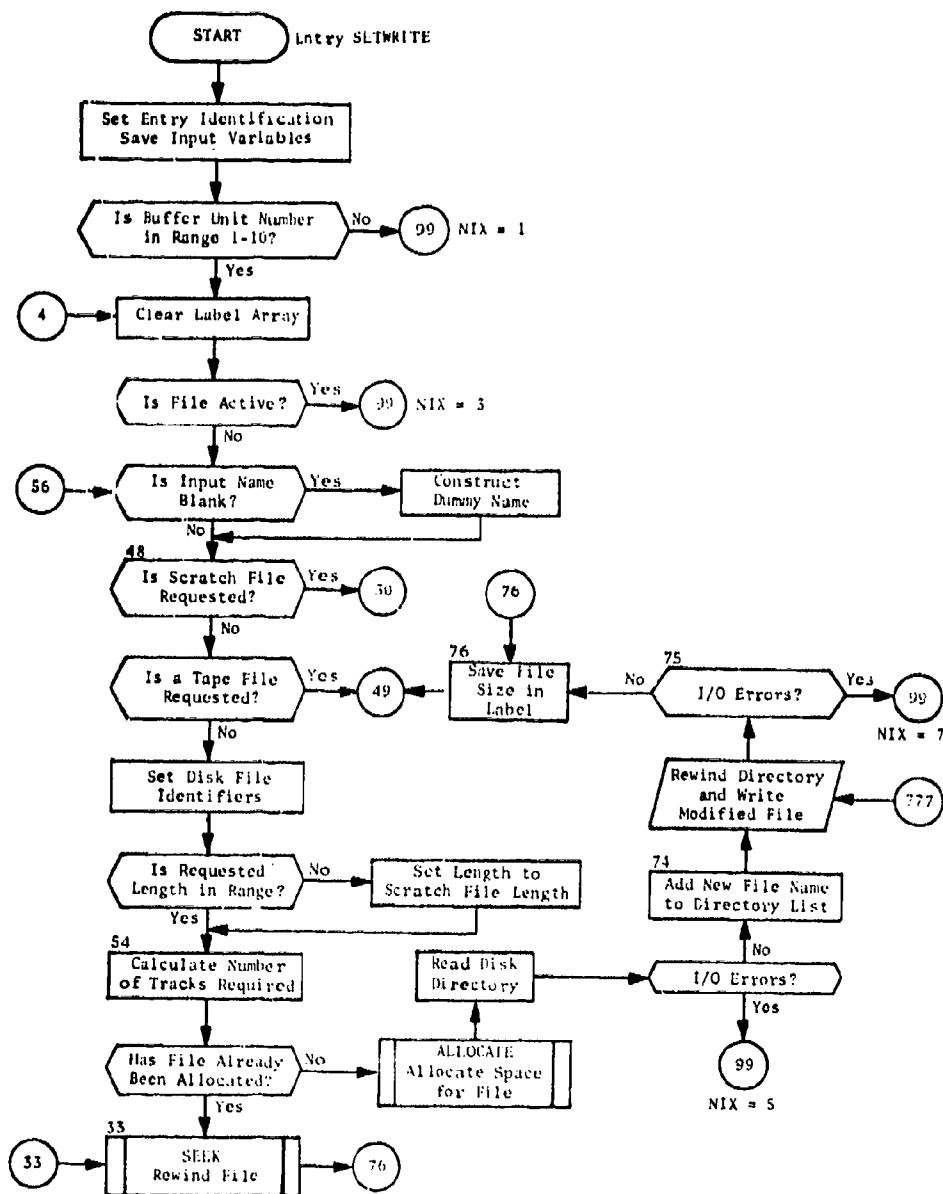


Fig. 3. (cont.)
Part V: Entry SETWRITE
(Sheet 1 of 3)

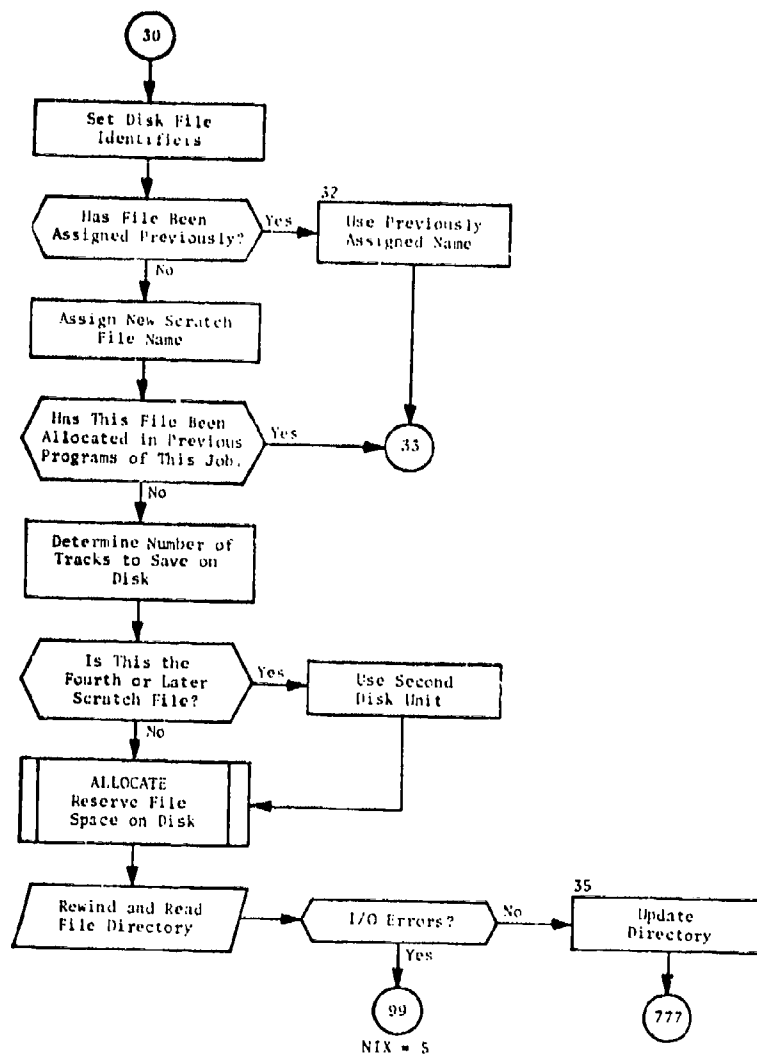


Fig. 3. (cont.)
 Part V: (cont.)
 (Sheet 2 of 3)

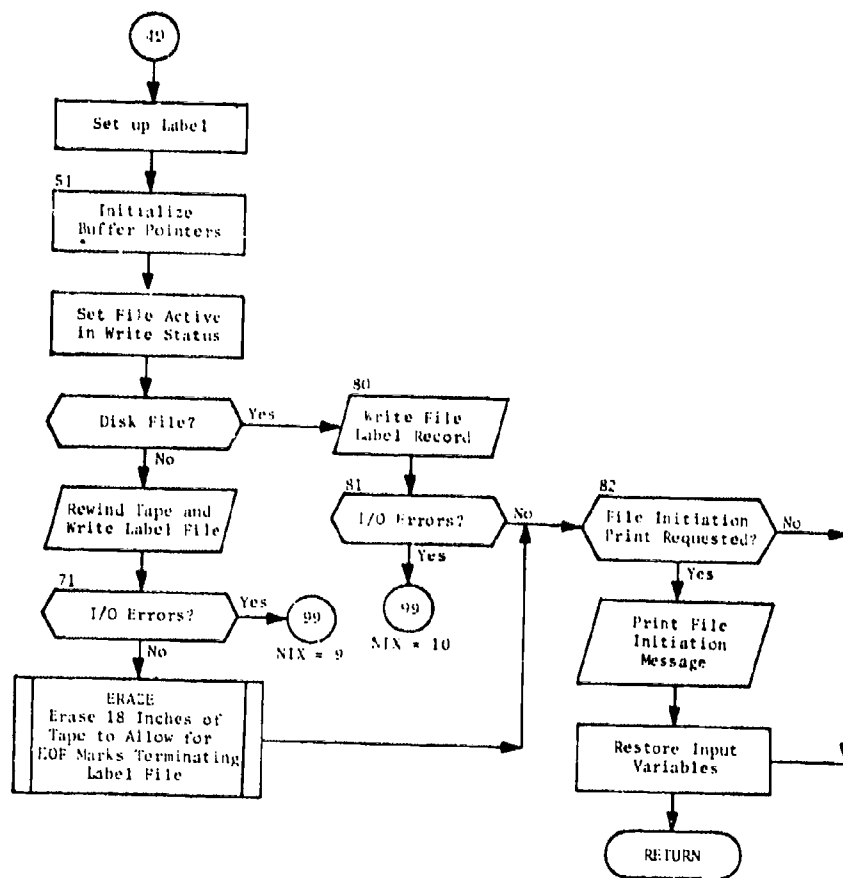


Fig. 3. (cont.)
 Part V: (cont.)
 (Sheet 3 of 3)

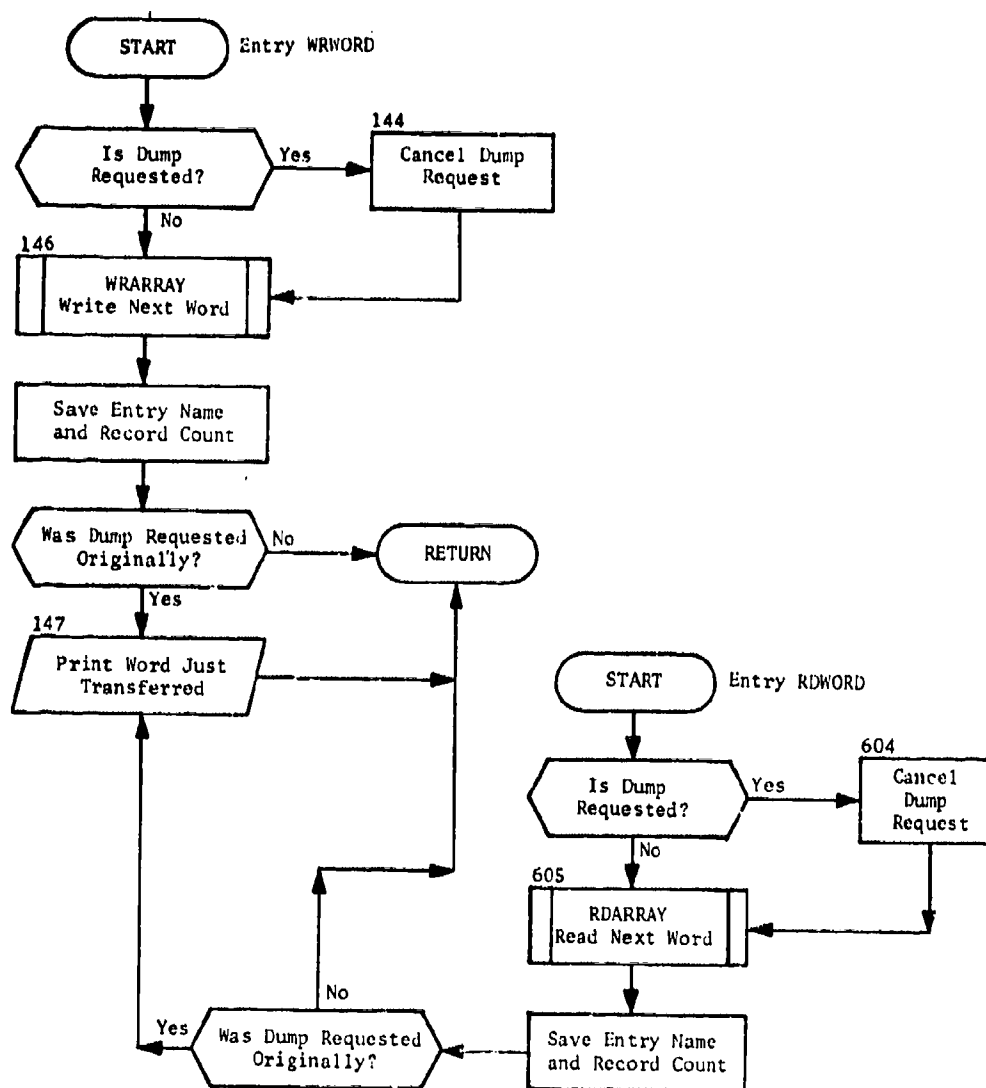


Fig. 3. (cont.)
Part VI: Entry WRWORD & RDWORD

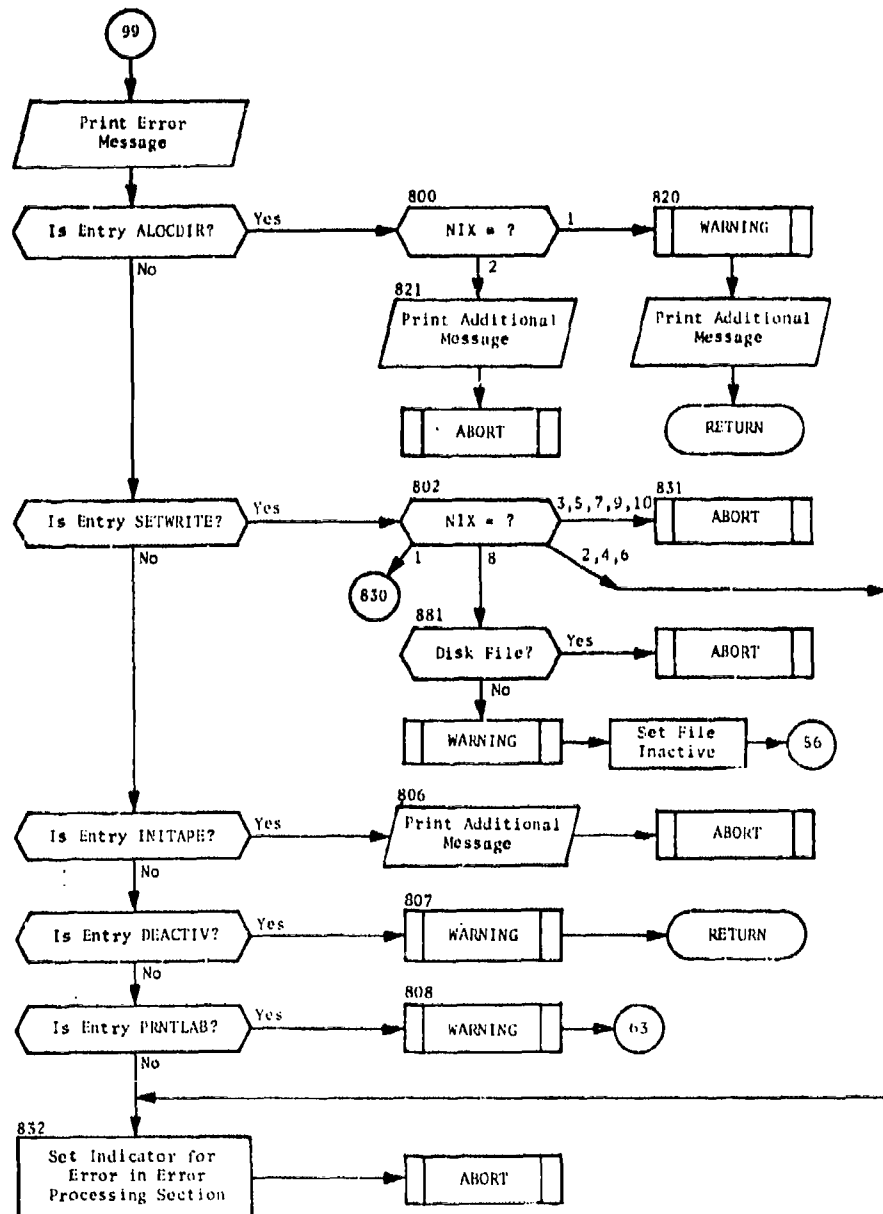


Fig. 3. (cont.)
Part VII: Error Processing
(Sheet 1 of 2)

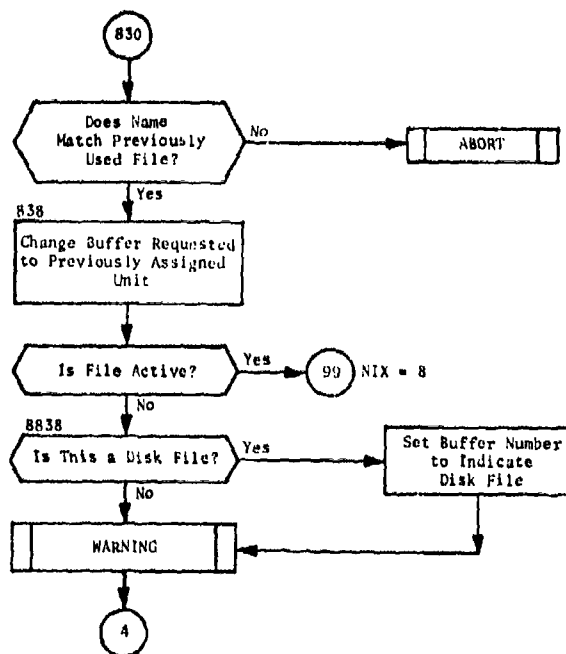


Fig. 3. (cont.)
 Part VII: (cont.)
 (Sheet 2 of 2)

SUBROUTINE RDARRAY

PURPOSE: This FORTRAN subprogram contains entry points RDARRAY and WRARRAY. Their purpose is to transfer blocks of data between core storage and the file buffers and also between the file buffers and the peripheral I/O devices. Entry RDARRAY is used for transfers from the files to core storage. Entry WRARRAY is used for transfers from core storage to the files.

ENTRY POINTS: RDARRAY, WRARRAY

FORMAL PARAMETERS: NAR - The first word in core storage to be used in the transfer
NUM - The number of contiguous words to be transferred

COMMON BLOCKS: DRC5162, INTFILE, FILE, 22642626, FILELAB, FORFLS, ITP, MYIDENT, TWORD, FILABEL, IFTPRNT, MYLABEL, NOPRINT, TODAY, LOCVAR

SUBROUTINES CALLED: ABORT, WARNING, LOCF, ERAZE, SETREAD, SCOPE PFS Disk I/O subroutines (SEEK, LOCATE, LSTATUS, DREAD, DWRITE), SCOPE PFS Tape I/O subroutine, LENGTHF

CALLED BY: All programs of the QUICK system

Method

The formal parameters specify the block of core storage to be used in the transfer. The parameter NAR is the first word of the block. This parameter is given a length of one in a DIMENSION statement. This procedure allows the subroutine to treat the storage block as an array of any length. The parameter NUM is the number of words to be transferred.

Figure 4 shows the processing operation performed by subroutine RDARRAY(WRARRAY). The flowchart is presented in five parts. Parts I and II show the initial operations for entries RDARRAY and WRARRAY, respectively. Part III shows exit processing procedures and is common to both entries. Error processing operations are shown in Part IV for entry RDARRAY and in Part V for WRARRAY.

Entry RDARRAY: This entry is used to read data from a file into core storage. After checking to determine if the file is active and in read status, RDARRAY determines if enough words remain in the current buffer to fulfill the request for NUM words (statement 605). If so, the first NUM elements of the destination array NAR are filled with the next NUM words in the buffer (statement 610). The buffer pointers are updated and control passes to the data dump section (statement 630). If part or all of another buffer is needed to fill the request, the routine empties the current buffer into the destination array NAR (statement 640). The other buffer was being filled by a read operation initiated by a BUFFER IN operation in either SETREAD or RDARRAY. RDARRAY checks the status of this operation (statement 607). If there were errors, control passes to the error correction section (statements 199, 99). If the read operation was completed successfully, a new read operation is initiated to fill the buffer just emptied. The buffer pointers are then modified to reverse the buffer roles. Control then returns to statement 605 to determine if the remainder of the requested block of data can be filled from the current buffer. This process continues until all words have been transferred. For disk files, "dummy" records may be encountered amidst the data records. These dummy records contain the word NODATA in all 96 words of the record. These records are written to space the disk file over sectors in which the I/O subroutines are encountering errors. RDARRAY ignores dummy records. When one is encountered, the next record is read and so on until the next data record is encountered.

In RDARRAY, two entries to the error processing section are used, statements 99 and 199. Statement 199 merely increments the error code NIX by one before transferring control to statement 99. This procedure allows the error routine to differentiate between parity errors and unexpected end-of-file marks. The end-of-file error codes are one greater than the parity error codes. The error correction procedures for entry RDARRAY begin at statement 805. For unexpected end-of-file marks, the job aborts (statement 842). For parity errors, the processing of tape and disk differs. For tape errors, operating system function LENGTHF is called (statement 870) to determine the length of the record with bad parity. If the length is correct (i.e., 96 words), there can be no recovery, and the job aborts. If the record is short, it may be an error record written previously. In this case, the next data record is read. If it has the correct record number, recovery is successful. Otherwise, the job aborts via a transfer to statement 99 with NIX=7 (from a statement following statement 624). For disk errors, five attempts are made to reread the data. If these attempts fail, the job aborts.

Entry WRARRAY: The processing in this entry closely parallels the operation of RDARRAY. In this case, however, NAR is the origin array. Words are transferred from NAR to the buffer and from the other buffer to the file. Statement 146 parallels the operation of statement 605 in RDARRAY in determining the number of words to be transferred from the current buffer. If the current request for NUM words will not fill the

buffer, statement 710 performs the complete transfer. Otherwise, statement 740 transfers full buffer loads from core to the buffers, until the number of words remaining to be transferred is less than the full buffer size. There is one major difference between the logical flow of RDARRAY and WRARRAY. Subroutine SETREAD assigns both buffers since they will both be used immediately. Subroutine SETWRITE, however, allocates only one buffer, the first to be filled. Therefore, the checking of the status of I/O operations starts only after the first write by subroutine WRARRAY. In addition, no distinction is made between write parity and write end of "reel" errors in WRARRAY. There is no end of "reel" error for disk files, and end-of-reel errors for tape are irrecoverable. Therefore, WRARRAY uses only statement 99 as entry to the error processing section.

The error correction processing for WRARRAY begins at statement 804. For tape write parity errors, the tape is backspaced over the problem area. Subroutine ERAZE is called to erase over the bad spot. The record is then rewritten. If this procedure fails after five attempts (and 30 inches of tape), the job aborts. For disk files a similar procedure is followed. The file cannot be "backspaced" in the usual sense. Therefore, a pointer array LOCNOW is saved in common /FILE/. For each file on disk, LOCNOW(ITP) contains the sector address of the sector following the last good I/O operation. (Operating system routine LOCATE is used to determine this information.) When an error occurs, operating system routine SEEK is used to reposition the disk file to this sector. This is the disk equivalent to backspacing. Then the dummy NODATA record is written. If this write operation encounters errors (NIX=5), it is repeated five times. After five attempts or a successful write, the data record is written starting with the next sector available after the NODATA record.

Data Dump Processing: Both RDARRAY and WRARRAY share the data dump code starting at statement 630. The complex DO loop between statements 3 and 11 is required by the format of the data dump. The value of IFTPRNT(ITP) is interpreted as the number of words to be printed from each end of the array NAR. The print is therefore divided into halves. The word counters for the halves are checked in this loop to determine when to print the data. Utility subroutine LOCF is used to retrieve the address of NAR which is printed in the dump heading.

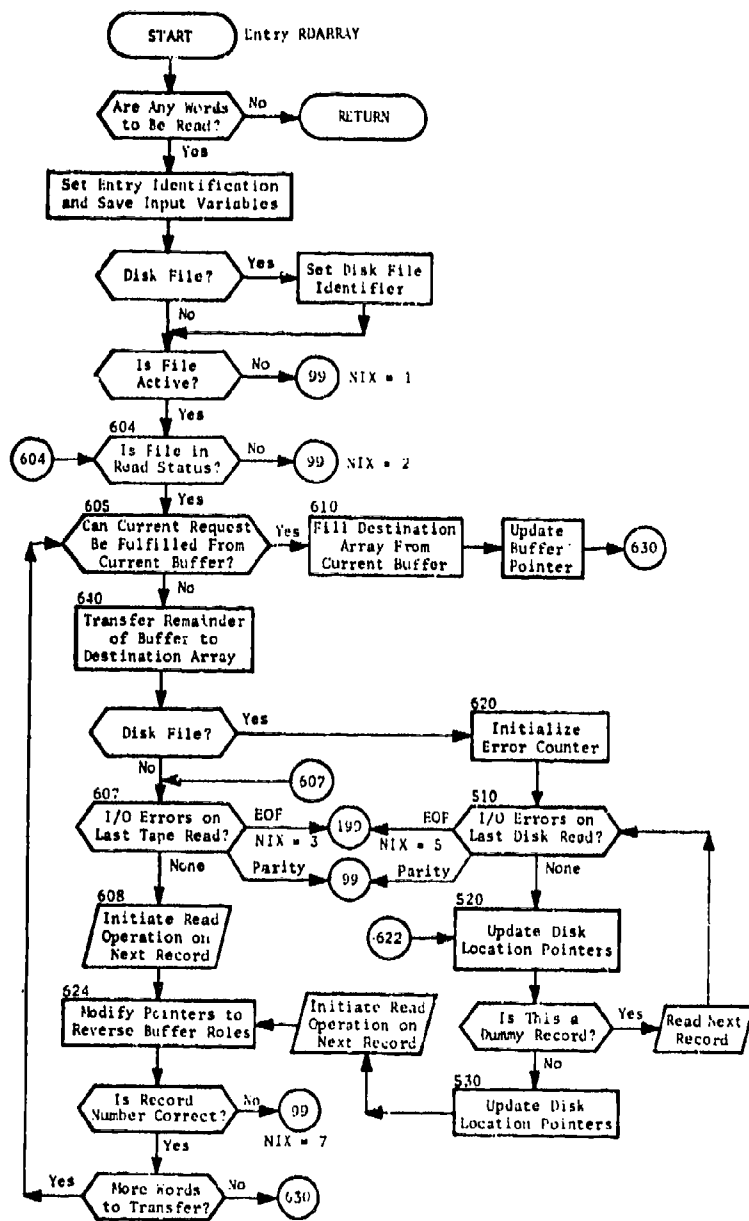
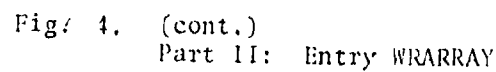


Fig. 4. Subroutine RDARRAY
Part I: Entry RDARRAY



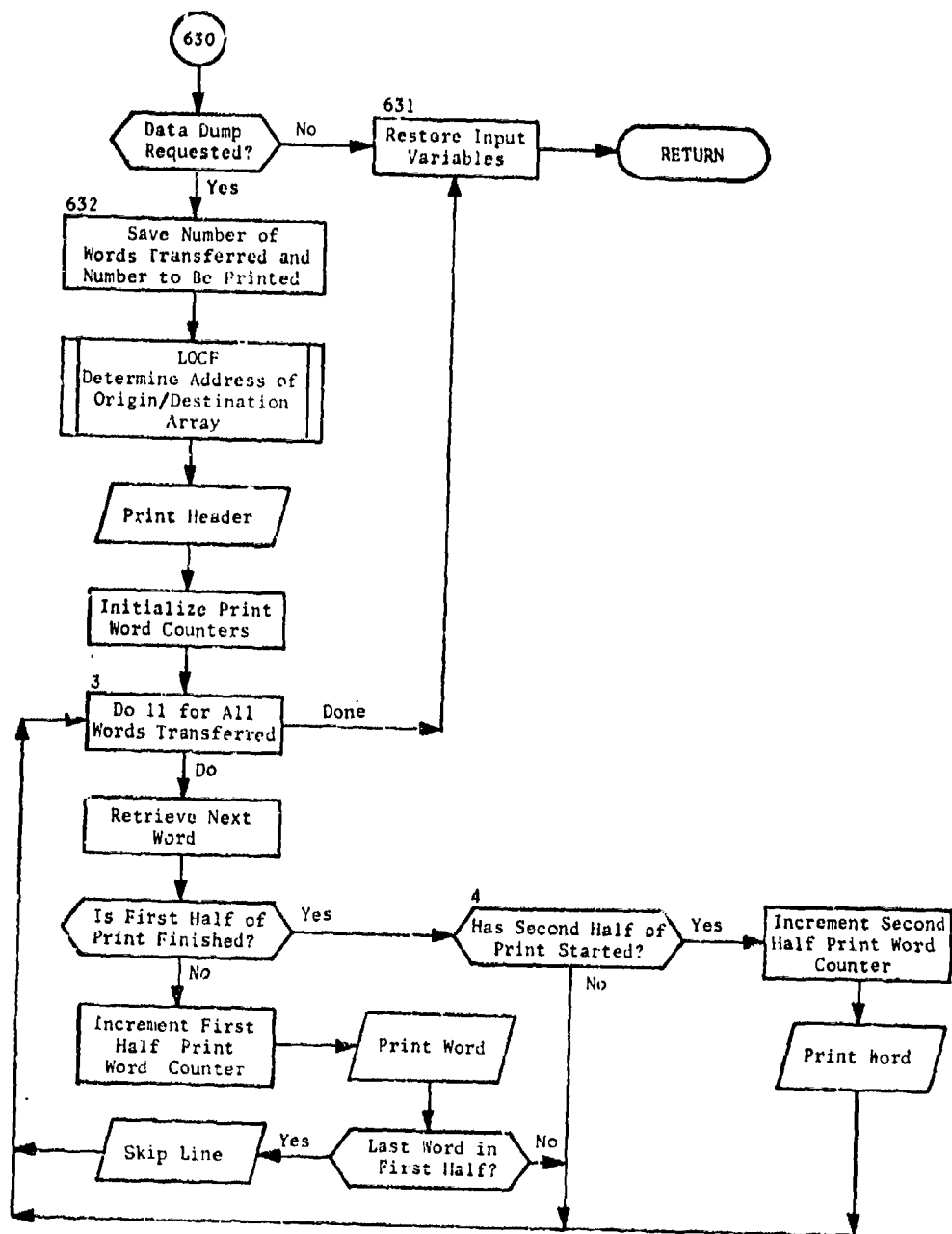


Fig. 4. (cont.)
Part III: RDARRAY/WRARRAY Exit Processing

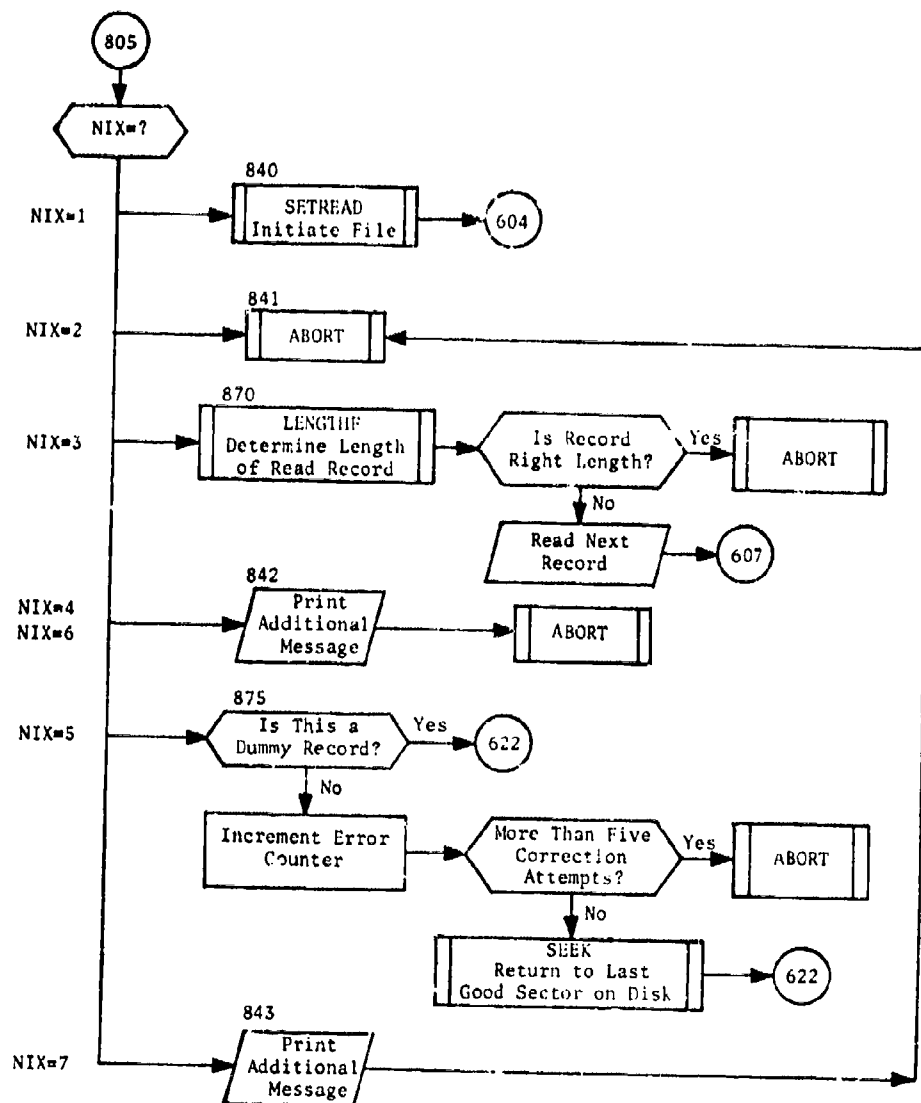


Fig. 4. (cont.)
Part IV: EDARRAY Error Processing

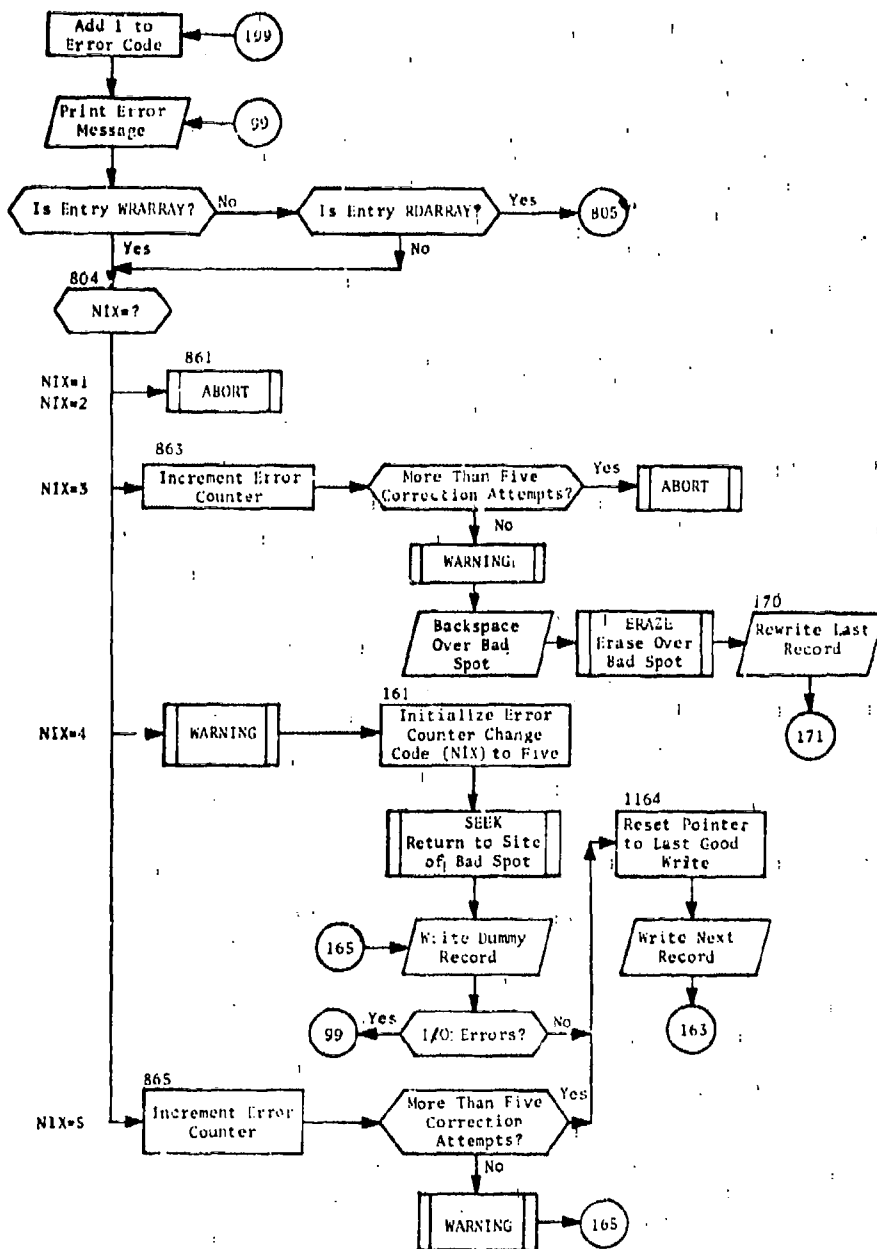


Fig. 4. (cont.)
Part V: WRARRAY Error Processing

SUBROUTINE SETREAD

PURPOSE: This routine initializes a file for reading. It allocates two input buffers to the file according to the buffer number ITP. The input label is processed, and the first two buffer loads of data are input.

ENTRY POINTS: SETREAD

FORMAL PARAMETERS: None

COMMON BLOCKS: DRC5162, INTFILE, FILE, 22642626, FILELAB, FORFLS, ITP, MYIDENT, TWORD, FILABEL, IFTPRNT, MYLABEL, NOPRINT, TODAY, LOCVAR

SUBROUTINES CALLED: ABORT, WARNING, TERMTAP, SCOPE PFS Operating System routines (SEEK, LOCATE, LSTATUS, DREAD, LENGTHF)

CALLED BY: All programs of the QUICK system

Method

SETREAD receives as input from the calling program the buffer number ITP in block /ITP/, and the logical file name MYIDENT in block /MYIDENT/. The interpretation of these variables is discussed in the Functional Description section of this chapter.

Figure 5 shows the processing operations performed by subroutine SETREAD. The flowchart is presented in three parts: Part I, Status Checking and Label Input; Part II, Pointer Setup and Buffer Filling; and Part III, Error Processing.

SETREAD begins by checking to determine if a scratch file is requested. If so, control passes to statement 480 where the name of the file is determined. The file name will be the name used in writing the scratch file, using the same buffer number. For example, if file SCRATCH 1 was written with ITP=7, then a call to SETREAD with ITP=7, MYIDENT=SCRATCH will set the logical file name to be read as SCRATCH 1.

In any event, SETREAD checks the buffer number to be sure it is in the right range. If the file is currently being used by the filehandler for I/O, then subroutine TERMTAP is called to terminate the file. Once

the calling procedure errors are checked, SETREAD reads the file label. Some of the label contents are transferred to block /FILABEL/ where they can be accessed by the calling program.

SETREAD then sets the buffer pointers for the two buffers for the file. The first data record is then read. Note that for tape files, an end of file will be read before the first data record. (As was discussed in the Concept of Operation section and the discussion of entry SETWRITE, two file marks and a dummy record are inserted by the operating system after the label.) This error is error code 4 (i.e., NIX=4). The error correction procedure for subroutine SETREAD allows this error to occur seven times without taking any action. In this manner the operating system insertions on the tape are ignored, and the first data record can be read.

When the first data record has been read, the buffer pointers are set to start emptying this buffer when RDARRAY is called. The filling of the other buffer is then initiated before control returns from SETREAD.

The error correction procedures of SETREAD involve attempts to reread the data. For tape files, five reread attempts are made before the job aborts in trying to read the first data record. For disk files, SETREAD must check that the first data record was not replaced by a dummy NODATA record by subroutine WRARRAY. In any event, up to five reread attempts are made to input the first data record. Note that SETREAD reads and checks the first data record. The immediate checking is required since the call to SETREAD may be followed immediately by a call on RDARRAY to begin data transfer. The second data record, however, is only read in SETREAD. The error detection and correction of the second and later data records is performed in subroutine RDARRAY.

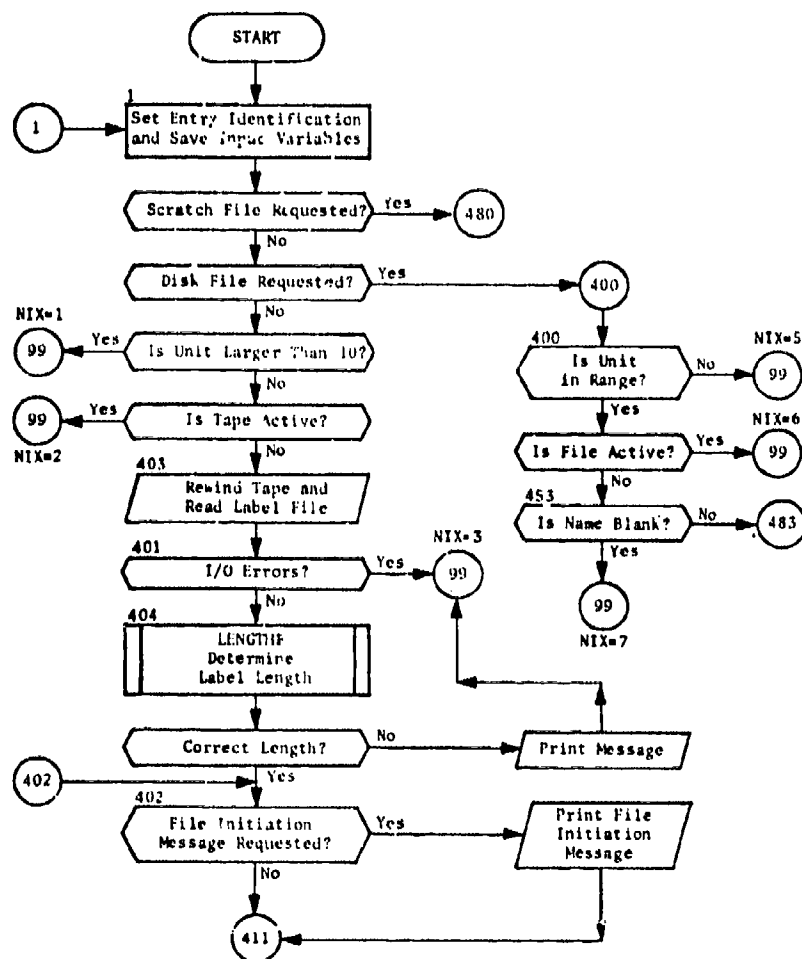


Fig. 5. Subroutine SETREAD
Part I: Status Checking and Label Input
(Sheet 1 of 2)

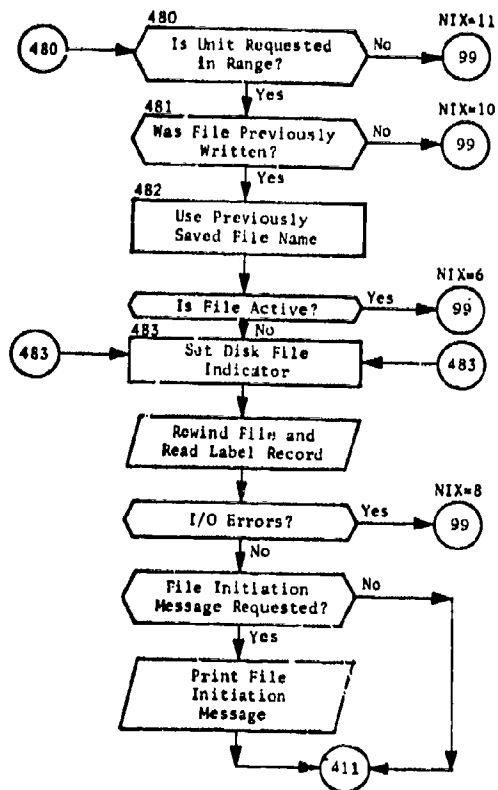


Fig. 5. (cont.)
 Part I: (cont.)
 (Sheet 2 of 2)

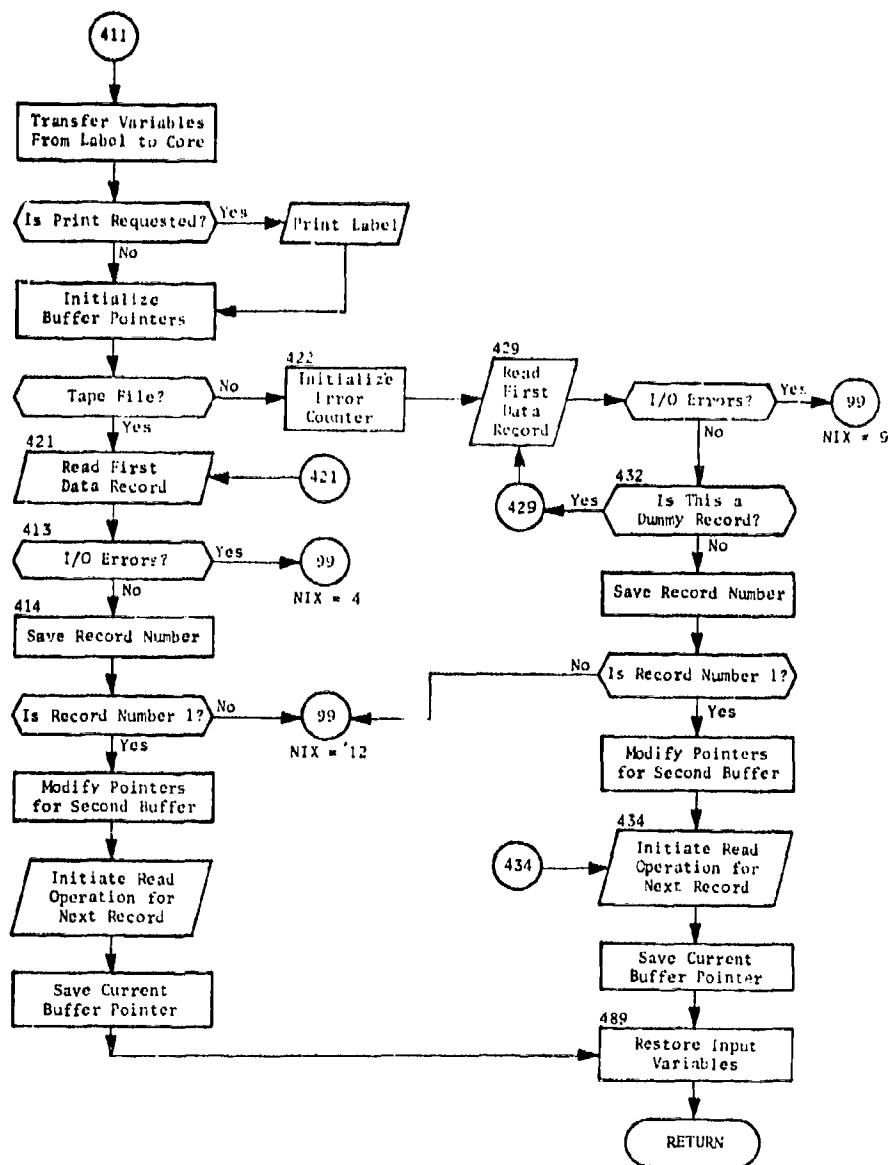


Fig. 5. (cont.)
Part II: Pointer Setup and Buffer Filling

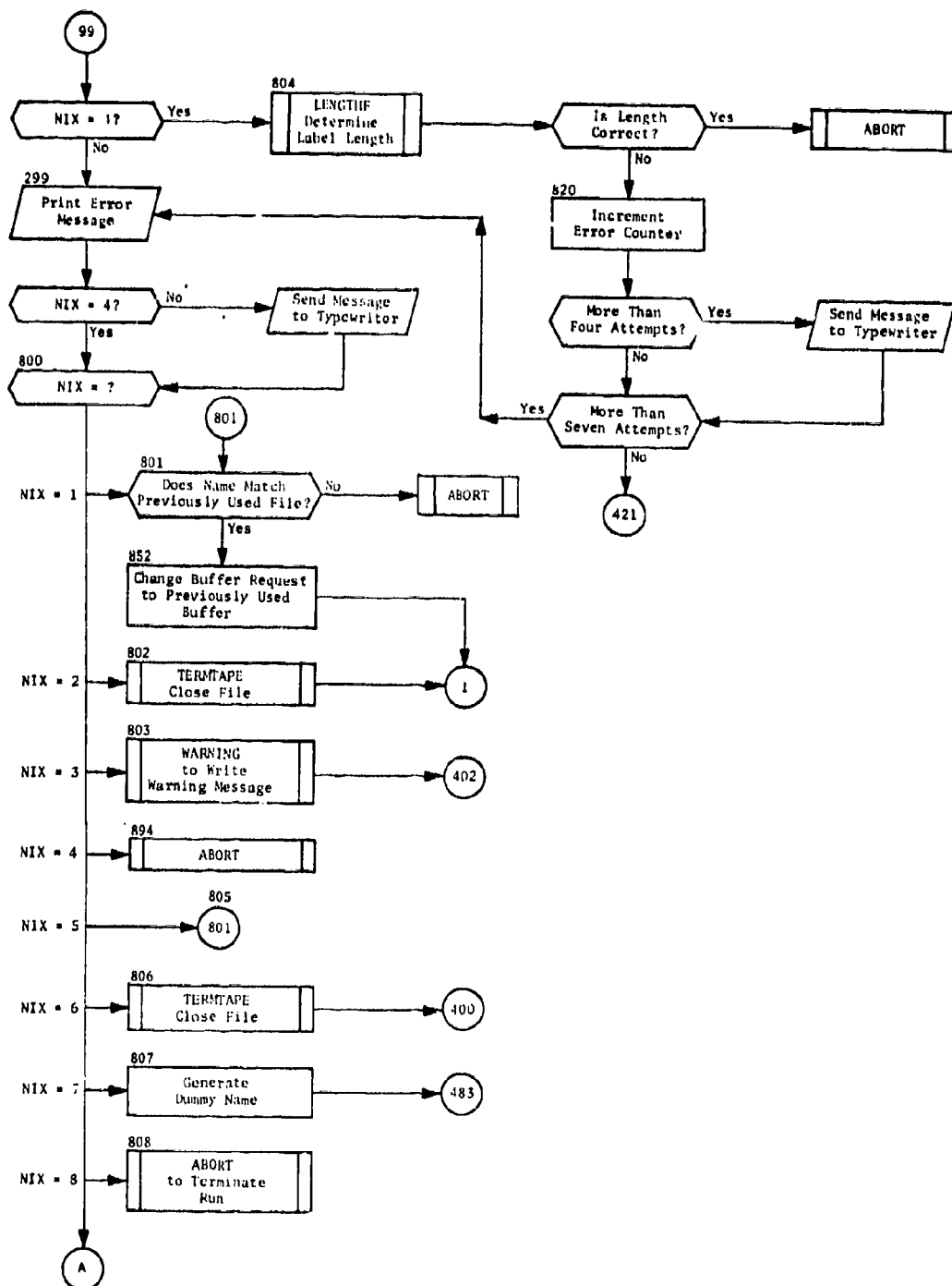


Fig. 5. (cont.)
Part III: Error Processing
(Sheet 1 of 2)

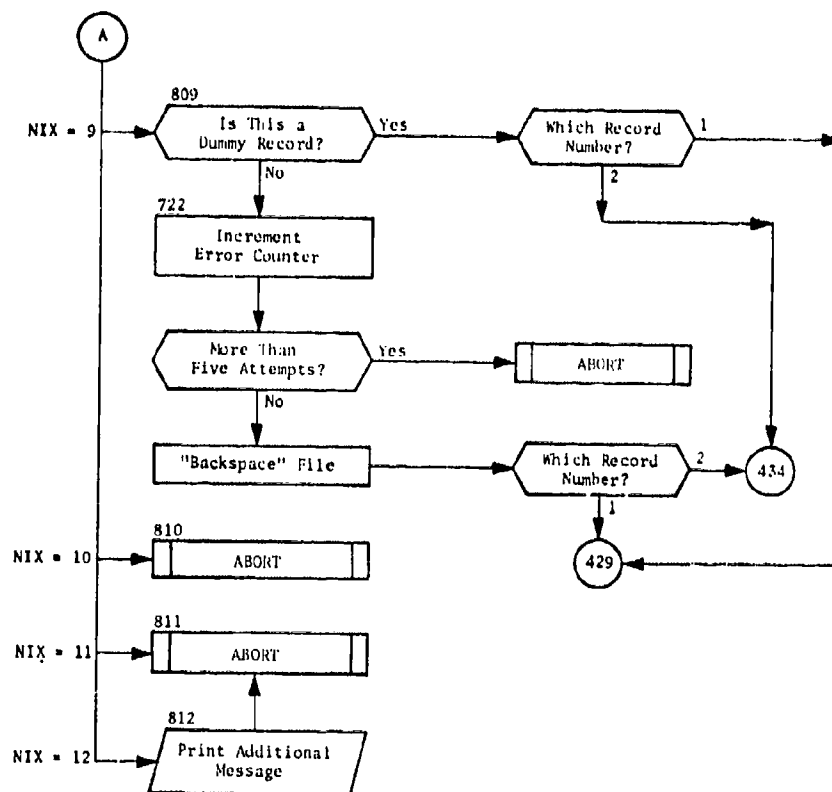


Fig. 5. (cont.)
 Part III: (cont.)
 (Sheet 2 of 2)

SUBROUTINE TERMTAP

PURPOSE: This routine terminates the file using buffer ITP and returns the buffer for use by other files.

ENTRY POINTS: TERMTAPE, TERMTAP, TERMTPE

FORMAL PARAMETERS: None

COMMON BLOCKS: DRC5162, INTFILE, FILE, 22642626, FILELAB, FORFLS, ITP, MYIDENT, TWORD, FILABEL, IFTPRNT, MYLABEL, NOPRINT, TODAY, LOCVAR

SUBROUTINES CALLED: ABORT, WARNING, ERAZE, SCOPE PFS Operating System Disk I/O subroutines (LSTATUS, SEEK, LOCATE, DREAD, DWRITE, DEOF)

CALLED BY: All programs of the QUICK system

Method

The three entry points are at the same location. The three spellings of the entry correspond to the spellings used in the calling sequences.

The only input from the calling program to TERMTAP is the buffer number ITP in common block /ITP/.

For read status files, processing by TERMTAP is very straightforward. As shown in figure 6, tape files are rewound (statement 506) and disk files are returned to their first sector address by subroutine SEEK (statement 505). The file is set not active (statement 507), and ITP is restored to its input value (statement 508), since its absolute value was used as the buffer number.

For write files, processing is more complex for three reasons. First, TERMTAP must empty the last data buffer onto the file and add a padding record to the file. Second, the file label must be rewritten so that it includes the number of words written on the file in its third word. Last, if the file is a non-scratch disk file, its length must be added to the disk directory file DIRC5162.

Before writing the last data record, TERMTAP must check the status of the last write operation. (If the last data record is also the first data record this check is not performed, as the last write operation was the label write operation, checked in SETWRITE.) If errors are encountered

in writing either the last or next to last data record, TERMTAP applies the same error correction procedures described in subroutine WRARRAY for write operation errors. When all data records have been written successfully, a padding record is written. This record contains 96 words of the word PADFILE. This record is required by the double buffering operation of RDARRAY. While transferring data from one data record (in the filehandler buffer), RDARRAY is reading the next data record from the file. Thus, retrieval of information from the last data record requires a padding record to follow. If this were not the case, an end-of-file error would occur whenever data were retrieved from the last data record. If TERMTAP encounters errors in writing the padding record, no attempt is made to correct the error. Finally, an end-of-file mark is placed after the padding record. (Operating system routine DEOF is used for this purpose on all disk files.)

The file is rewound (for disk files, positioned at the beginning), and the label is read. The number of words written on the file is inserted in word three of the label and the file again positioned at the beginning. The new label is then written on the file. No attempt is made to correct I/O errors in reading or rewriting the label.

For tape files, the tape is then rewound. At that point, the operating system inserts two end-of-file marks and a short BCD record between the label and the first data record. The file buffer is deactivated (statement 507), and the subroutine returns control to the calling program.

For scratch disk files, the label rewrite is followed by the buffer return processing of statement 507. For non-scratch disk files, the disk file directory must be updated. The directory is read from the disk. The logical file name is found in the active file list, and the number of words written on the file is inserted in the list. The directory is then rewritten on the disk file. No attempt is made to correct I/O errors in reading or writing the disk file directory. Also, if the logical file name is not found in the active list, only a warning message is issued. In any event, control goes to statement 507 for buffer return processing.

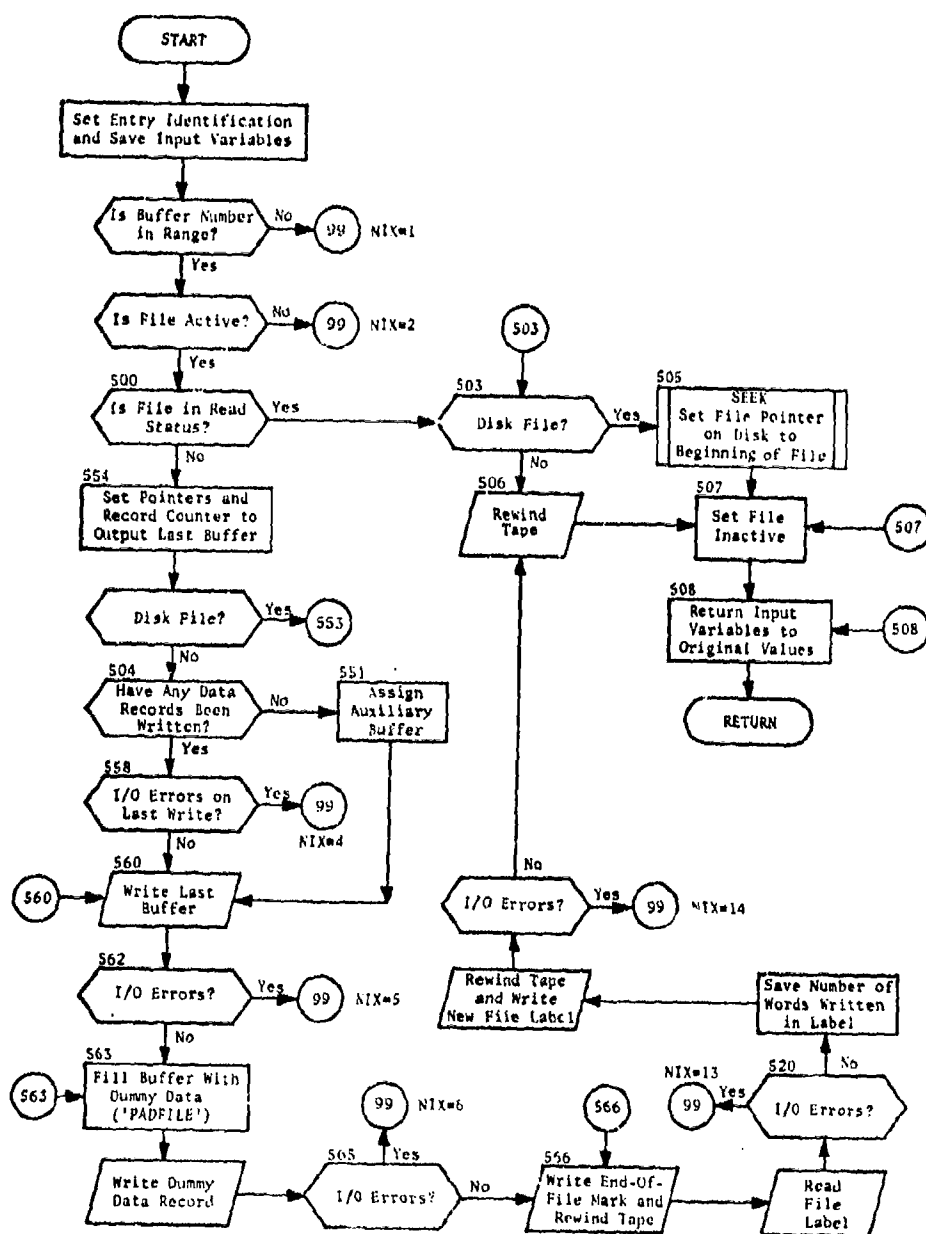


Fig. 6. Subroutine TERMIAP
(Sheet 1 of 4)

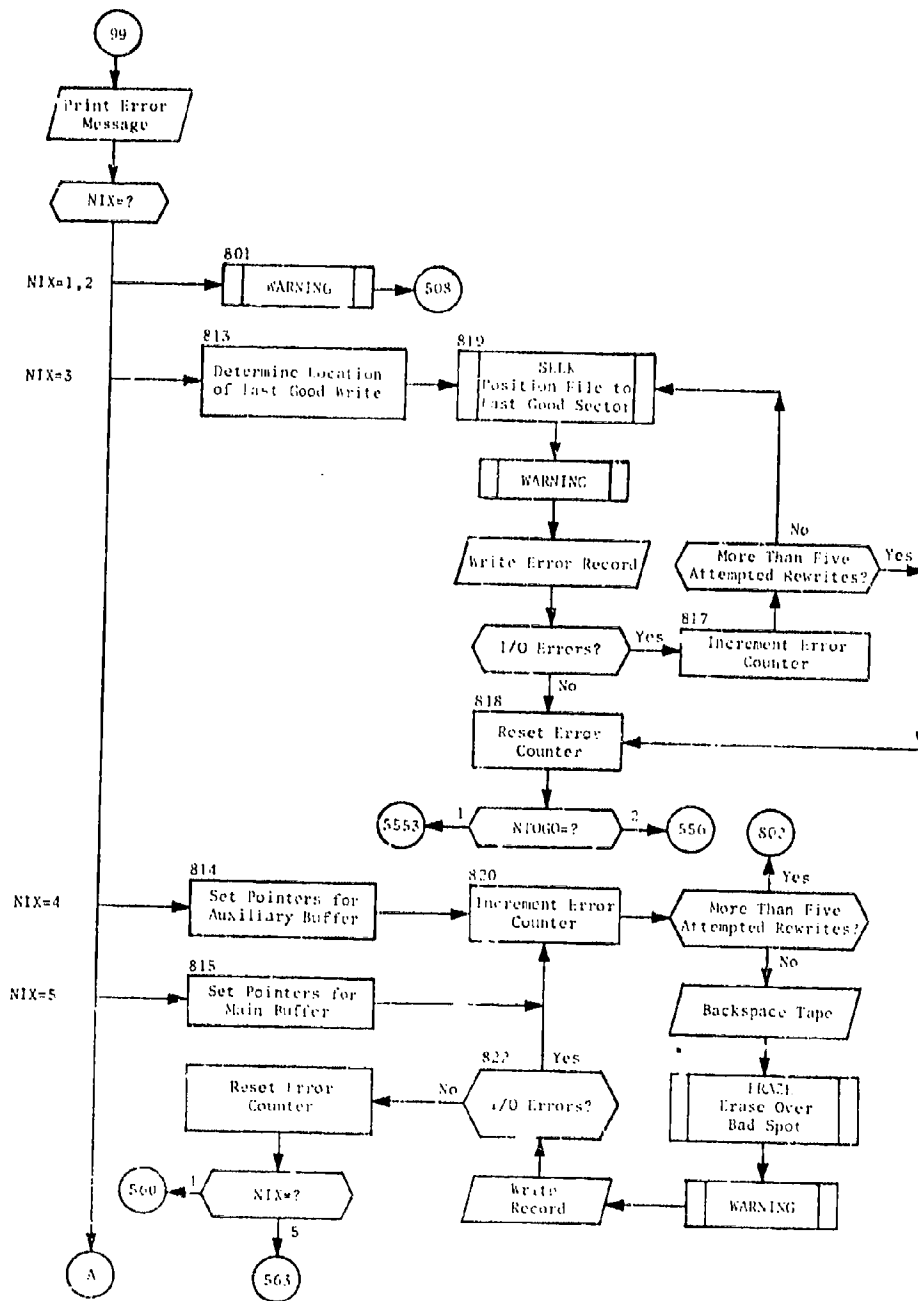


Fig. 6. (cont.)
(Sheet 3 of 4)

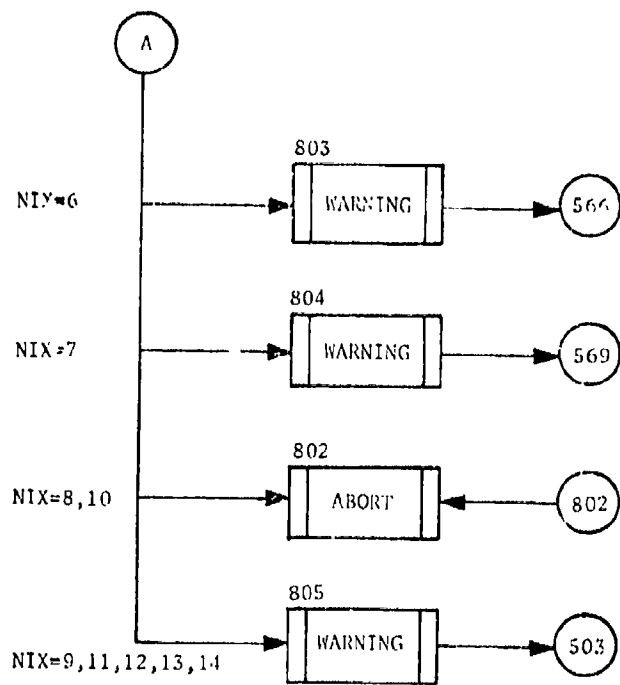


Fig. 6. (cont.)
(Sheet 4 of 4)

CHAPTER 3 SPECIAL-PURPOSE UTILITY ROUTINES

This chapter contains a description of the special-purpose programs OUTFILE, RELOADF, DECLARES, and FILEDUMP. While these programs are part of the QUICK utility package, they are presented in a separate chapter because of the unique nature of the support functions they perform.

The common blocks used by programs OUTFILE and RELOADF are described in table 3 (following the description of program OUTFILE). The common blocks used by DECLARES and FILEDUMP are applicable to other utility routines and are, therefore, presented in appendix A.

PROGRAM OUTFILE

PURPOSE:

To read information from disk files and write it on magnetic tape in such a way that either the last program run can be repeated or the next program can be run by restarting the subsystem (using program RELOADF to reload the disk files from the magnetic tapes).

ENTRY POINTS:

OUTFILE

FORMAL PARAMETERS:

None

COMMON BLOCKS:

BUFFERS, DICTARY, DISKIO, DRC5162, DSKHARD, ERRMESS, ERRNUM, FILENOW, FILEREC, LOCATOR, MACHINE, SAVWRIT, SUPRVIS, TAPHARD

SUBROUTINES CALLED:

CLOSPIL, CLRCMON, GETDF, LOCATE, NODIRC, OPENSPL, OUTBFTP, OUTERTP, SETHEAD

Method

The primary function of program OUTFILE is to dump information from disk files used by a program in one of the QUICK subsystems onto spill tapes in such a way that program RELOADF can be used to reload the disk files from the disk files at a later time so that the QUICK subsystem can be restarted where it stopped.

OUTFILE itself does not read or write information to or from tapes or disks, but instead calls subroutines which handle these functions. First it uses CLRCMON to initialize the variables in all of its common blocks.

Then it supervises the process by which the master directory (common /DRC5162/) is read from the disk file whose name is contained in variable NAMDIRC (common /SUPRVIS/). It reads the directory twice and each time uses subroutine SEEK* to position the disk read head at the beginning of the file. The full length of the directory is unknown until it is read; hence, OUTFILE first uses subroutine DREAD* to input 32 (LENLIST = 32) words of the directory. The second time DREAD inputs the full master directory since now LENLIST = DIRC(1) which is the first word of the master directory. In the event that end-of-file or parity errors are

*PFS Disk I/O subroutine

encountered during this process, OUTFILE repeats it until the error is corrected or until it has failed MTIMESR (common /SUPRVIS/) times to correct the error; in the latter case it calls NODIRC to write appropriate error information on the standard output file and to abort the run.

When the full length of the master directory has been read, OUTFILE calls SETHEAD to determine the values of variables in the major directory (common /LOCATOR/). This directory contains information about the manner in which the disk files will be dumped onto the spill tapes; OUTFILE writes this directory as the first file of the first spill tape. However, if it encounters an end-of-tape or parity error, it calls OUTERTP with the argument equal to 1, to correct the operation by repeating it.

Finally, OUTFILE controls the process by which the spill tapes are written. OPENSPL is called once for each tape to set up and buffer out the minor directory (common /DICTARY/), which references it, to the first file of the tape (except for the first spill tape where the minor directory is the second file). Then, for each record on the tape OUTFILE first calls GETDF to fill the input buffer from the current disk file(s), and then calls OUTBFTP for the record to buffer out the output buffer to the record on the spill tape. When all the records for a spill tape have been buffered out, CLOSPIL is called to terminate the tape and see that a new tape is mounted if another reel will be needed.

OUTFILE terminates its processing after writing error recovery messages to the computer operator and to the output file.

Table 3 lists the common blocks used by program OUTFILE*. The flow of operations in OUTFILE is shown in figure 7.

*Program RELOADF also uses these common blocks.

Table 3. Common Blocks (Programs OUTFILE
and RELOADF)
(Sheet 1 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
BUFFERS	INBUFF	The index of the buffer currently being used for input (INBUFF=1 or 2)
	IOUTBUF	The index of the buffer currently being used for output (IOUTBUF=1 or 2)
	NSECTBF	The number of disk sectors which will fit in one buffer
	LBUFF	The length in words of one buffer
	IOBUFF	The input/output buffer (IOBUFF has the dimensions LBUFF and 2 - i.e., IOBUFF(LBUFF,2))
DICTARY		DICTARY is output as the first file on all spill tapes except the first tape where it is the second file; it is the 'minor directory' and refers to only one tape
	IAM	Spill tape number
	NRFC	Number of records on this tape
	IDENT	Spill tape identifier
	NUMONME	Number of files on this tape
	NAMEONT	An array containing the names of files on this tape (it is dimensioned MAXFILE - see LOCATOR block description)
	LONT	An array containing the word lengths of files on this tape (it is dimensioned MAXFILE - see LOCATOR block description)
DISKIO	ISTART	Index to the starting word in the input/output buffer IOBUFF
	IEND	Index to the ending word in the input/output buffer IOBUFF
	ISTREC	Index to the starting word in the input/output buffer for this record
	IENDREC	Index to the ending word in the input/output buffer for this record

Table 3. (cont.)
(Sheet 2 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
DRC5162		DRC5162 is the master directory; it is maintained on the disk by the filehandler and contains information about the disk files which are to be spilled
	LENLIST	Length of the master directory file
	NOWACTV	The number of active files
	LFNAME	An array containing the names of the files
	NOWORDS	An array containing the lengths in words of the disk files
	NINTRNL	The number of internal scratch files currently allocated to the program
	LINTRNL	The length in words of a scratch file
DSKHARD		This block contains information about the disk hardware
	NWDSCT	The number of words per disk sector
	NSCTRAK	The number of disk sectors per track
ERRMESS		A common block used in conjunction with subroutine ABORT (see appendix A for description)
ERRNUM	NTERRTP	The total number of tape read or write errors
	NCERRTP	The number of tape read or write errors for the current tape
	NTIMTAP	The number of times which the current tape has been changed
	NTERRDF	The total number of disk read or write errors
	NCERRDF	The number of disk read or write errors for the current disk file

Table 3. (cont.)
(Sheet 3 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
ERRNUM (cont.)	IAMOK	An error recovery indicator for the spill tapes = 1 Successful error recovery = 0 Unsuccessful recovery from error not on first tape =-1 Unsuccessful recovery from error on first tape
	IRPERR	A disk file read or write error indicator = 0 No error .GT.0 The number of words left in the file after a parity error .LT.0 End-of-file error
	FILENOW	This common block is a work area for a read/ write operation affecting the current disk file
	NAMENOW	The logical name of the current disk file
	NWORDNW	The number of words read from/written to the current file
	LIMIT	The number of words left to be read from/written to the current file
	LOCNOW	The address of the present sector in the current file
	NUMNOW	The number of the current file
	NXTRAIL	The number of trailer words for the current disk file to be read from/written on the current spill tape
	NXTHEAD	The number of header words for the current disk file to be read from/written on the current spill tape
FILERECD	LFILABL	The length of the filehandler disk file label record
	LPHYREC	The length of disk physical records which are provided by the filehandler

Table 3. (cont.)
(Sheet 4 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
LOCATOR		This block is the first file on the first spill tape and is the major directory for the spill tapes
	NTAPES	The number of spill tapes used
	NFILES	The number of disk files spilled
	MAXFILE	The maximum number of disk files which can be spilled
	NAMES	An array containing the logical file names of the disk files dumped onto the spill tapes; it is dimensioned MAXFILE
	LENGTHS	An array containing the lengths in words of the disk files which are contained on the spill tapes; it is also dimensioned MAXFILE
MACHINE	IREAD	Logical unit number for standard input file
	IWRIT	Logical unit number for standard output file
	ICOMM	Logical unit number for standard comment file to computer operator
	IPUNCH	Logical unit number for standard punch output file
SAVWRIT		This common block saves the starting pointers for the first data on the current spill tape
	NUMS TRT	The file number of the first disk file on the current spill tape
	LOCSTRT	The disk sector address for the first data on the current spill tape
	NXTHSV	The number of header words to be output on the spill tape before any disk files are dumped
	NXTTSV	The number of trailer words to be output on the spill tape before any disk files are dumped
	NWDRDSV	Number of words already input from the first disk file and written on the current spill tape

Table 3. (cont.)
(Sheet 5 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
SUPRVIS	MAXUNIT	The maximum number of spill tape logical units
	IUNIT	An array containing the three logical unit numbers of the spill tapes
	NOWUNIT	The logical unit number of the current spill tape
	NAMDIRC	The logical file name of the master directory (i.e., common /DRC5162/)
	MTIMESR	The maximum number of times the program attempts to correct a read error
	MTIMESW	The maximum number of times the program attempts to correct a write error
	MCHANGE	The maximum number of tape changes allowed while attempting to correct a tape error
	LWRIT	The number of words written from the current output buffer onto the current spill tape record
	LDIM	The dimensioned length of the master directory (common /DRC5162/)
	IDNTSAV	The value of IDENT which is set in subroutine CLRCMON
	LUNDISK	The logical unit number for disk
	LNHDTL	The length of the header and trailer blocks which precede and follow, respectively, a disk file which is dumped on a spill tape
TAPHARD		This block contains information about the magnetic tape hardware
	MXWDSTP	The maximum number of words which can be written on one reel of magnetic tape if the physical record size is LBUFF (in common /BUFFERS/)
	IMODE	The parity mode of tape I/O operations = 1 if mode is BINARY

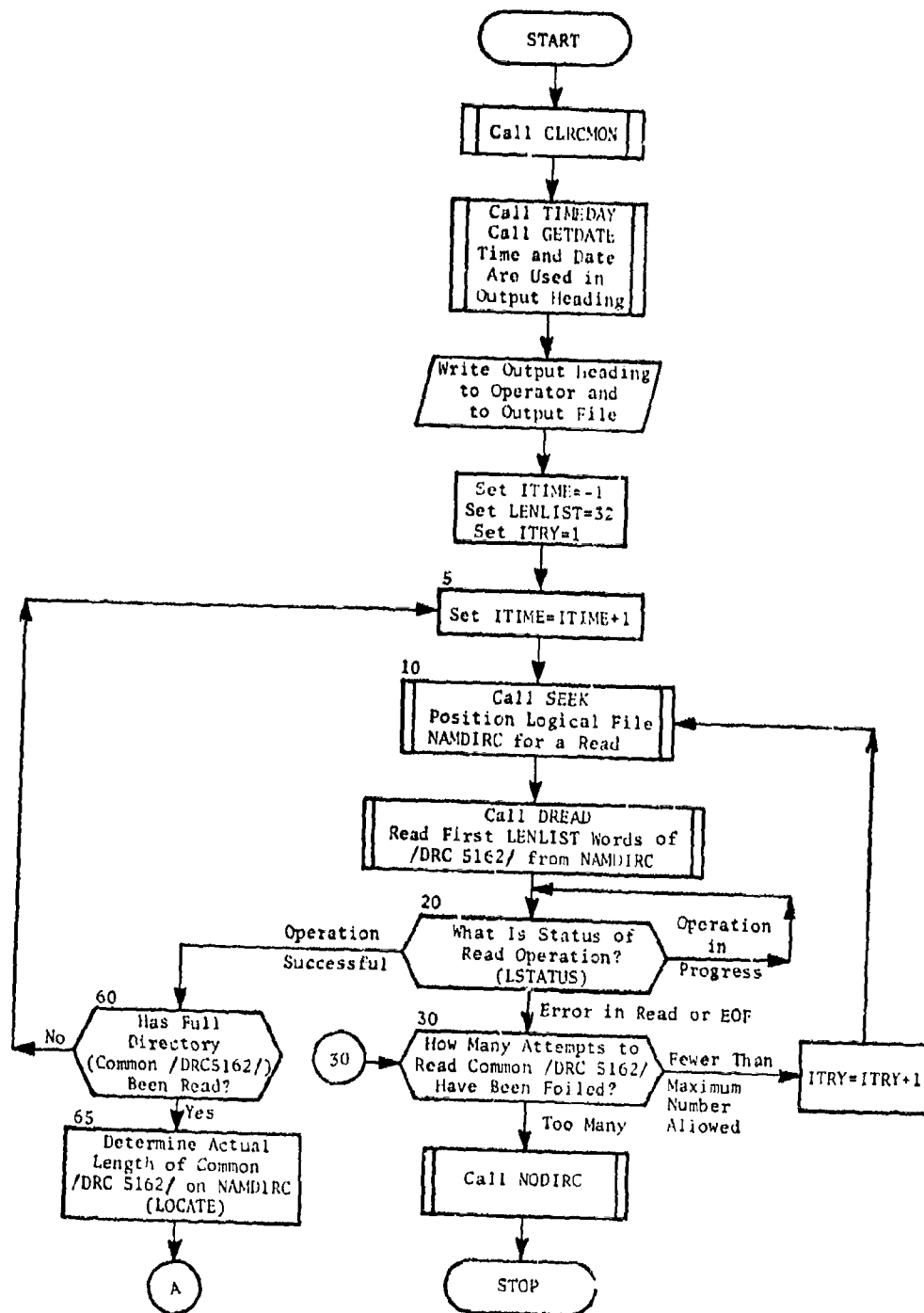


Fig. 7. Program OUTFILE
(Sheet 1 of 3)

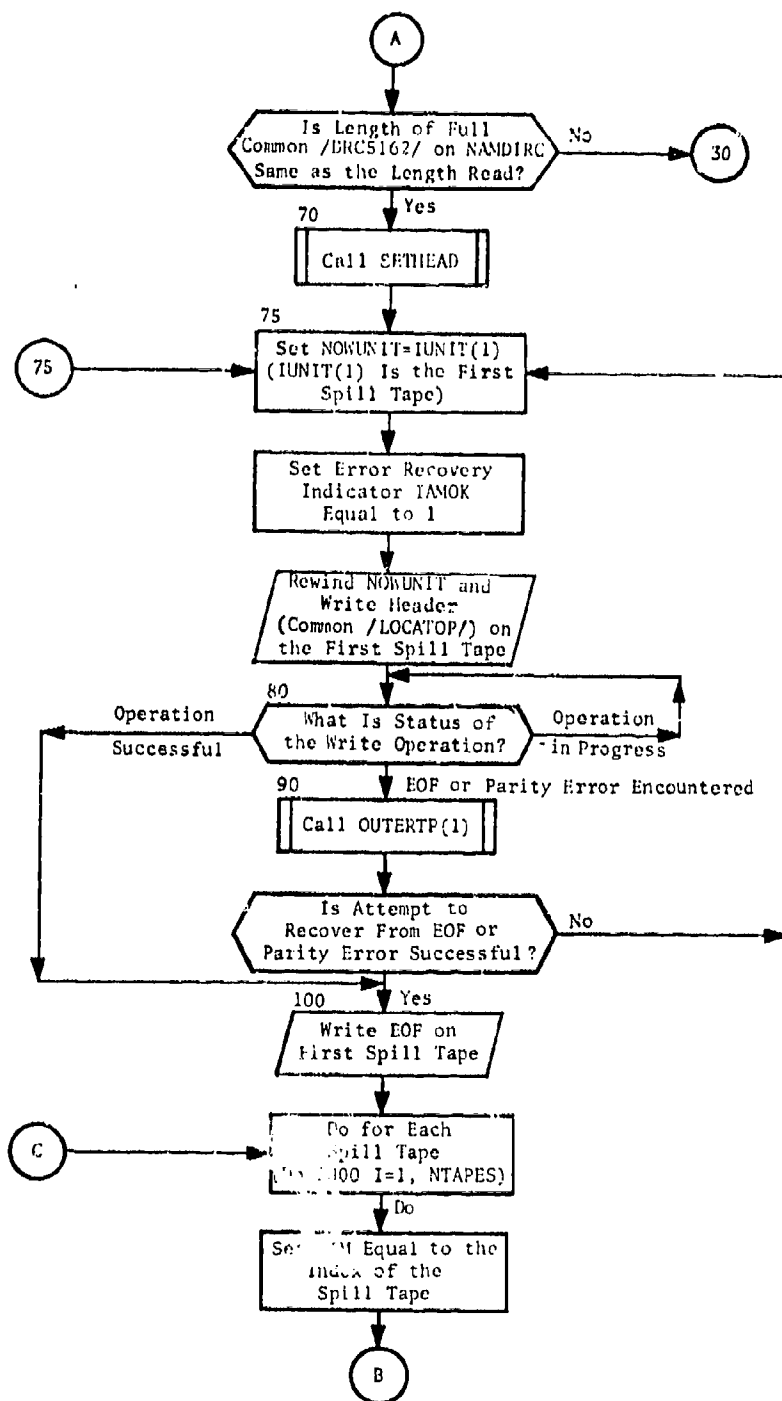


Fig. 7. (cont.)
(Sheet 2 of 3)

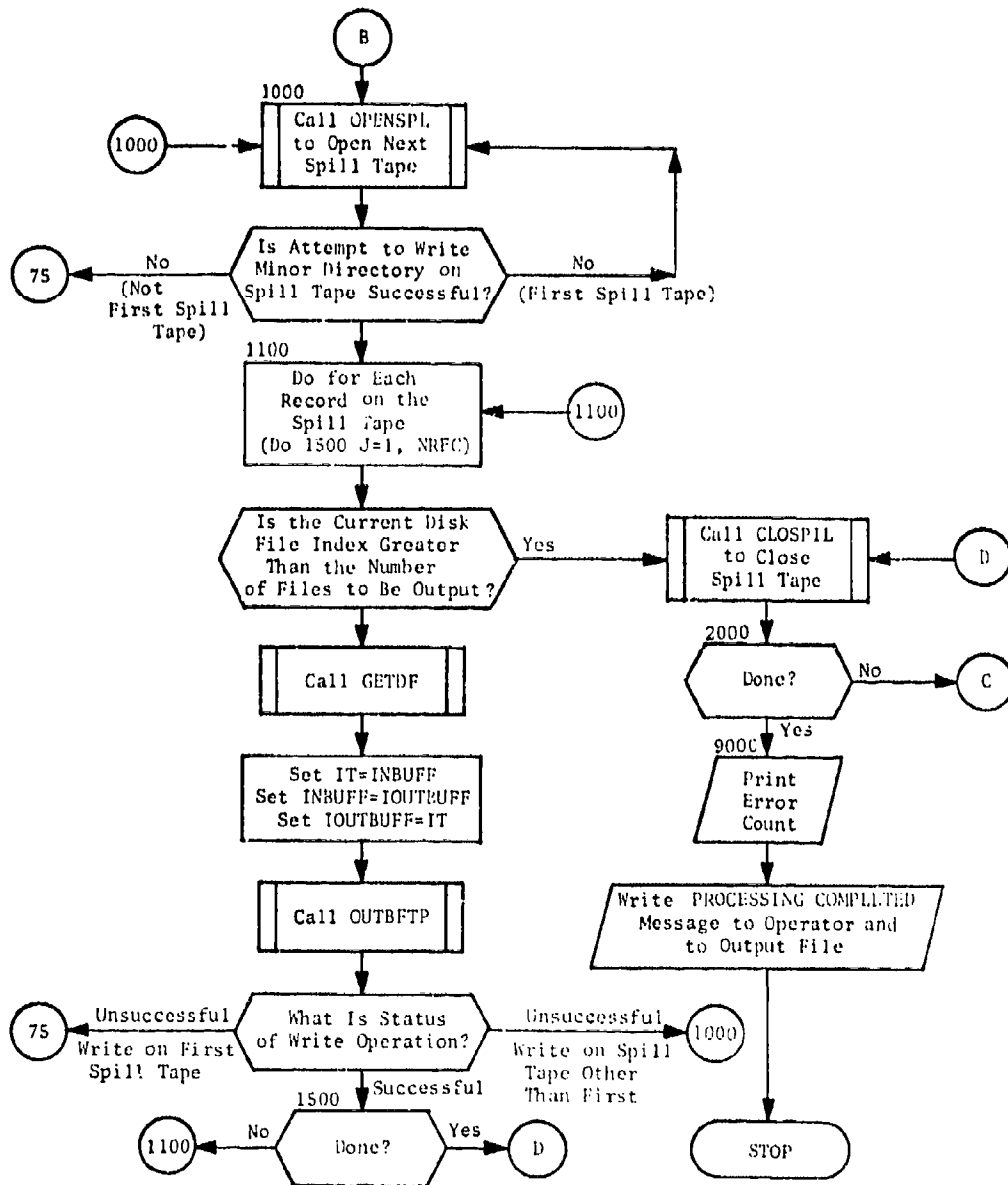


Fig. 7. (cont.)
(Sheet 3 of 3)

SUBROUTINE CLOSPIL

PURPOSE: To terminate the spill tape and to recover from end-of-tape or parity errors on a spill tape, if necessary.

ENTRY POINTS: CLOSPIL

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DICTARY, ERRNUM, FILENOW, LOCATOR, MACHINE, SAVWRIT, SUPRVIS

SUBROUTINES CALLED: OUTERTP, ANOTHER

CALLED BY: OUTFILE

Method

Subroutine CLOSPIL first checks the status of the spill tape which it is to terminate. If an end-of-tape or parity error was encountered by OUTFILE when it tried to write on the tape, CLOSPIL calls OUTERTP with the argument equal to 3 to attempt to recover from the error. If recovery is successful, CLOSPIL continues processing the tape as indicated below. Otherwise OUTERTP aborts the run.

If and when all records have been successfully written on the spill tape, CLOSPIL puts an end-of-file mark on the tape. CLOSPIL uses subroutine ANOTHER to unload the terminated tape.

CLOSPIL then instructs the computer operator first to save and label the old tape and, if another spill tape is to be required on the unit, to mount a fresh tape on the same tape unit. CLOSPIL completes its processing by resetting the variables in common block /SAVWRIT/ and writing a message to the output file concerning the number of errors encountered and corrected before the spill tape was successfully written.

Subroutine CLOSPIL is illustrated in figure 8.

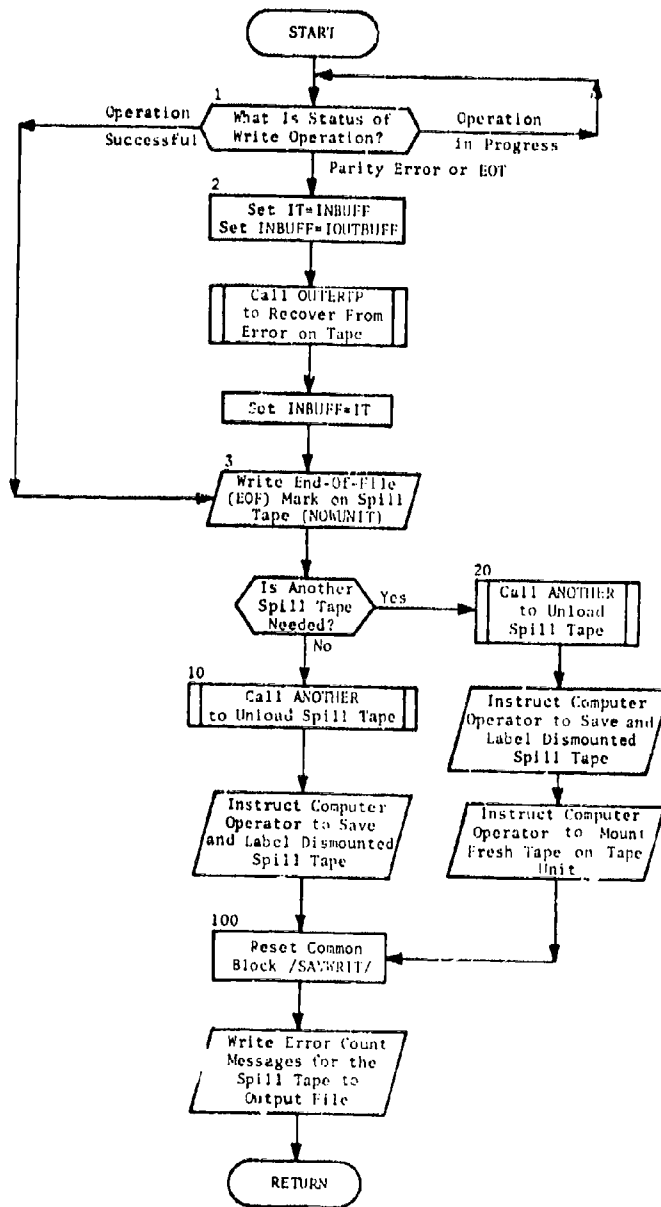


Fig. 8. Subroutine CLOSPIL

SUBROUTINE CLRCMON

PURPOSE: To clear or initialize all common blocks for program OUTFILE (or program RELOADF).

ENTRY POINTS: CLRCMON

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DICTARY, DISKIO, DRC5162, DSKHARD, ERRMESS, ERRNUM, FILENOW, FILEREC, LOCATOR, MACHINE, SAVWRIT, SUPRVIS, TAPHARD

SUBROUTINES CALLED: None

CALLED BY: OUTFILE (or RELOADF)

Method

Subroutine CLRCMON initializes all the variables in the common blocks used by program OUTFILE (or program RELOADF). It assumes that the variables LBUFF, MAXFILE, MAXUNIT, and LDIM have been set by DATA statements at load time; the variables are defined in the calling program. There is no significance to the order in which the common block variables are initialized.

Subroutine CLRCMON is illustrated in figure 9.

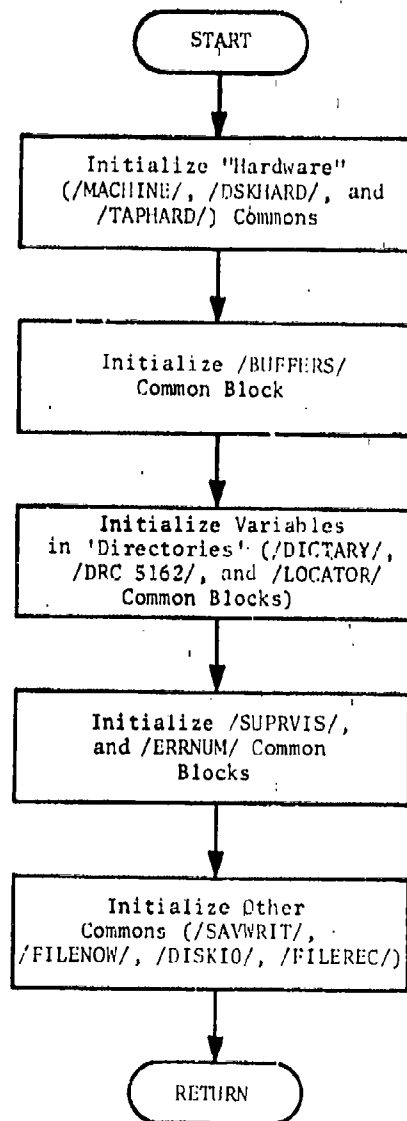


Fig. 9. Subroutine CLRCMON

SUBROUTINE GETDF

PURPOSE: To set up a disk file for input into OUTFILE.

ENTRY POINTS: GETDF

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DISKIO, DSKHARD, ERRNUM, FILENOW, FILEREC, LOCATOR, MACHINE, SAVWRIT, SUPRVIS

SUBROUTINES CALLED: INBUFDK, INERRDK, INLABEL

CALLED BY: OUTFILE

Method

If, when the last call to GETDF occurred, the last word placed in the input buffer either immediately preceded a trailer label or was part of a trailer or header label for a disk file, subroutine GETDF now places the remainder of the trailer label and the header label (or the remainder of the header label) at the beginning of the input buffer; then it calls INLABEL to place the filehandler label after these labels in the buffer.

If the remainder of the input buffer can be filled from the current disk file, GETDF determines how many words are needed to fill it and calls INBUFDK to actually transfer the words from the disk file into the buffer. However, if more than one disk file is required to fill the buffer, GETDF processes one file at a time until the buffer is full. First it calls INBUFDK to place the remainder of the current file into the buffer; after that it places the whole trailer label into the buffer if it will fit. (Otherwise, part of the label is saved for the next buffer.) Then it determines whether the header label for the next disk file can be added to the buffer and places it there if it fits. (Again, part of the label is saved for the next buffer if it does not fit.) Finally, it calls INLABEL to place the filehandler label for the new file into the buffer. If the buffer is still not full, the length of the new disk file is compared with the size of the remaining buffer space to determine whether this file will now fit into the buffer; then processing continues as described above until the buffer is full.

Whenever end-of-file or parity errors are encountered by INBUFDK or INLABEL while these routines are reading words from the disk files, GETDF calls

INERRDK to attempt to recover from the errors. When the recovery is unsuccessful, warnings are written to the output file and an appropriate amount of the input buffer is filled with the error message PARITYER.

Subroutine GETDF is illustrated in figure 10.

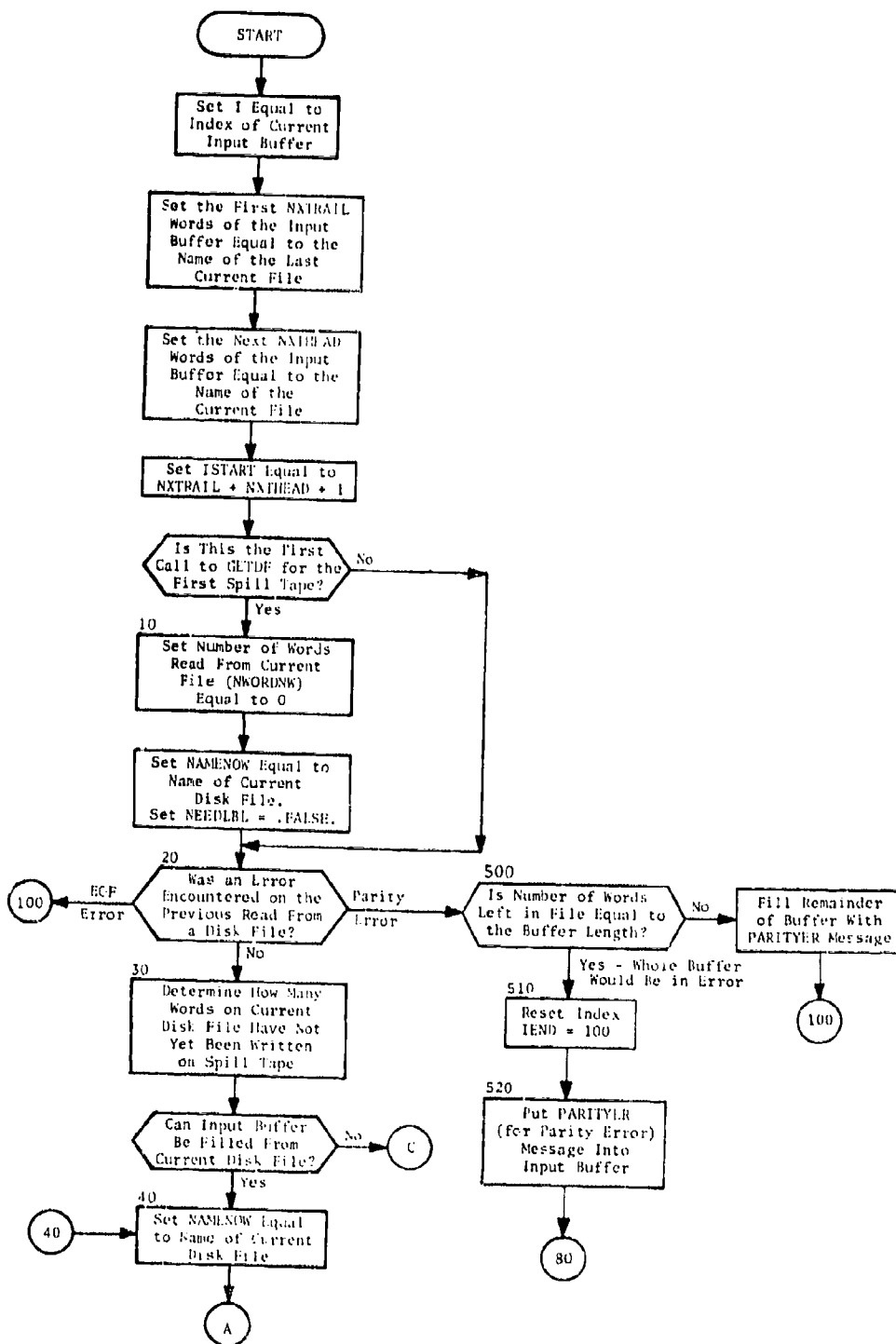


Fig. 10. Subroutine GETDF
(Sheet 1 of 4)

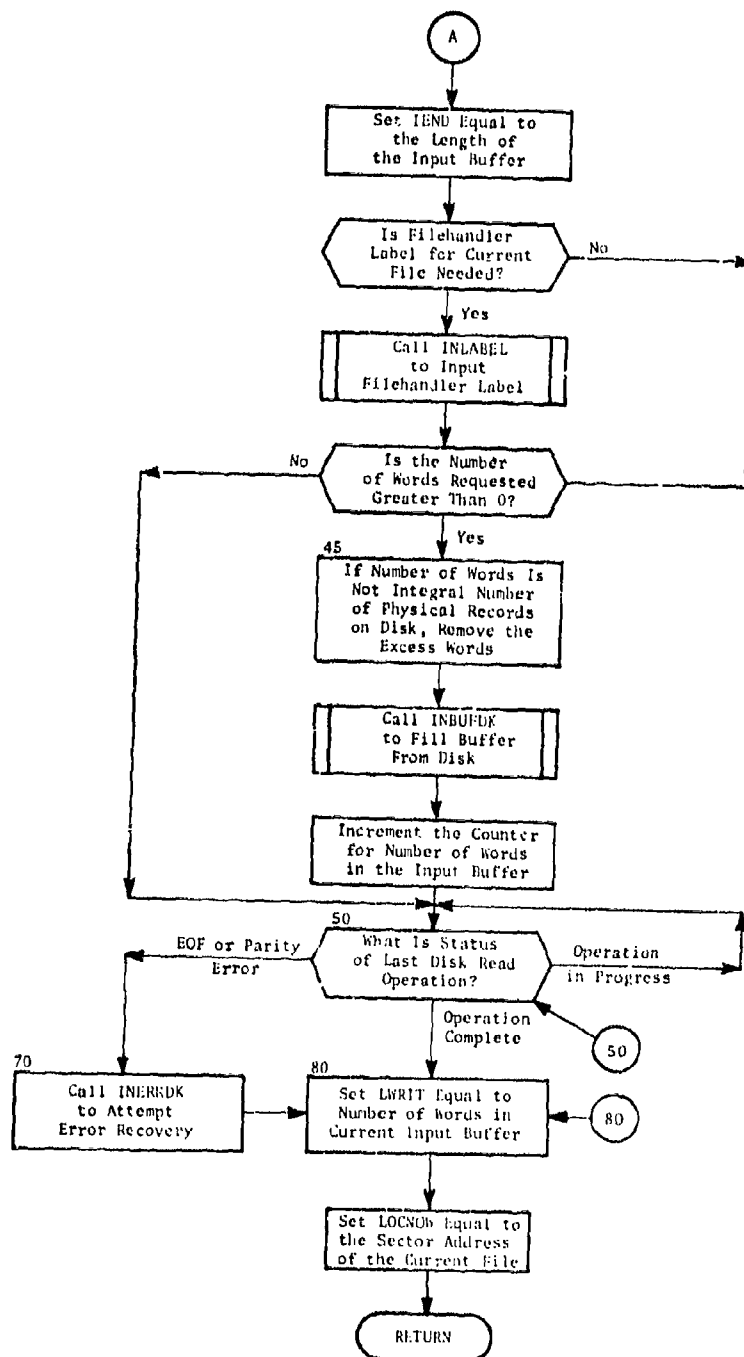


Fig. 10. (cont.)
(Sheet 2 of 4)

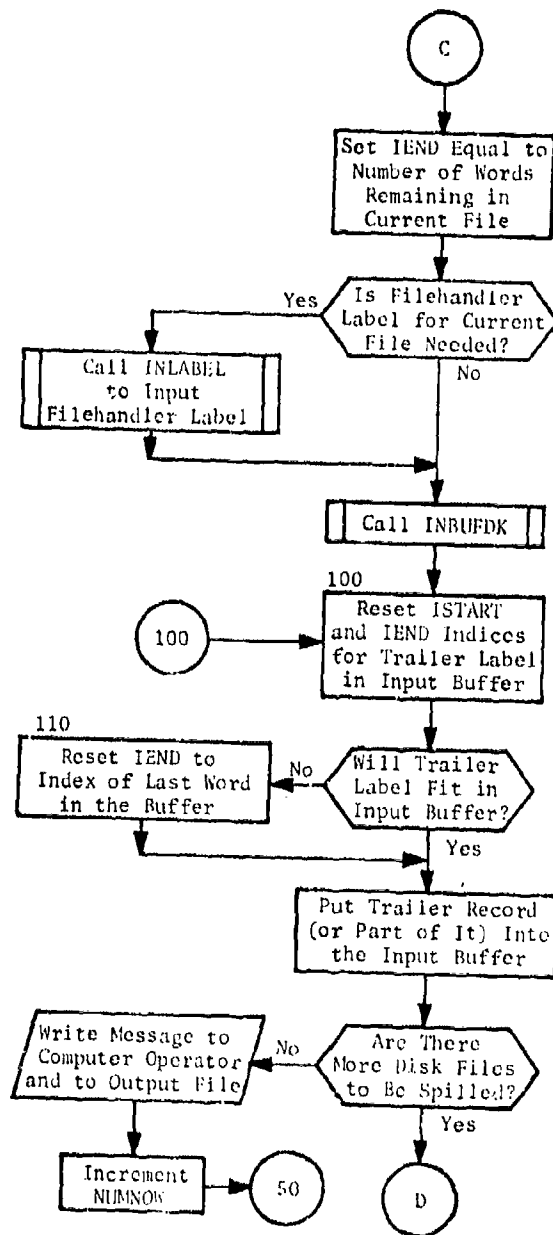


Fig. 10. (cont.)
(Sheet 3 of 4)

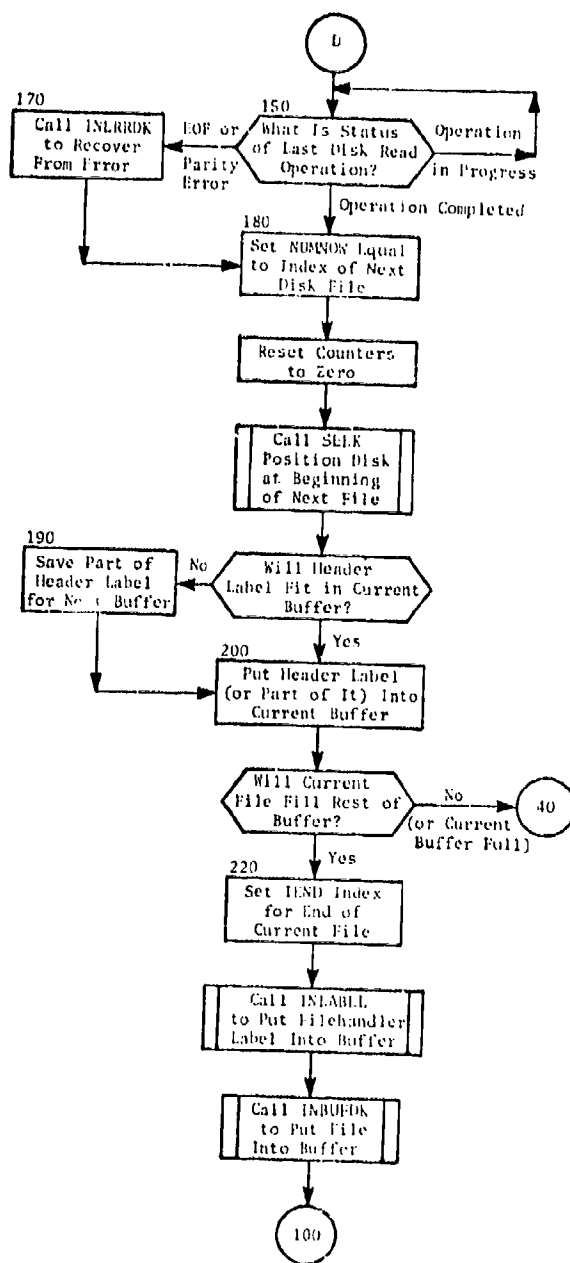


Fig. 10. (cont.)
(Sheet 4 of 4)

SUBROUTINE INBUFDK

PURPOSE: To fill the input buffer from the current file on the disk.

ENTRY POINTS: INBUFDK

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DISKIO, DSKHARD, FILENOW, FILEREC, SUPRVIS

SUBROUTINES CALLED: INERRDK

CALLED BY: GETDF, INLABEL

Method

Subroutine INBUFDK first determines how many whole physical records must be read from the current disk file to fill the input buffer from ISTART to IEND; it calls DREAD* once for each record to actually fill the buffer. Before each call to DREAD, INBUFDK calls LSTATUS* to determine whether the last disk read operation has been successfully completed. If an end-of-file or parity error has been encountered, INERRDK is called to attempt a recovery from the error. Finally, if only a part of a physical record must be read to finish filling the buffer, a last call to DREAD is made to perform the operation.

Subroutine INBUFDK is illustrated in figure 11.

*PFS I/O disk subroutines

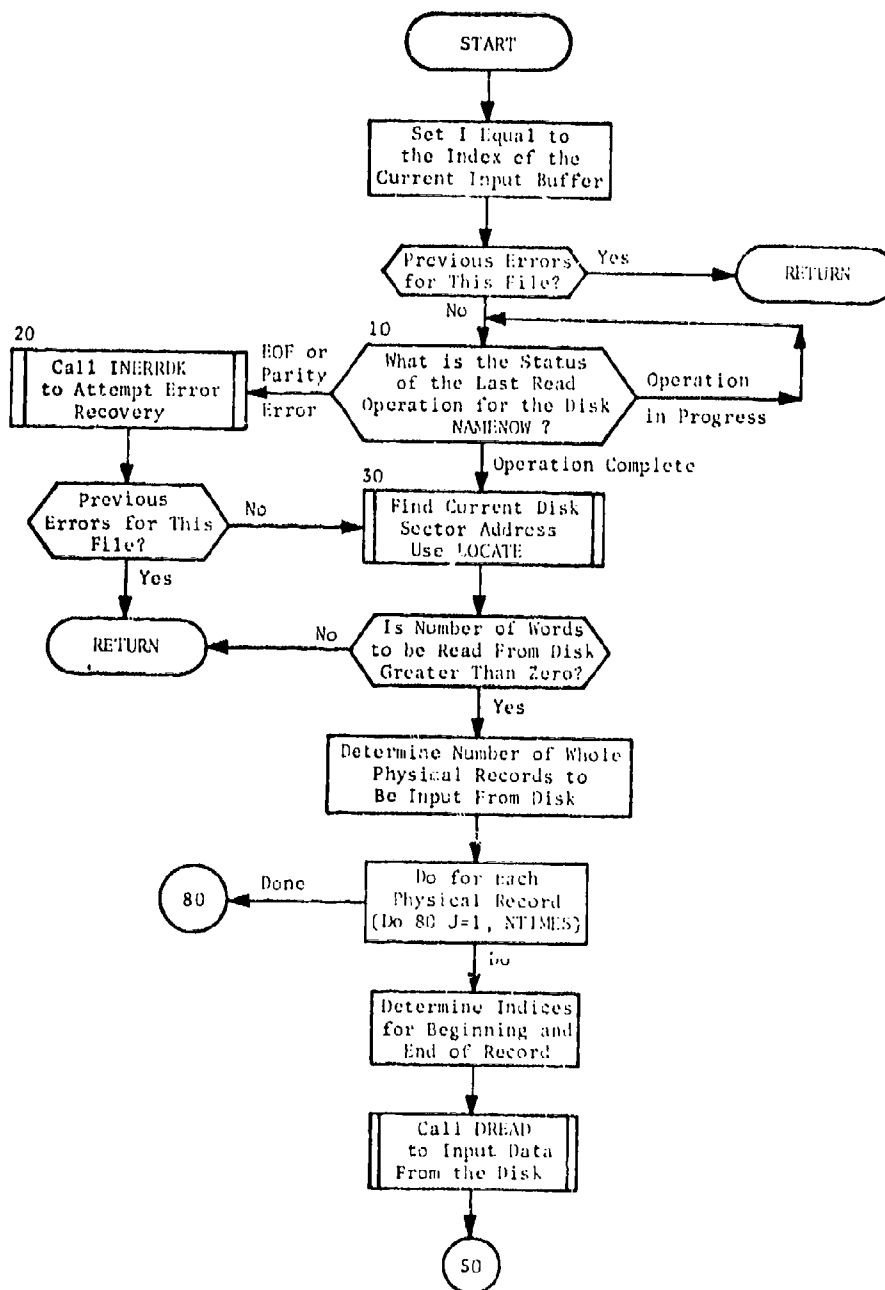


Fig. 11. Subroutine INBHFDK
(Sheet 1 of 2)

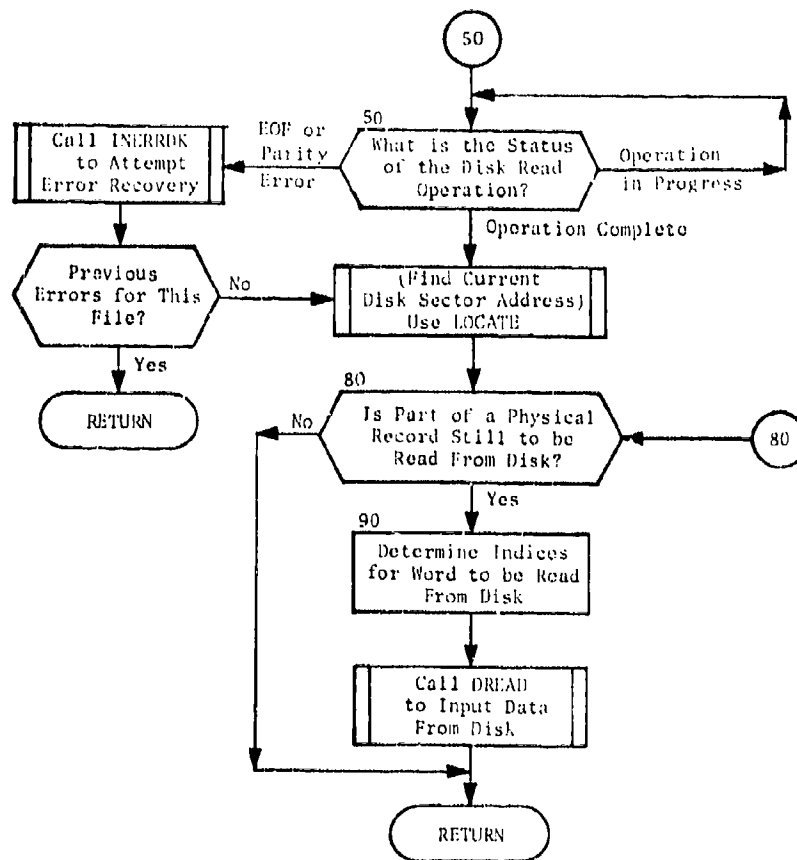


Fig. 11. (cont.)
(Sheet 2 of 2)

SUBROUTINE INERRDK

PURPOSE: To repeat a disk file read operation which terminated because of an end-of-file or parity error in order to correct the read if this is possible.

ENTRY POINTS: INERRDK

FORMAL PARAMETERS: I - An error type indicator.
I = 1 if input end-of-file error
I = 2 if input parity error

COMMON BLOCKS: BUFFERS, DISKIO, DSKHARD, ERRMESS, ERRNUM, FILENOW, MACHINE, SUPRVIS

SUBROUTINES CALLED: WARNING

CALLED BY: GETDF, INBUFDK

Method

Subroutine INERRDK calls WARNING to write a disk read error message on the output file. Then it calls SEEK* to reposition the current disk file at the beginning of the physical record so that DREAD* can be used to repeat the attempt to read the physical record. If the read cannot be successfully completed in MTIMESR (common /SUPRVIS/) correction attempts, INERRDK writes the message: IRRECOVERABLE READ EOF (or PARITY) ERROR ON DISK FILE to the computer operator and on the standard output file. Then it uses WARNING to write a DISK FILE ABANDONED message on the standard output file. Finally, INERRDK fills each word - corresponding to a word of the physical record which cannot be read - with END FILE or PARITYER depending on the type of error.

Subroutine INERRDK is illustrated in figure 12.

*PFS Disk I/O subroutines

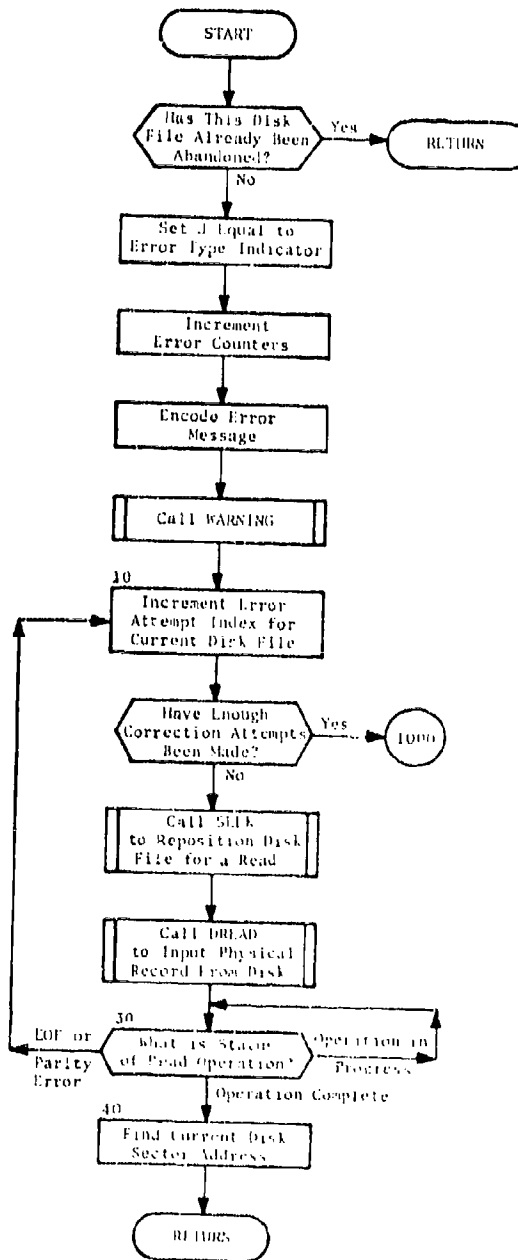


Fig. 12. Subroutine IXERRDK
(Sheet 1 of 2)

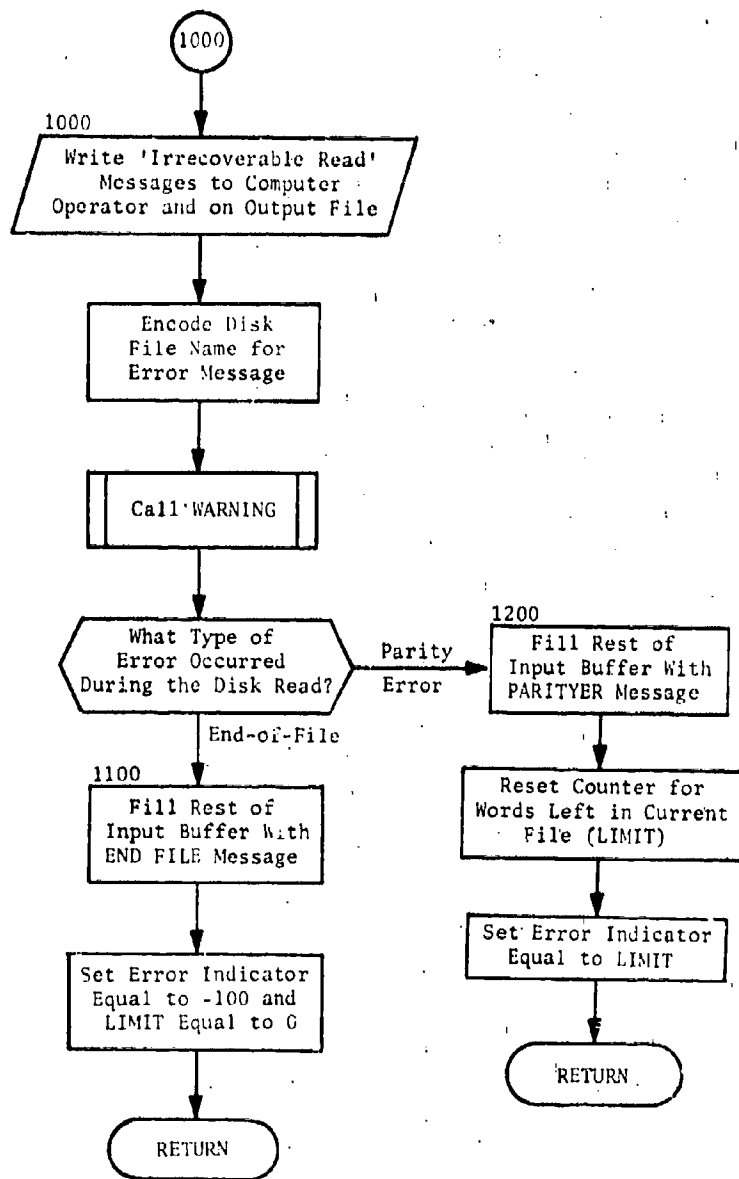


Fig. 12. (cont.)
(Sheet 2 of 2)

SUBROUTINE INLABEL

PURPOSE: To read the filehandler label for each disk file.

ENTRY POINTS: INLABEL

FORMAL PARAMETERS: FLAG - Is a logical variable having the value
.TRUE. if the filehandler label must be
input and .FALSE. otherwise

COMMON BLOCKS: DISKIO, ERRMESS, FILENOW, FILEREC

SUBROUTINES CALLED: INBUFDK, WARNING

CALLED BY: GETDF

Method

Subroutine INLABEL tests the value of FLAG; if it is FALSE, the call to INLABEL was extraneous and INLABEL calls WARNING to write the message EXTRANEIOUS CALL FOR LABEL INPUT on the standard output file. If FLAG is TRUE, INLABEL resets the IEND index to correspond to the last word of the filehandler label and calls INBUFDK to input the label from disk. Then it resets the ISTART and IEND indices as well as the word counter for words read from the disk file so far. Finally it sets FLAG equal to .FALSE..

Subroutine INLABEL is illustrated in figure 13.

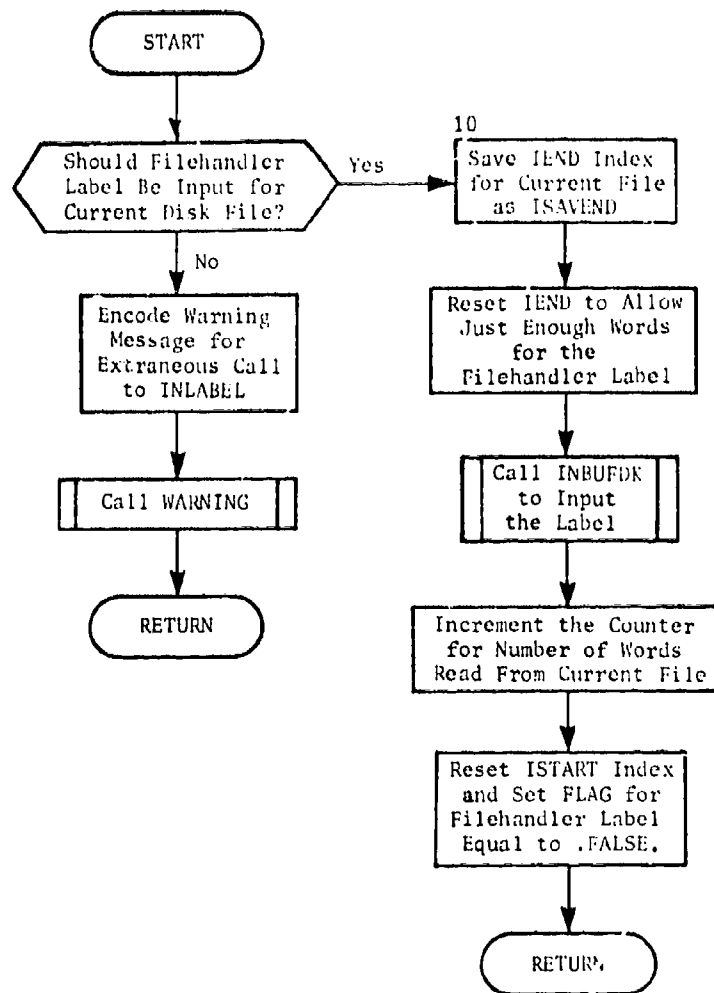


Fig. 15. Subroutine INLABEL

SUBROUTINE NODIRC

PURPOSE: To print an error message and abort the run if program OUTFILE (or RELOADF) is unable to read the master directory, common /DRC5162/, or if RELOADF is unable to read the major directory, common /LOCATOR/.

ENTRY POINTS: NODIRC

FORMAL PARAMETERS: None

COMMON BLOCKS: DRC5162, ERRMESS, MACHINE, SUPRVIS

SUBROUTINES CALLED: ABORT

CALLED BY: OUTFILE (or RELOADF)

Method

Subroutine NODIRC writes the message UNABLE TO READ DIRECTORY OF ACTIVE FILES to the computer operator and dumps the contents of the current directory. Then it calls ABORT to abort the run.

Subroutine NODIRC is illustrated in figure 14.

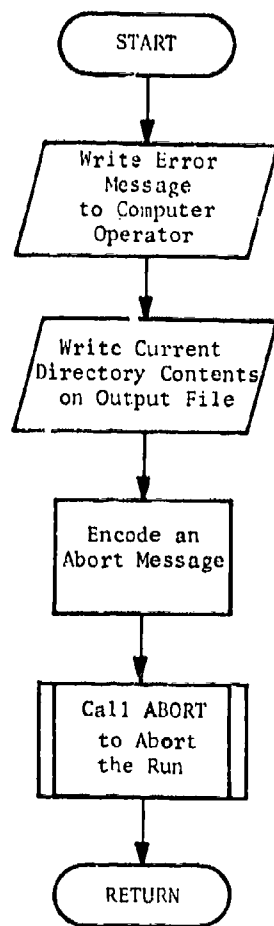


Fig. 14. Subroutine NODIRC

SUBROUTINE OPENSPL

PURPOSE: To set up a spill tape for writing information from disk file(s).

ENTRY POINTS: OPENSPL

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DICTARY, ERRNUM, FILENOW, LOCATOR, MACHINE, SAVWRIT, SUPRVIS, TAPIHARD

SUBROUTINES CALLED: OUTERTP

CALLED BY: OUTFILE

Method

Subroutine OPENSPL determines how many of the current disk files not yet written on spill tapes will fit on this spill tape. Then it determines how many records on the tape should be written to contain the disk file information. The length of each record is the length of the input/output buffer. During these determinations, it sets the values of the parameters which are used during the tape writing process. (These parameters are contained in common block /DICTARY/.)

As soon as these tape write parameters have been calculated, OPENSPL determines the logical unit number for the spill tape and rewinds the tape if it is not the first spill tape to be written. Then it writes the minor directory (common /DICTARY/) on the tape and also on the output file. If the write operation to the spill tape fails because of an end-of-tape or parity error, OPENSPL calls OUTERTP with the argument equal to 2 to repeat that operation until it is successful or impossible to complete. In the latter case, the run is aborted. Finally, SEEK* is called to reposition the disk read head for input.

Subroutine OPENSPL is illustrated in figure 15.

*PFS Disk I/O subroutine

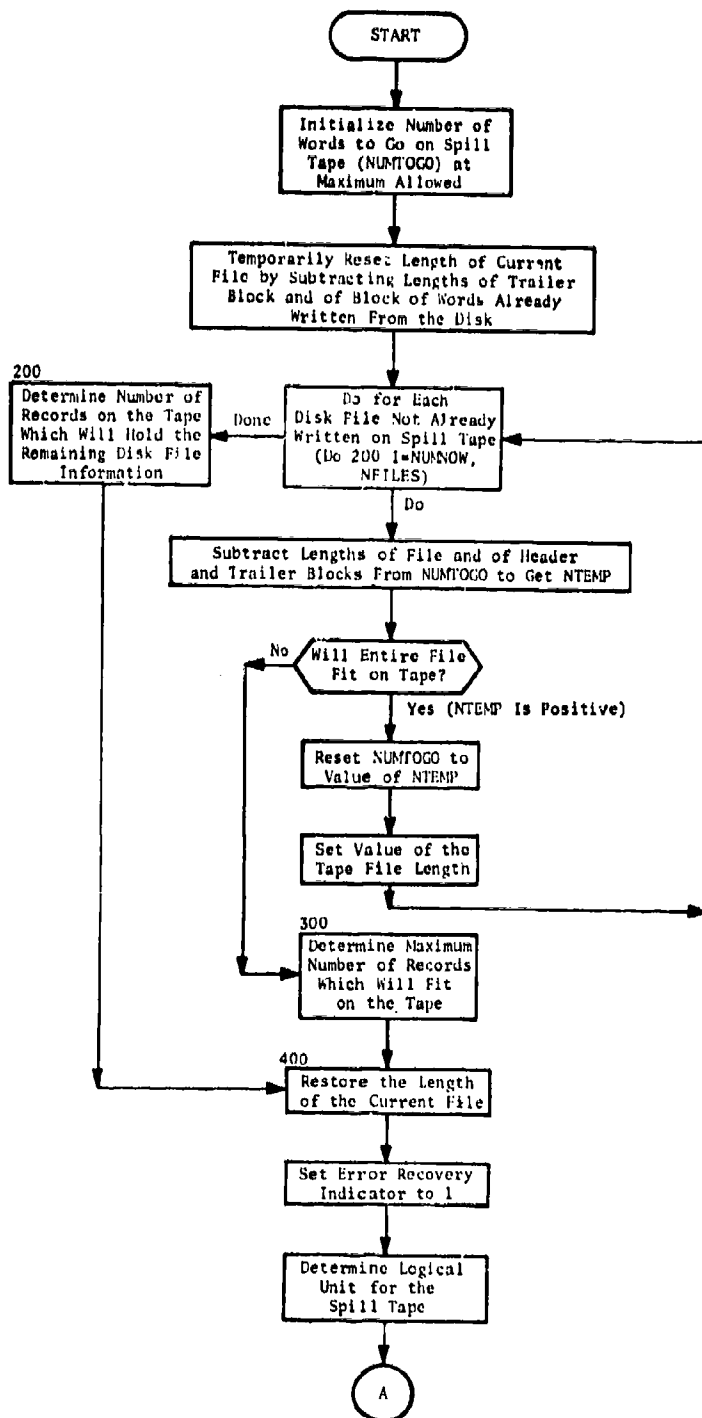


Fig. 15. Subroutine OPENSPL
(Sheet 1 of 2)

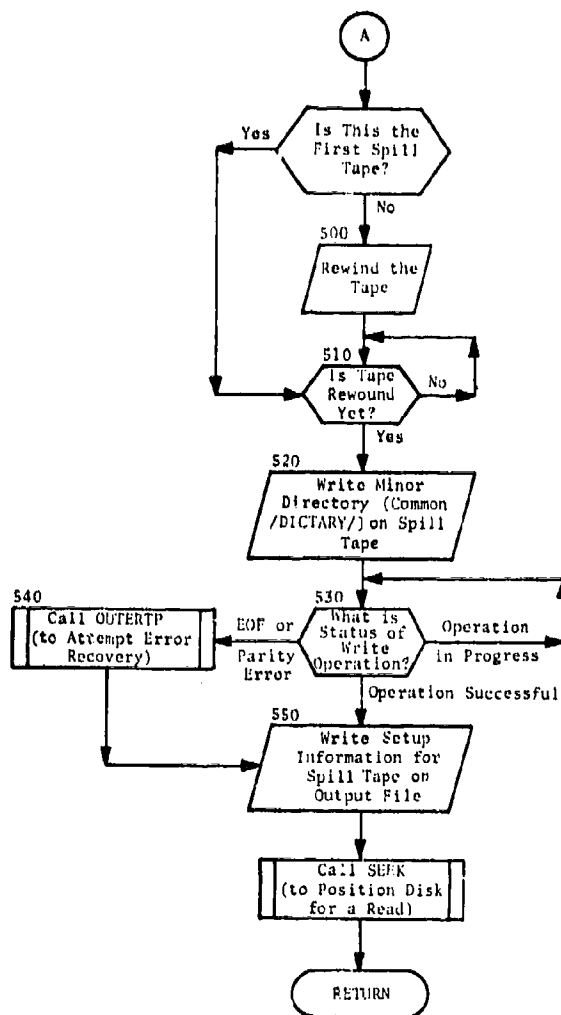


Fig. 15 (cont.)
(Sheet 2 of 2)

SUBROUTINE OUTBFTP

PURPOSE: To buffer out the information in the output buffer to the current spill tape.

ENTRY POINTS: OUTBFTP

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DICTARY, SUPRVIS, TAPHARD

SUBROUTINES CALLED: OUTERTP

CALLED BY: OUTFILE

Method

Subroutine OUTBFTP first checks to determine whether the previous buffer out operation to the spill tape is completed. If this operation resulted in an end-of-tape or parity error, OUTBFTP calls OUTERTP with argument equal to 3 to repeat the operation until it is successful or until too many rewrite attempts cause the run to be aborted. When the previous buffer out operation has been successfully completed, OUTBFTP buffers out the current information in the output buffer to the spill tape.

Subroutine OUTBFTP is illustrated in figure 16.

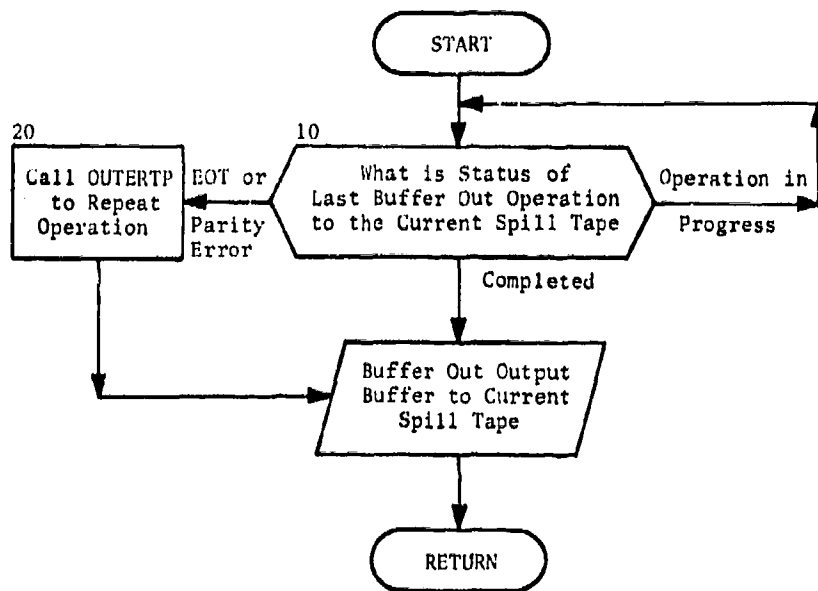


Fig. 16. Subroutine OUTBFTP

SUBROUTINE OUTERTP

PURPOSE: To repeat a tape write operation when an end-of-tape or parity error develops, or to abort the run if the write operation cannot be successfully completed.

ENTRY POINTS: OUTERTP

FORMAL PARAMETERS:

- I - Is used to determine the nature of the information being written on tape when the error developed
 - I = 1 if the major directory (common /LOCATOR/) was being written on the first spill tape
 - I = 2 if the minor directory (common /DICTARY/) was being written on the current spill tape
 - I = 3 if the current output buffer was being written on the current spill tape

COMMON BLOCKS: BUFFERS, DICTARY, ERRMESS, ERRNUM, FILENOW, LOCATOR, MACHINE, SAVWRIT, SUPRVIS, TAPHARD

SUBROUTINES CALLED: ABORT, ERAZE, NEWUNIT, SKIPFILE, WARNING

CALLED BY: CLOSPIL, OPENSPL, OUTBFTP, OUTFILE

Method

First subroutine OUTERTP calls WARNING to write an error warning message on the output file. Then it determines whether the major directory, the minor directory, or the output buffer was being written on the spill tape when the end-of-tape or parity error occurred. In each of the three cases OUTERTP positions the tape at the beginning of the record which was being written, calls ERAZE to erase six inches of tape and repeats the buffer out operation which was previously unsuccessful.

If the error recurs even after MTIMESR (common /SUPERVIS/) correction attempts, the tape unit is unloaded and the unit released by calling NEWUNIT and a fresh tape is mounted on a different tape unit; the disk pointers are then reset for the beginning of the disk file. (The first attempt to write on the new tape is made by the calling subprogram; thus, OUTERTP does not write on the tape unless errors are again encountered.)

If the error cannot be corrected after changing the tape NTIMTAP (common /ERRNUM/) times, it is considered irrecoverable; in this case appropriate messages recording the nature and number of errors are written to the computer operator and to the standard output file, after which OUTERTP calls ABORT to abort the run.

Similarly, under those circumstances where all the available tape units have been released because of errors, error messages are written to the output file and the run is aborted by a call to ABORT.

Subroutine OUTERTP is illustrated in figure 17.

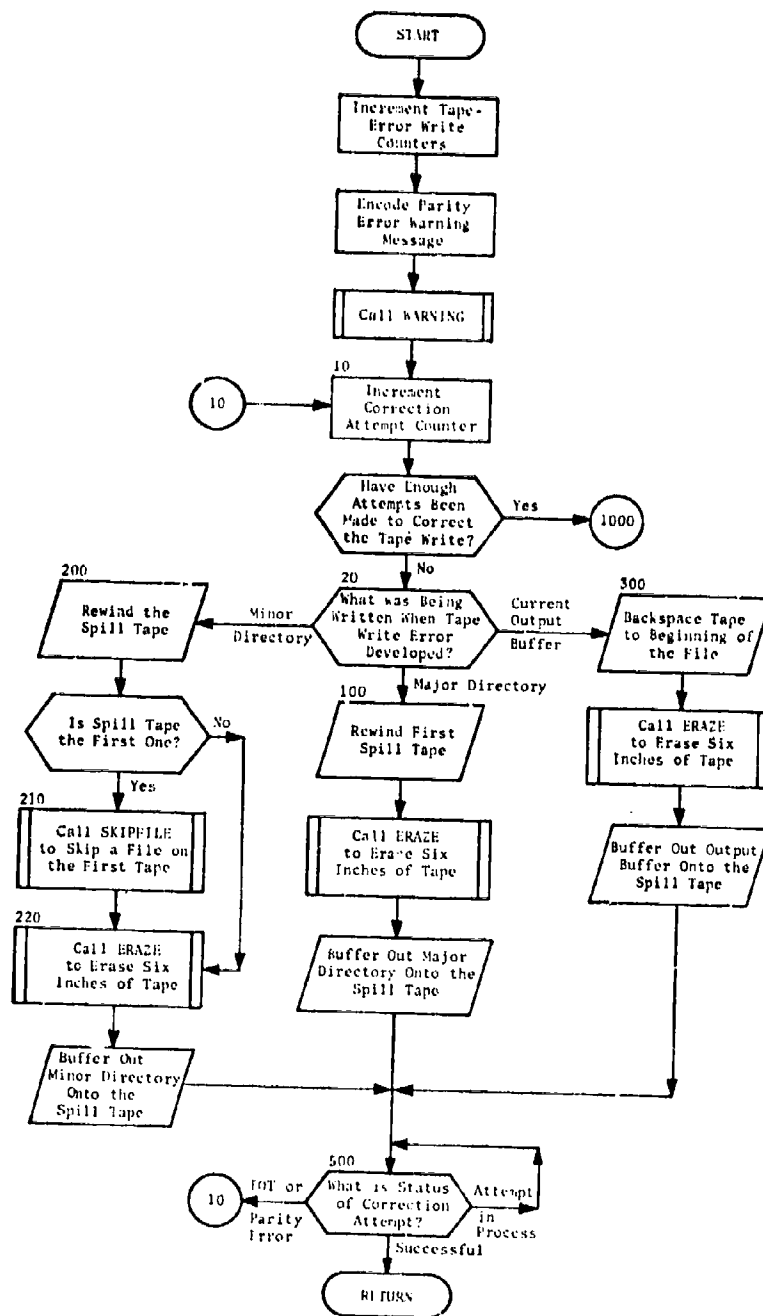


Fig. 17. Subroutine OUTERTP
(Sheet 1 of 2)

SUBROUTINE SETHEAD

PURPOSE: To prepare data for the major directory (common /LOCATOR/ before OUTFILE writes it out onto the first spill tape.

ENTRY POINTS: SETHEAD

FORMAL PARAMETERS: None

COMMON BLOCKS: DRC5162, ERRMESS, FILEREC, LOCATOR, MACHINE, SUPRVIS, TAPHARD

SUBROUTINES CALLED: WARNING

CALLED BY: OUTFILE

Method

Subroutine SETHEAD first determines whether the number of disk files to be dumped on spill tape exceeds the maximum, MAXFILE, which can be handled by OUTFILE. If it does, WARNING is called to write the message TOO MANY FILES TO SPILL on the output file; in this case OUTFILE will dump only the first MAXFILE files.

Finally, SETHEAD determines the values of the variables in the major directory (common /LOCATOR/).

Subroutine SETHEAD is illustrated in figure 18.

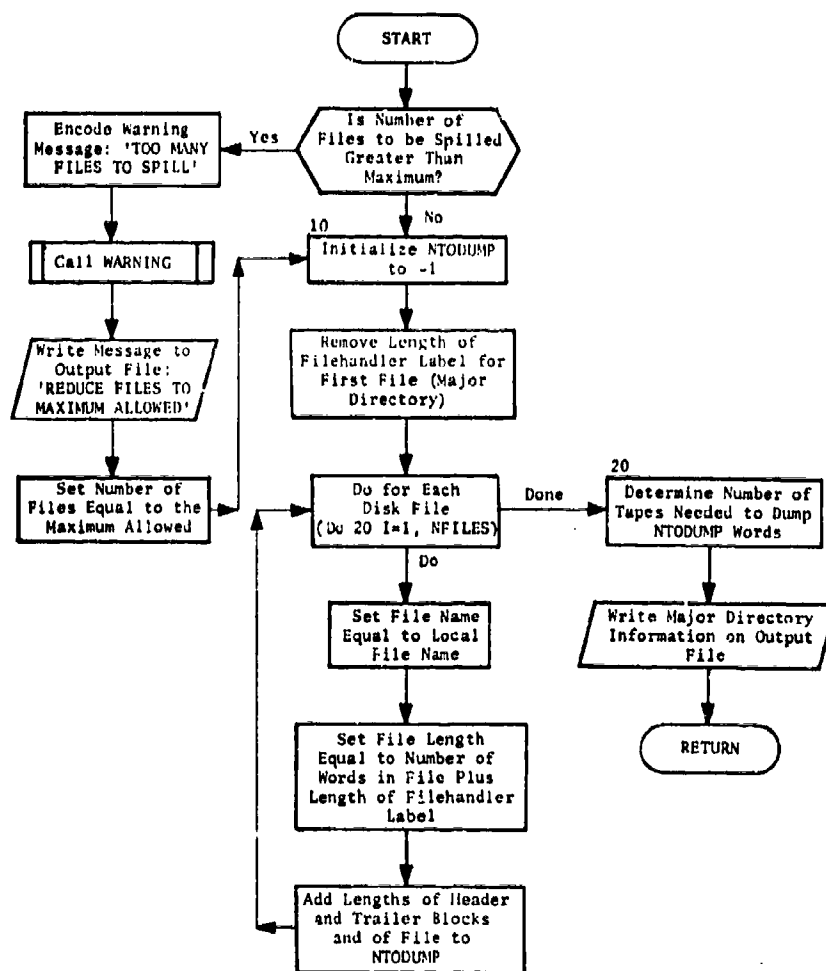


Fig. 18. Subroutine SETHEAD

PROGRAM RELOADF

PURPOSE: To reload disk files from the spill tapes produced by program OUTFILE so that QUICK subsystem processing can be continued or repeated.

ENTRY POINTS: RELOADF

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DICTARY, DISK10, DRC5162, DSKHARD, ERRMESS, ERRNUM, FILENOW, FILEREC, LOCATOR, MACHINE, SAVWRIT, SUPRVIS, TAPHARD

SUBROUTINES CALLED: CLRCMON,* ENDTAPE, GETDATE, GETLOC, INBUFTP, NEXTAPE, NODIRC,* OUTDF, TIMEDAY

Method

Program RELOADF first calls CLRCMON to initialize the variables in all the common blocks. Then, after writing an output header (using TIMEDAY to obtain the time and GETDATE to obtain the date) on the output file and the message file for the computer operator, it calls GETLOC to input the major directory, /LOCATOR/, from the first spill tape. (GETLOC calls ABORT to abort the run if this directory cannot be read.)

Next, for each spill tape, RELOADF first calls NEXTAPE to input the minor directory, /DICTARY/, from the tape. INBUFTP is called to input the first record from the spill tape. Then, for each of the remaining spill tape records, the indices of the input and output buffers are switched, INBUFTP is called to fill the current input buffer from the spill tape, and OUTDF is called to set up the disk files by calling OUTBFDK to transfer the data from the output buffer onto the disk file. Finally, the indices of the input and output buffers are switched again, and RELOADF calls ENDTAPE to finish reading the current tape, to release it, and request that another spill tape be mounted (when more than three spill tapes are used); OUTDF is called once more to transfer the remaining data to the disk file(s).

When all files listed in the major directory, /LOCATOR/, have been reloaded, RELOADF calls ENDTAPE to finish reading the current tape and discontinues its processing of the spill tapes.

*Described in program OUTFILE

In all cases, when processing is completed, RELOADF writes tape and disk error counter messages on the standard output file and on the output comment medium and also the message PROCESSOR RELOADF COMPLETED.

The common blocks used by program RELOADF are the same as those previously described for program OUTFILE. See table 3.

Figure 19 shows the flow of operation within program RELOADF.

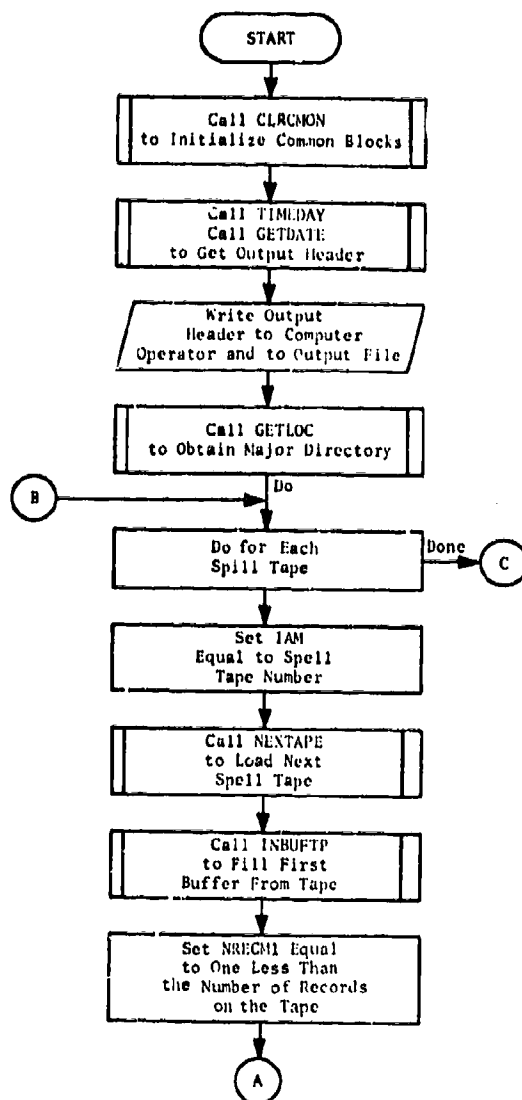


Fig. 19. Program RELOADF
(Sheet 1 of 2)

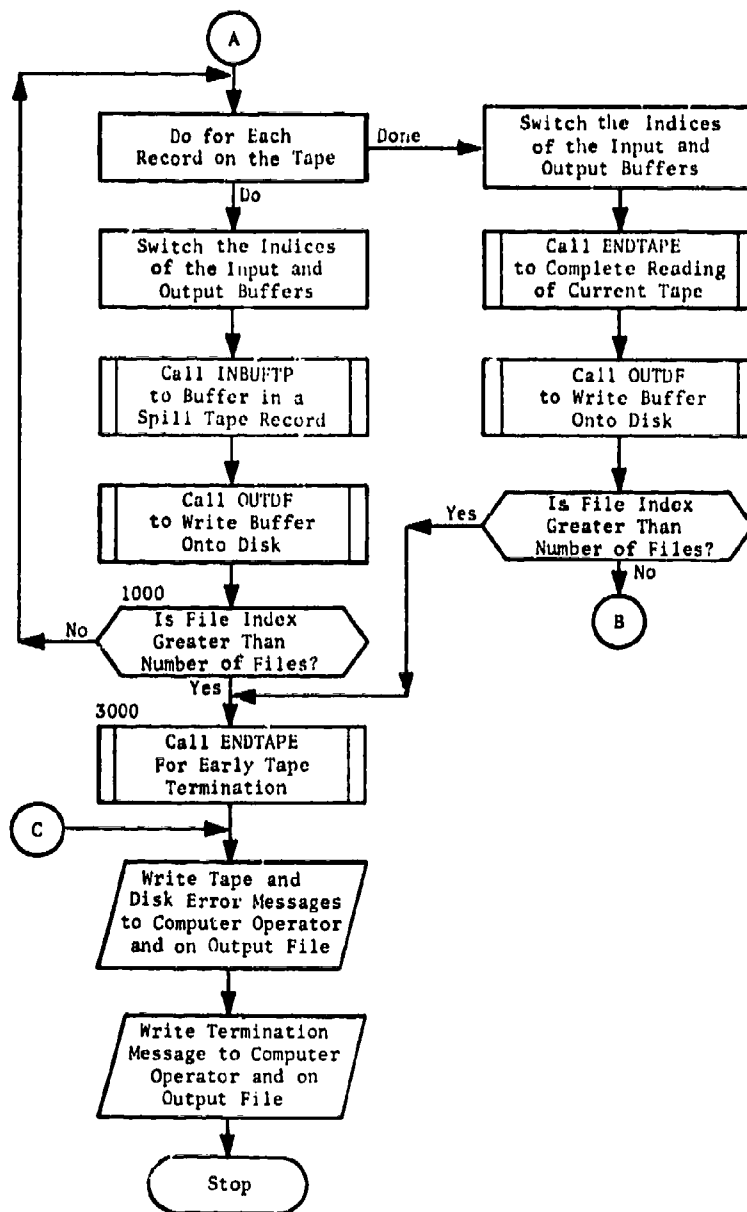


Fig. 19. (cont.)
(Sheet 2 of 2)

SUBROUTINE ENDTAPE

PURPOSE: : complete the reading of the current spill
e.

ENTRY POINTS: ENDTAPE

FORMAL PARAMETERS: None

COMMON BLOCKS: DICTARY, ERRNUM, LOCATOR, MACHINE, SUPRVIS

SUBROUTINES CALLED: INERRTP, ANOTHER

CALLED BY: RELOADF

Method

Subroutine ENDTAPE checks the status of the last tape read operation for the current tape. If the operation terminated with an end-of-file or parity error, it calls INERRTP to attempt to reread the record.

When the tape has been successfully read, ENDTAPE calls ANOTHER to release it and then checks to see if the tape unit will be needed for another spill tape. If so, ENDTAPE instructs the computer operator to mount the next spill tape on the tape unit.

The subroutine returns after printing out error counts which indicate the number of tape read errors encountered on the last tape and on all the tapes read so far.

Subroutine ENDTAPE is illustrated in figure 20.

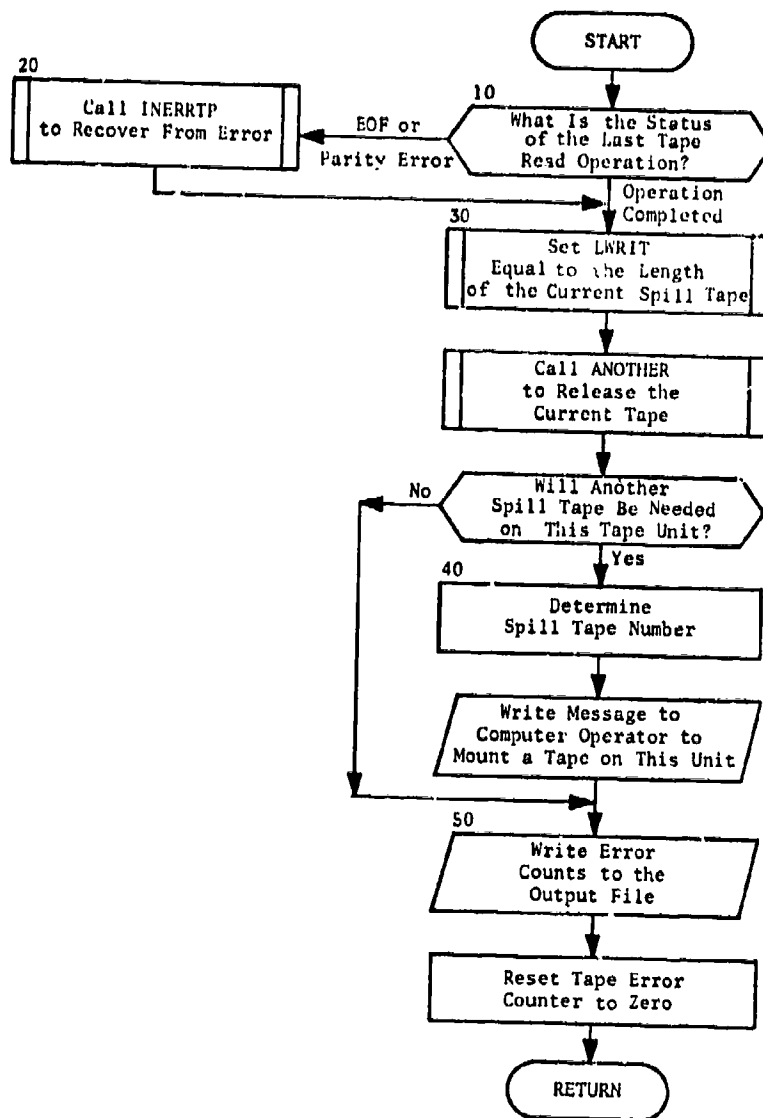


Fig. 20. Subroutine ENDTAPE

SUBROUTINE GETLOC

PURPOSE: To read the major directory to the spill tapes (common /LOCATOR/) from the first spill tape.

ENTRY POINTS: GETLOC

FORMAL PARAMETERS: None

COMMON BLOCKS: DRC5162, ERRMESS, LOCATOR, MACHINE, SUPRVIS, TAPHARD

SUBROUTINES CALLED: NODIRC, ABORT

CALLED BY: RELOADF

Method

Subroutine GETLOC first attempts to buffer in the major directory -- common /LOCATOR/ -- from the first spill tape. If an end-of-file or parity error is encountered during the buffer operation, the tape is rewound and the buffer attempt is repeated. If this buffer process is repeated MTIMESR, common /SUPRVIS/, times without success, GETLOC determines how many words have been read successfully from the tape and places them (the words) into the master directory (common /DRC5162/) so that it can then use NODIRC to print out error information and abort the run.

If, on the other hand, the buffer operation is successful, GETLOC checks the value of MAXFILE (which is the maximum number of disk files contained on the spill tapes) against the value of MAXFILE set earlier in RELOADF. If they are not equal, then either RELOADF has not been updated to correspond to program OUTFILE, or the value read from the tape is in error for some other reason. In either case, the discrepancy is encoded into an error message and ABORT is called to abort the run. The computer operator is also informed of the discrepancy.

In the normal case where the major directory is read successfully and MAXFILE does not have an unexpected value, GETLOC writes the major directory on the standard output file and processing control is returned to RELOADF.

Subroutine GETLOC is illustrated in figure 21.

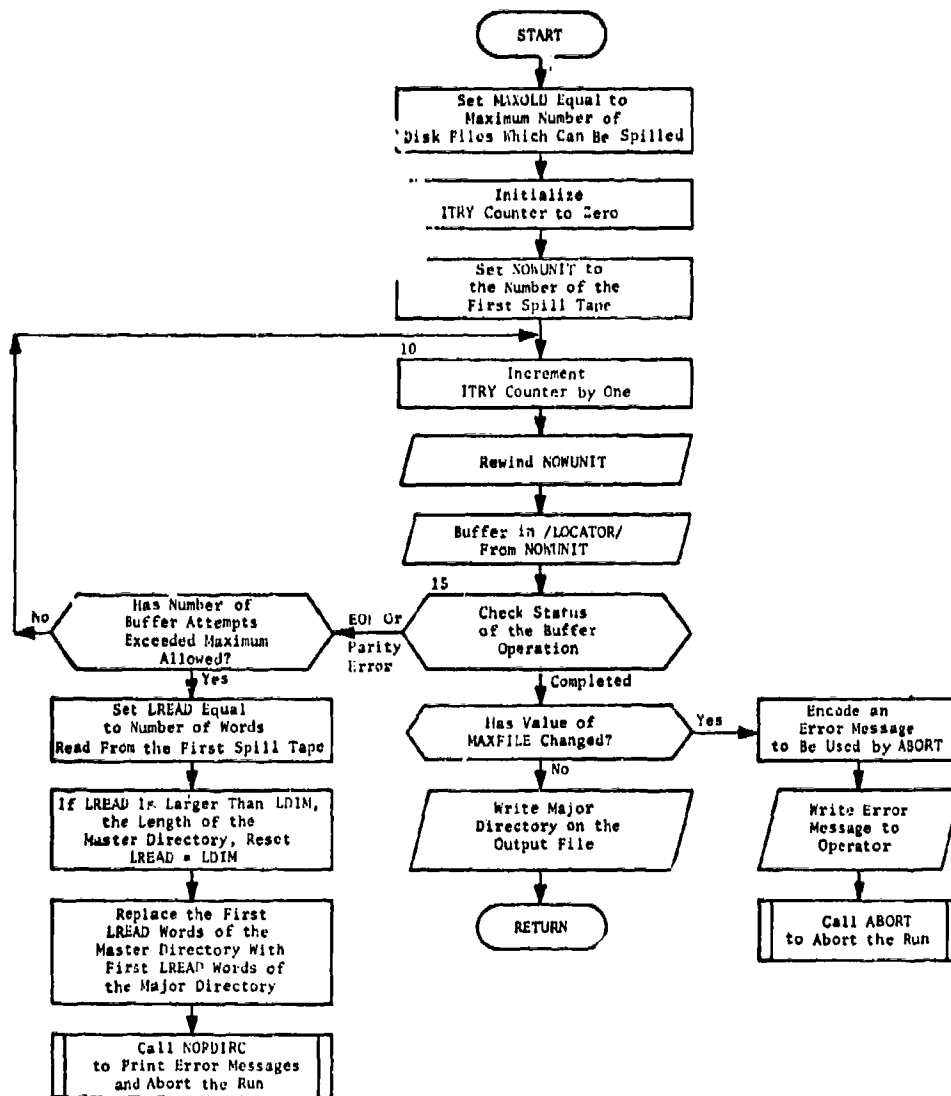


Fig. 21. Subroutine GETLOC

SUBROUTINE INBUFTP

PURPOSE: To buffer in a spill tape record into the input buffer.

ENTRY POINTS: INBUFTP

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DICTARY, SUPRVIS, TAPHARD

SUBROUTINES CALLED: INERRTP

CALLED BY: RELOADF

Method

After setting local variable I equal to the index for the current input buffer, INBUFTP checks the status of the last buffer in operation for the current spill tape. If the last operation ended with an end-of-file or parity error, INBUFTP calls INERRTP (with argument=3) to repeat the operation until it succeeds or until the error is encountered at least MTIMESR, common /SUPRVIS/, times; in the latter case, the unfilled portion of the buffer is filled with the word PARITYER.

Whether the preceding operation ended successfully or not, processing control returns to RELOADF after INBUFTP first saves the length of the last record read in LWRT, common /SUPRVIS/, and then buffers in the next tape record into the input buffer.

Subroutine INBUFTP is illustrated in figure 22.

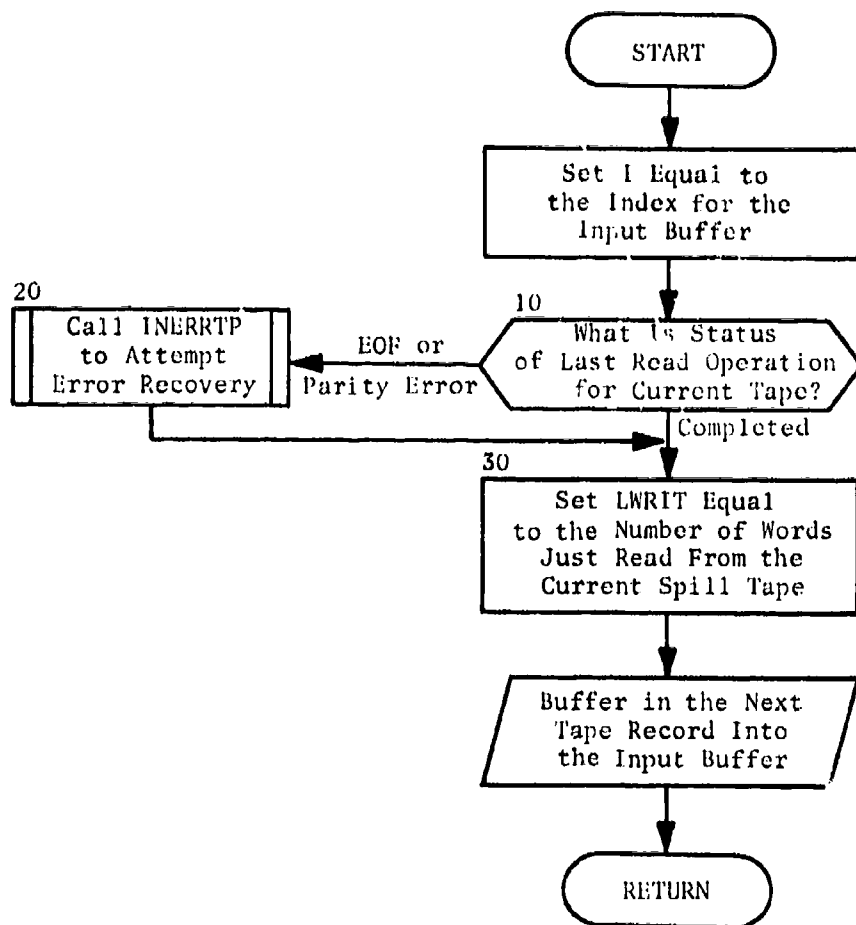


Fig. 22. Subroutine INBUFTP

SUBROUTINE INERRTP

PURPOSE: To repeat a tape read operation which terminated with an end-of-file or parity error in an effort to correct it.

ENTRY POINTS: INERRTP

FORMAL PARAMETERS: I - An indicator which reveals the nature of the information being read from tape when the error occurred
I = 2 if the minor directory was being read from the tape
I = 3 if the current input buffer was being filled

COMMON BLOCKS: BUFFERS, DICTARY, DSKHARD, ERRMESS, ERRNUM, FILENOW, LOCATOR, MACHINE, SUPRVIS, TAPHARD

SUBROUTINES CALLED: WARNING

CALLED BY: ENDTAPE, INBUFTP, NEXTAPE

Method

After encoding the error message READ PARITY ERROR OR END OF FILE ON UNIT ____, INERRTP calls subroutine WARNING to output the message and other error information. Then it attempts to repeat the buffer in operation which caused the error. First, it backspaces the tape to the beginning of the record; if the current tape record is to be reread, each word of the input buffer is first filled with the message PARITYER. Then depending on whether the formal parameter I is 2 or 3, the minor directory (common /DICTARY/) or the current tape record is buffered in from the current spill tape. In the latter case, the words input from the current tape record replace the words PARITYER in the input buffer if the operation is successful.

In either case if the "buffer in" operation is not successful, the tape-read error counter is incremented and the process described above is repeated until it is successful or the number of reread attempts exceeds the value of the variable MTIMESR in common /SUPRVIS/.

If MTIMESR attempts have been made to correct the read, and if end-of-file or parity errors are still encountered, INERRTP encodes the message UNABLE

TO READ MINOR DIRECTORY ON UNIT ____ if I is 2 or IRRECOVERABLE READ ERROR ON SPILL TAPE ____ ON UNIT ____ if I is 3; then it calls WARNING. Further, if I is 2, INERRTP determines whether the variable specifying the maximum number of records on a tape (NREC) has been changed by the flawed read operation and whether the number of disk files on the spill tape (NUMONME in /DICTARY/) is less than or equal to MAXFILE. If either of these conditions is not satisfied, NREC or NUMONME, respectively, is reset. When I is 3, INERRTP determines the starting and ending addresses in the current disk file being written for the current tape record and writes the information on the standard output file.

In all cases, before control is returned to the calling subprogram, INERRTP resets the value of IAMOK, the error recovery indicator.

Subroutine INERRTP is illustrated in figure 23.

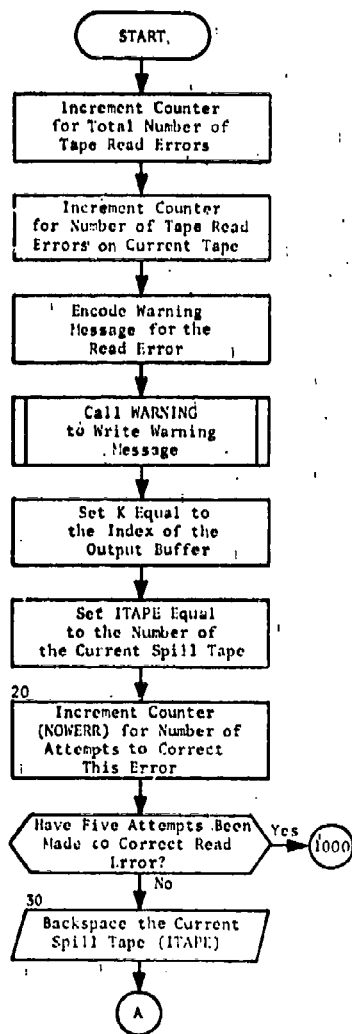


Fig. 23. Subroutine INERRTP
(Sheet 1 of 3)

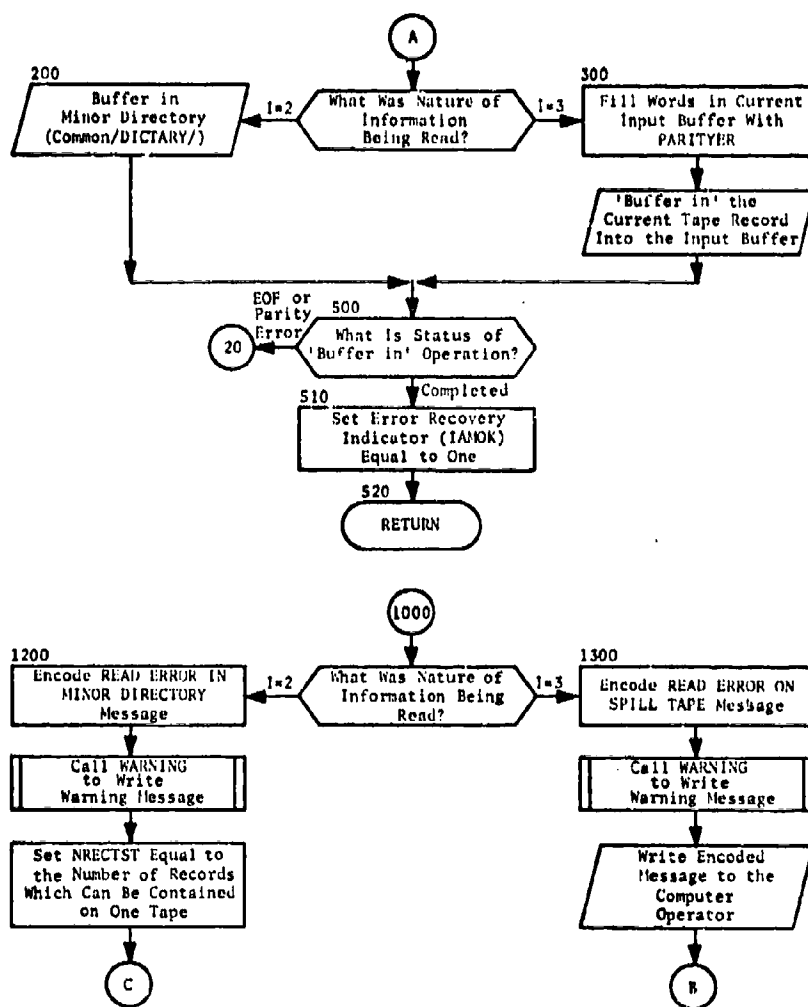


Fig. 23. (cont.)
(Sheet 2 of 3)

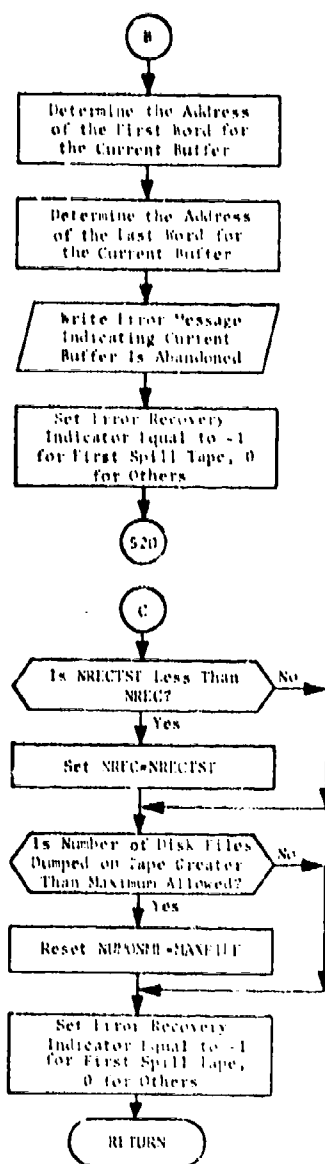


Fig. 25 . (cont.)
(Sheet 3 of 3)

SUBROUTINE NEXTAPE

PURPOSE: To read the minor directory from the next spill tape.

ENTRY POINTS: NEXTAPE

FORMAL PARAMETERS: None

COMMON BLOCKS: DICTARY, LOCATOR, MACHINE, SUPRVIS, TAPHARD

SUBROUTINES CALLED: INERRTP, SKIPFILE

CALLED BY: RELOADF

Method

After determining the logical unit number for the next spill tape (NOWUNIT), and saving the spill tape number as IAMSAVE, NEXTAPE rewinds NOWUNIT and uses SKIPFILE to skip the first file if the first spill tape is to be read (because the first file of the first spill tape contains the major tape directory). Then NEXTAPE buffers in the minor directory (common /DICTARY/) from the spill tape. INERRTP is called (with argument=2) to repeat the buffer operation in the event an end-of-file or parity error is encountered during this process.

When the minor directory has been read from the tape, the actual spill tape number (IAM) and label (IDENT) are known, so NEXTAPE compares these values with those it expected. If the values of IAM and IDENT are not the expected ones, and if the computer operator has not been instructed to ignore mismatches in tape labels (via sense switch 1), NEXTAPE then instructs the operator to mount the requested tape.

Finally, when the minor directory has been read satisfactorily, NEXTAPE writes it on the standard output file and resets IAM to IAMSAVE, the expected spill tape number.

Subroutine NEXTAPE is illustrated in figure 24.

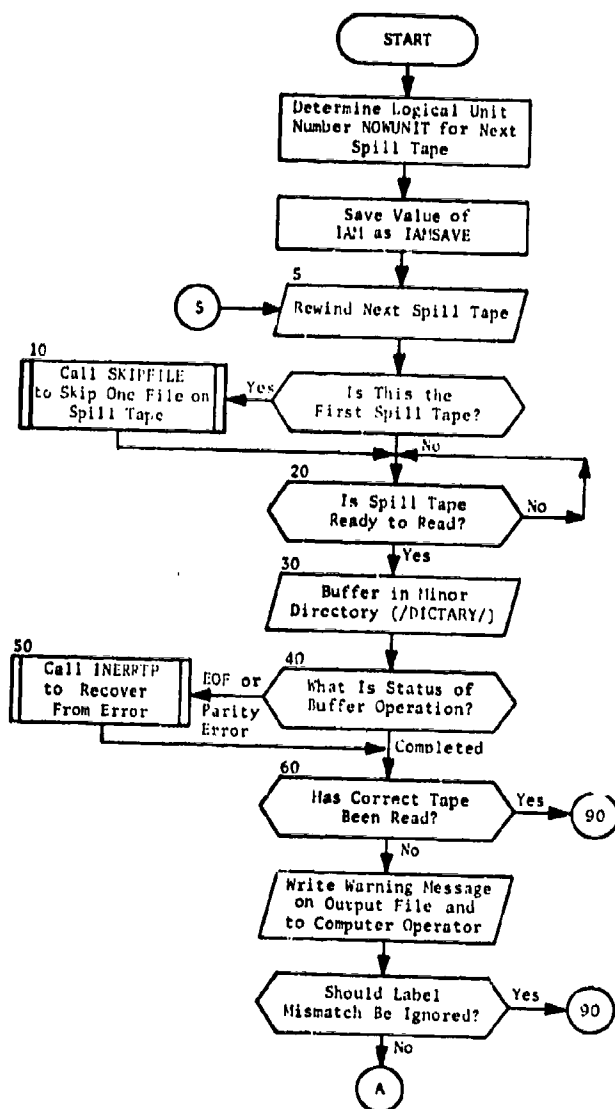


Fig. 24. Subroutine NEXTAPE
(Sheet 1 of 2)

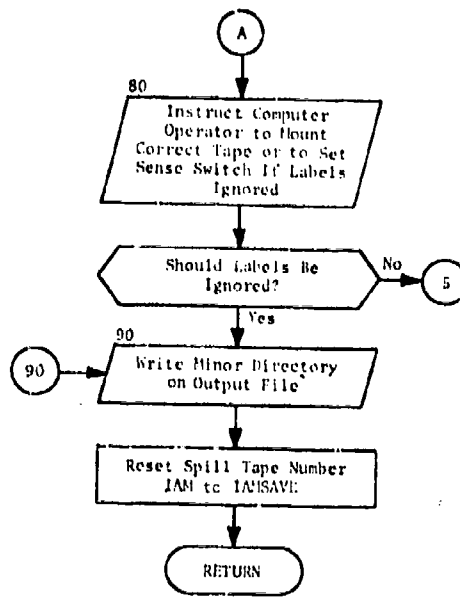


Fig. 24. (cont.)
(Sheet 2 of 2)

SUBROUTINE OUTBFDK

PURPOSE: To buffer out information from the output buffer to disk.

ENTRY POINTS: OUTBFDK

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DISKIO, DSKHARD, ERRNUM, FILENOW, FILEREC, SUPRVIS

SUBROUTINES CALLED: OUTERDK

CALLED BY: OUTDF

Method

Subroutine OUTBFDK first determines whether the last disk write operation terminated with an irrecoverable end-of-file (EOF) or parity error. If it did, processing control returns to the calling subprogram. Otherwise, the status of the last operation is checked. If it ended with an EOF or parity error, OUTERDK is called to attempt recovery from the error. Again, if the error is irrecoverable, processing control returns to the calling subprogram, OUTDF.

As soon as the previous disk write operation is successfully completed, OUTBFDK saves the current disk sector address and determines the number of physical records to be written on the disk. It also initializes the indices ISTREC and IENDREC, which locate the record in the IOBUFF array. Then it uses WRITECK* to transfer each disk physical record from the IOBUFF array onto the disk; each write operation is checked for a possible parity or end-of-file error. If one occurs, OUTERDK is called to attempt error recovery. Finally, processing control returns to the calling subprogram.

Subroutine OUTBFDK is illustrated in figure 25.

*PFS Disk I/O subroutine

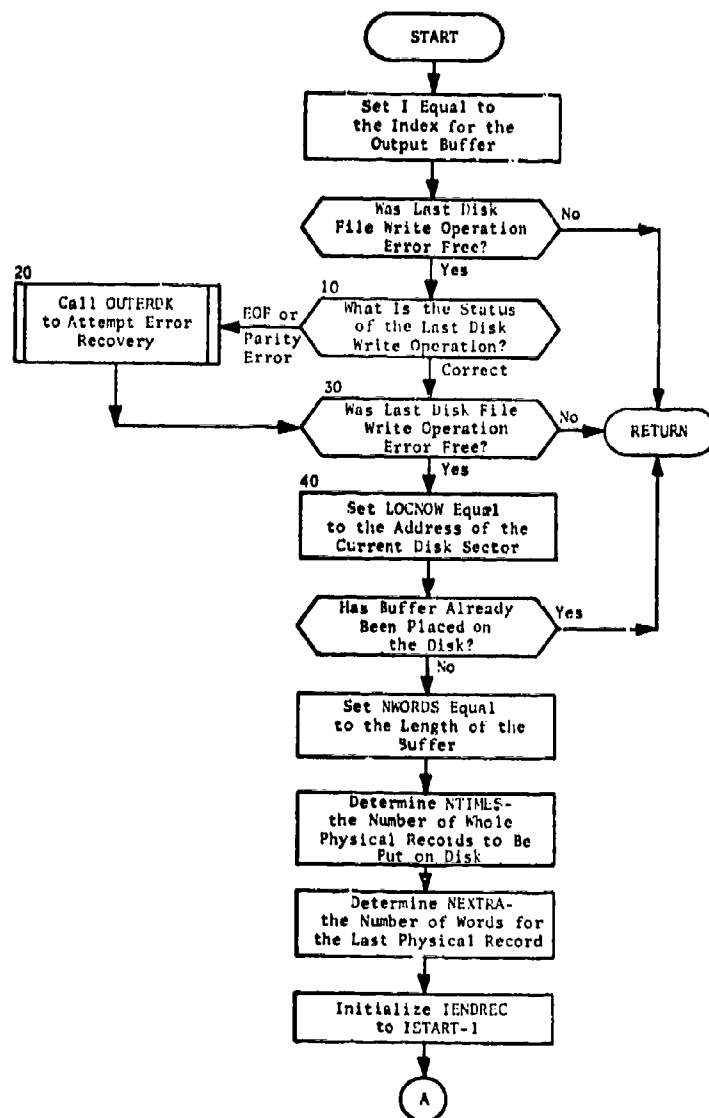


Fig. 25. Subroutine OUTBFDK
(Sheet 1 of 2)

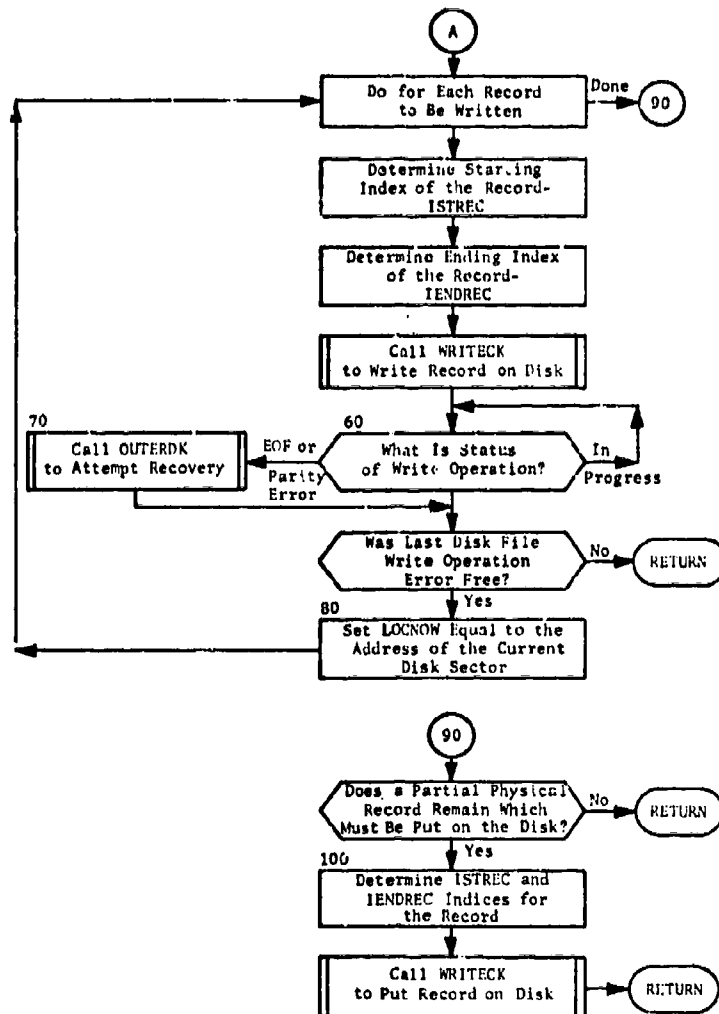


Fig. 25. (cont.)
(Sheet 2 of 2)

SUBROUTINE OUTDF

PURPOSE: To set up the disk files for output.

ENTRY POINTS: OUTDF

FORMAL PARAMETERS: None

COMMON BLOCKS: BUFFERS, DISKIO, DSKHARD, ERRNUM, FILENOW,
FILEREC, LOCATOR, MACHINE, SUPRVIS

SUBROUTINES CALLED: OUTBFDK, OUTERDK

CALLED BY: RELOADF

Method

Subroutine OUTDF is called once by RELOADF each time the IOBUFF array has been filled from a spill tape. It is used to determine which portions of the disk files should be filled from IOBUFF and to set up the files for output.

Each time a new disk file is to be loaded, OUTDF determines how many tracks will be needed and uses ALLOCATE* and SEEK* to allocate the disk tracks and to position the disk respectively. Then it sets the indices ISTART and IEND so the filehandler label is put out when OUTBFDK is called.

When the array IOBUFF does not begin with the first word of a new disk file, OUTDF first checks the array for trailer words which correspond to the name of the current disk file (NAMENOW). If it finds a trailer label, it sets the indices ISTART, IEND, and NWORDNW in such a way that OUTBFDK writes the remaining portion of IOBUFF corresponding to the current disk file (excluding the trailer label) onto the disk file. If no trailer word is found in the rest of the buffer, OUTDF calls OUTBFDK to put out the rest of the buffer onto the current disk file. Sometimes, IOBUFF will contain data for two or more disk files. When this situation arises, OUTDF first determines which words go onto the first file and uses OUTBFDK to put them out as described above. Then it uses DEOF* to put an end-of-file mark on the current file and checks the header label for the next file. If not all of the header words are in the buffer, no further data are output to disk files until OUTDF is

*PFS Disk I/O subroutines

called again. If the buffer contains the whole header, OUTDF sets up the next new disk file for output and proceeds to empty IOBUFF onto this new file in the manner described above.

Whenever end-of-file or parity errors are encountered by OUTBFDK in writing from IOBUFF onto disk files, OUTDF uses OUTERDK to attempt recovery from those errors. Processing of IOBUFF continues until either all of IOBUFF is on disk or until no more disk files are to be loaded. In the latter case, OUTDF informs the computer operator that the last file has been output to disk. Then processing control returns to the calling program.

Subroutine OUTDF is illustrated in figure 26.

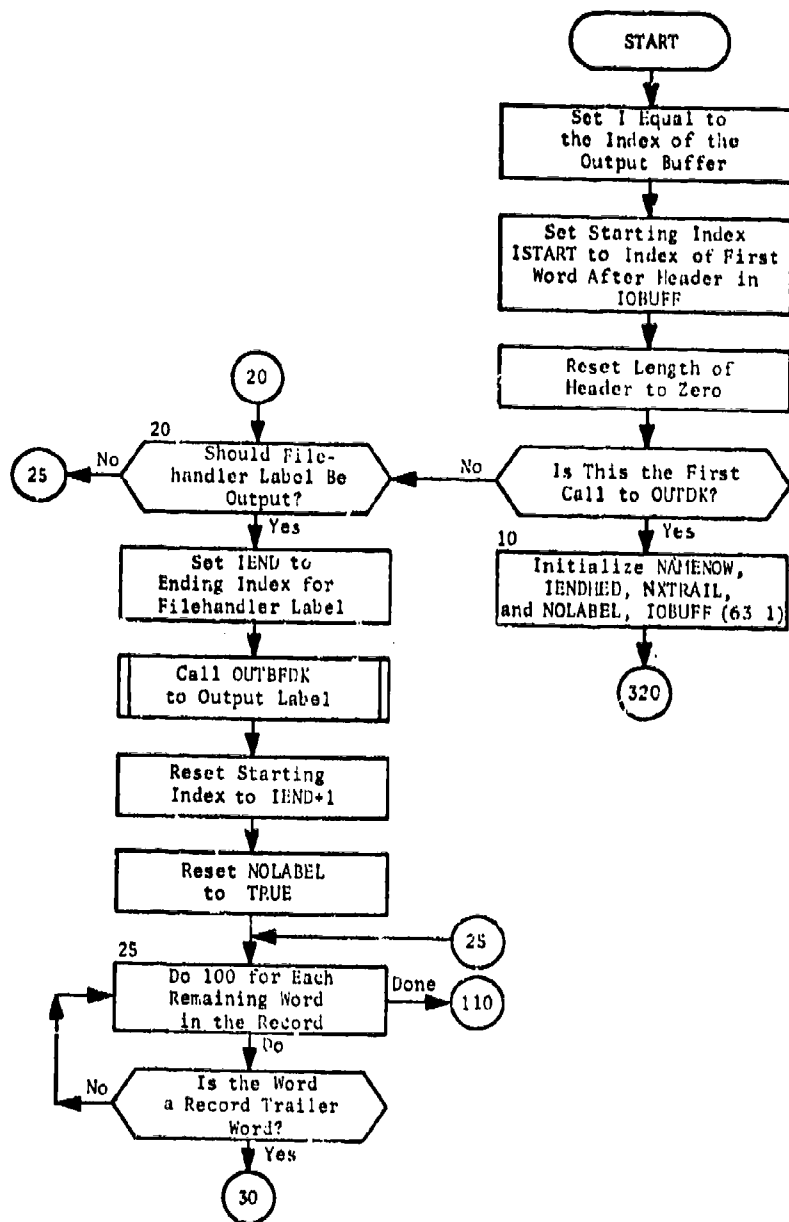


Fig. 26. Subroutine OUTDF
(Sheet 1 of 5)

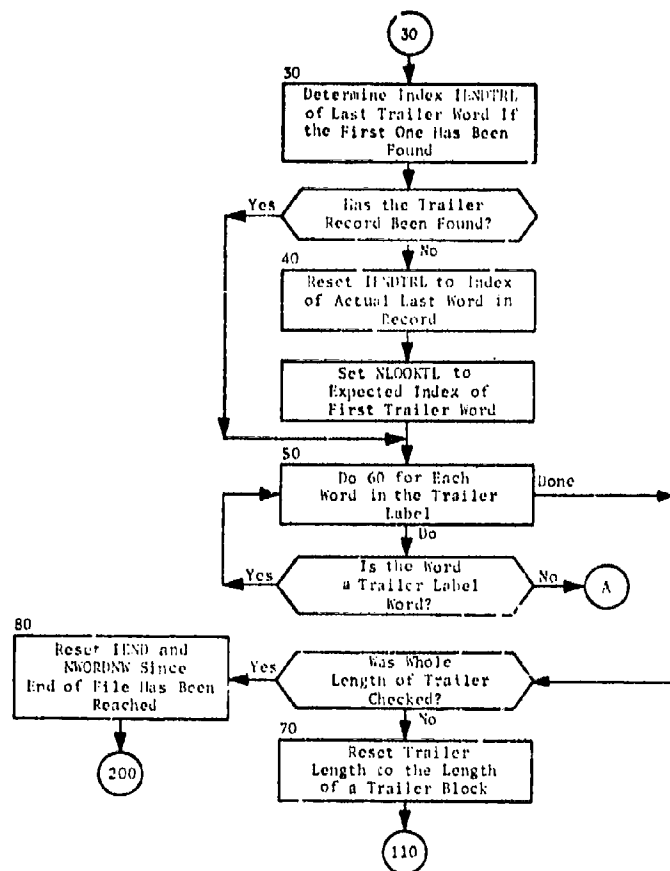


Fig. 26 . (cont.)
(Sheet 2 of 5)

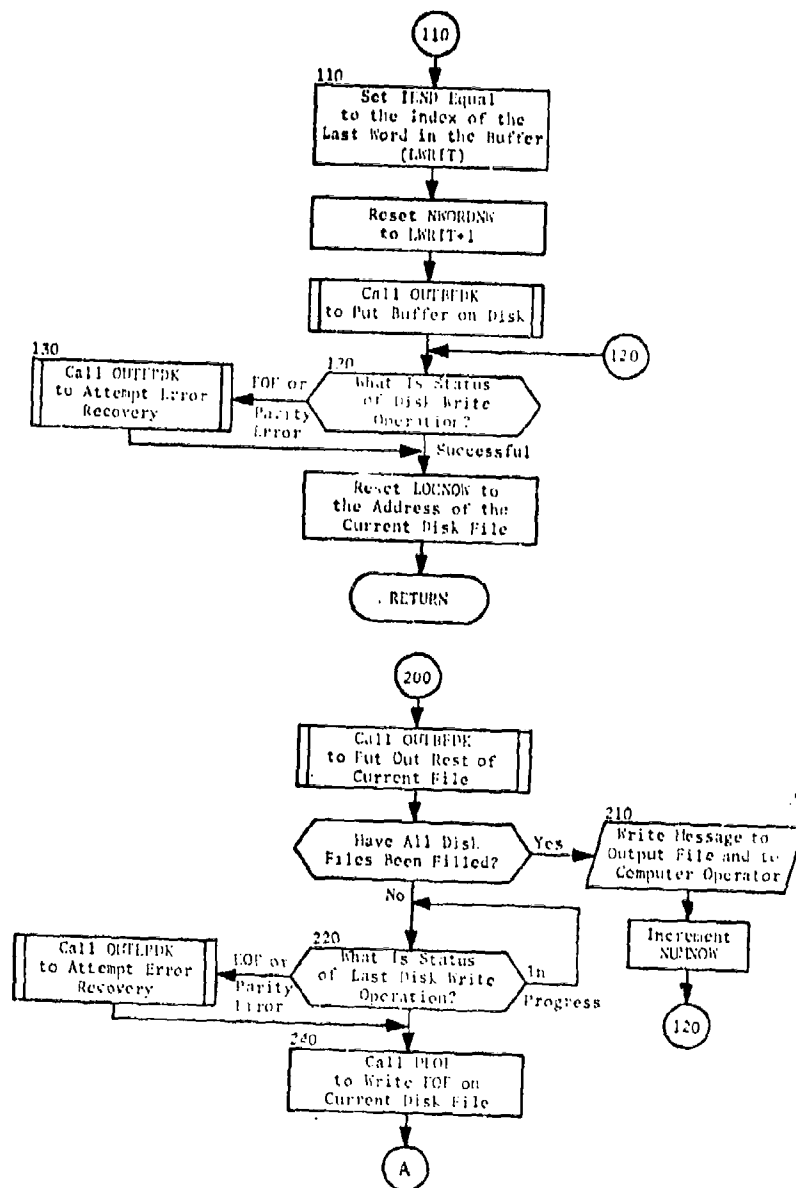


Fig. 26. (cont.)
(Sheet 3 of 5)

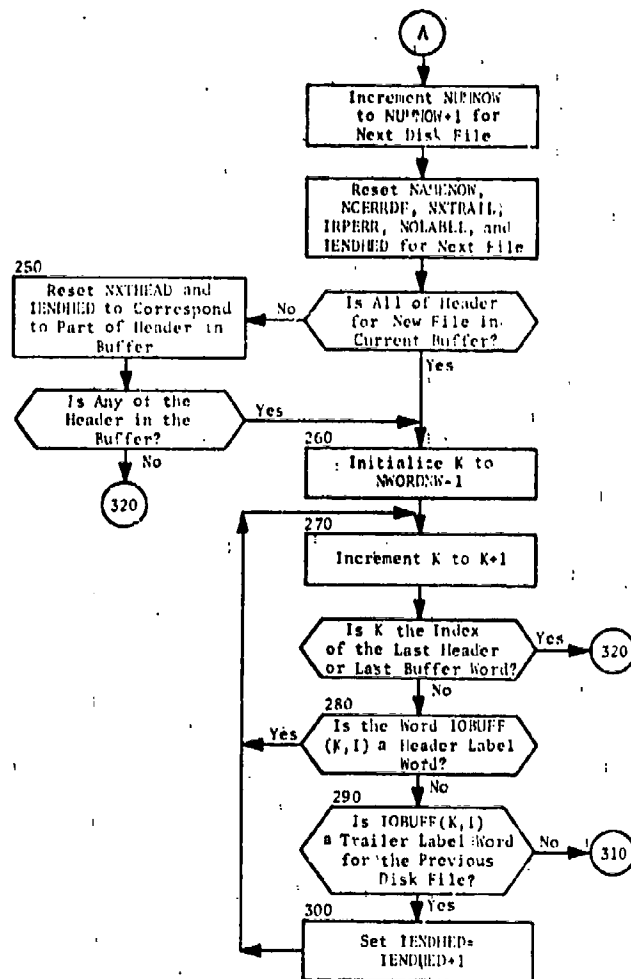


Fig. 26. (cont.)
(Sheet 4 of 5)

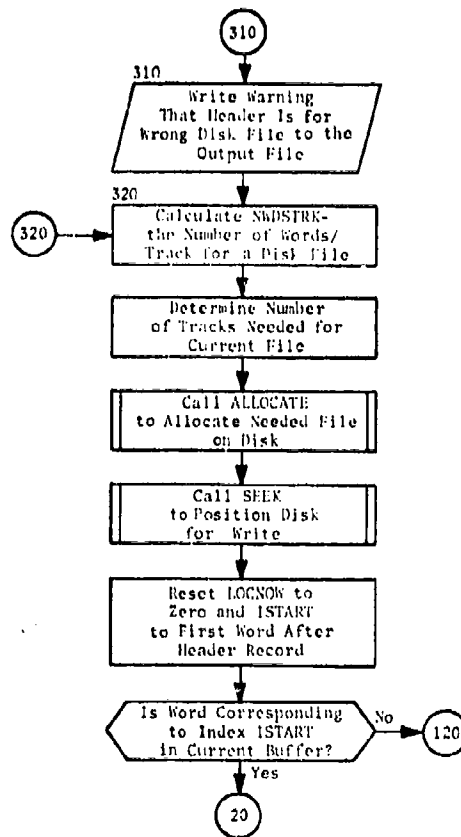


Fig. 26 . (cont.)
(Sheet 5 of 5)

SUBROUTINE OUTERDK

PURPOSE: To repeat a disk file write operation in an attempt to correct an output parity or end-of-file error.

ENTRY POINTS: OUTERDK

FORMAL PARAMETERS: I - Is an error type indicator
I = 1 if end-of-file error
I = 2 if parity error

COMMON BLOCKS: BUFFERS, DISKIO, DSKHARD, ERRMESS, ERRNUM, FILENOW, LOCATOR, MACHINE, SUPRVIS

SUBROUTINES CALLED: WARNING

CALLED BY: OUTBFDK

Method

Subroutine OUTERDK first determines whether the current file (named NAMENOW) already contains an irrecoverable error. If it does, no further attempt is made to correct errors in this file and processing control is returned to OUTBFDK.

Otherwise, error counters are incremented or initialized and a warning message is transmitted to the standard output file via WARNING. Then OUTERDK calls SEEK* to reposition the disk and WRITECK* to repeat the attempt to write the record onto the disk. This procedure is repeated until the write attempt is successful or until MTIMESW (common /SUPRVIS/) recovery attempts have failed. In the first case, the error recovery indicator IRPERR (common /ERRNUM/) is set to zero. In the second case, messages concerning the number of attempts made to correct the write error are written on the standard output file and the computer operator is informed that there is an irrecoverable write error on the disk file and that it is being abandoned. Finally, in this second case, the error recovery indicator IRPERR is set to -100 if the error type is end-of-file and to the number of words left in the file if the error type is parity.

Subroutine OUTERDK is illustrated in figure 27.

*PFS Disk I/O subroutines

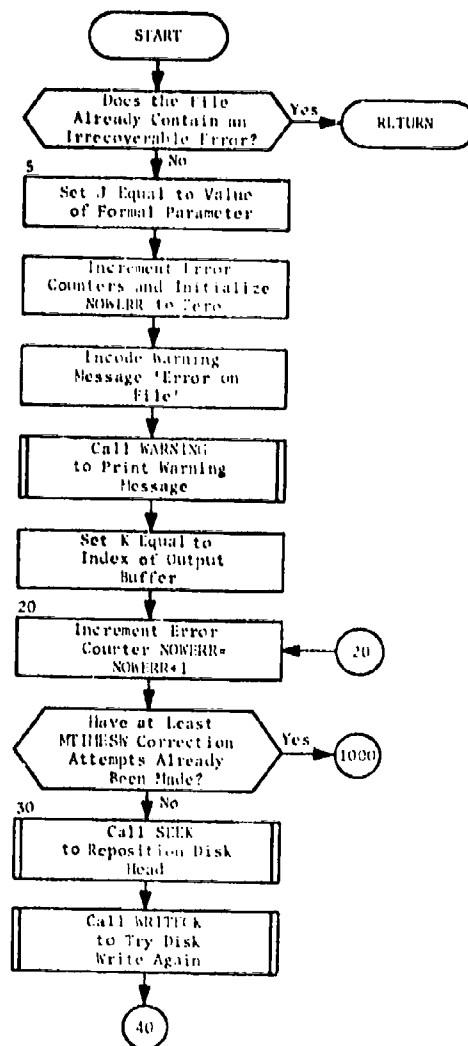


Fig. 27. Subroutine OUTERDK
(Sheet 1 of 2)

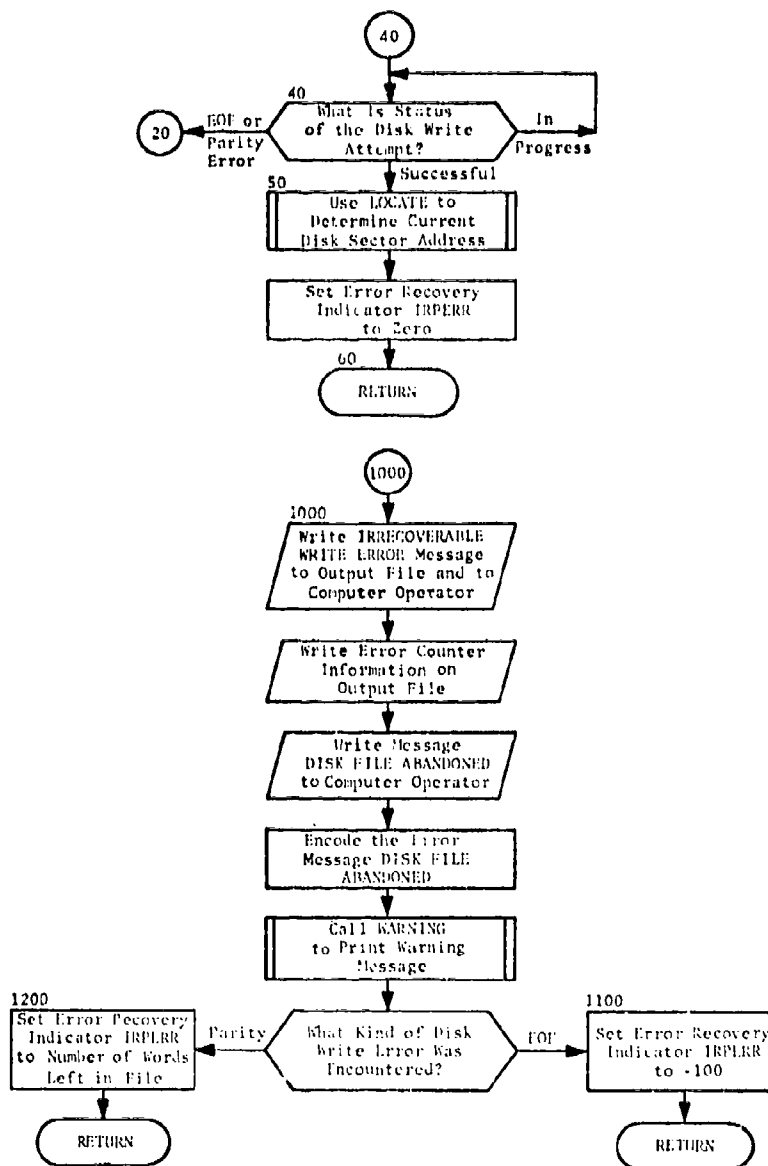


Fig. 27. (cont.)
(Sheet 2 of 2)

PROGRAM DECLARES

PURPOSE: DECLARES is a computer program processor which is used to insert the FORTRAN common, equivalence, and type statements into programs which process QUICK data base tapes.

ENTRY POINTS: DECLARES

FORMAL PARAMETERS: None

COMMON BLOCKS: OUTFILES, DIRECTORY, XPRT

SUBROUTINES CALLED: ALOCDIR, EQUIV, FILEINR, NUMGET, READDIR, TERMTAPE

Method

Program DECLARES is a tool designed to aid the NMCSSC analyst in maintaining four QUICK programs which process QUICK data base tapes. The programs involved are BASEMOD and INDEXER of the Data Input subsystem, PLANSET of the Plan Generation subsystem, and program READSUM of the Data Output subsystem.

The programming techniques and FORTRAN coding used within each of these programs are directly related to the structure and content of the directory associated with the QUICK data base. Changes in the directory which involve the addition, deletion, or change in the ordering of the attributes listed therein, will always require a programming modification to BASEMOD, INDEXER, PLANSET, and READSUM. Program DECLARES provides a relatively simple method of implementing the required program modifications. To use this capability a set of DECLARES command cards are inserted in the program source code (FORTRAN) deck. Program DECLARES is then used to process the program deck, effect the required modifications, and prepare an output tape containing the modified FORTRAN program. The modified program is subsequently compiled using standard procedures and is ready for execution. A more specific explanation of DECLARES and its operation is presented in subsequent paragraphs.

Programs BASEMOD, INDEXER, PLANSET, and READSUM must include: the common blocks /PROCESS/, /EDITAPE/, and /EDITERM/; equivalences between the mnemonic attribute identifiers and words in the array VALUE (common block /PROCESS/); and the type declarations for these attributes. These equivalences are changed when: (1) the order of the attributes in the directory is modified; or (2) the directory is changed by adding or deleting attributes. When such changes are made, program DECLARES is

used to automatically insert the common statements, current equivalence statements, and associated type declarations into these FORTRAN programs.

The input to DECLARES is a pseudo FORTRAN source program which contains commands recognized by DECLARES. Based upon these commands, DECLARES inserts the appropriate common statements, equivalence statements, and type declarations in the source program. Additionally, DECLARES inserts "CALL CHANGE" statements and statements to assign appropriate values to the common variable NC required by subroutine CHANGE.

The action of DECLARES is strictly linear in the following sense: if an instruction read by DECLARES does not contain one of the DECLARES commands, its image is written on the output tape; if a card contains one of the commands CDECLAREX, CDECLARE, or CHANGE, the appropriate FORTRAN statements are written on the output tape. This requires that the DECLARES commands be placed in the user's source program in locations which are proper for the FORTRAN statements generated by DECLARES. The command END also controls the execution of DECLARES but does not affect the output FORTRAN program. A FORTRAN source program which is to be processed by DECLARES may contain any number of FORTRAN subroutines, and must contain the PROGRAM, SUBROUTINE, END, and SCOPE cards which are normally required by the FORTRAN compiler of the NMCSSC CDC 3800 computer.

Each card image which contains one of the four commands recognized by DECLARES is regarded as eight fields of 10 columns each; alphanumeric words are left-justified in these fields, and integers are right-justified. The use of these commands, including their placement in the FORTRAN source program, is described below.

1. CDECLAREX and CDECLARE Commands. The CDECLAREX command causes DECLARES to insert in the FORTRAN source program the COMMON statements for common blocks /PROCESS/, /EDITERM/, and /EDITAPE/. Additionally, the equivalence statements and type declarations for the attributes whose names appear in the fields following CDECLAREX are inserted. If no attributes are listed, the equivalences and declarations will be inserted for all attributes in the data base directory. Since this command is replaced in the FORTRAN program by nonexecutable statements, it must precede all executable statements in the program or subroutine in which it occurs. More than one occurrence of CDECLAREX in a program or subroutine will result in a compiler diagnostic, because DECLARES would cause duplicate common statements to appear in the program being processed. When CDECLAREX is used with a list of more than seven attributes, the CDECLARE command can be used to obtain the equivalences and declarations for the additional attributes. Any number of CDECLARE cards can be used; they must precede all executable statements in the program in which they appear. Both CDECLAREX and CDECLARE cards are terminated when a blank field is encountered. If required, a blank card must be inserted to terminate.

2. CHANGE Command. The CHANGE command generates a call on subroutine CHANGE with NC (in common block /PROCESS/) set to the index of the word corresponding to the attribute ATTR in the array VALUE. Whenever the value of a data base attribute is changed in a program, the change must be preceded by the appropriate CHANGE command.
3. END Command. The END command causes DECLARES to terminate. This card is at the end of the user's FORTRAN deck following the SCOPE card, and must be followed by a blank card.

As indicated in figure 28, processing begins by establishing the input/output logical unit assignments. These assignments are established by an input parameter card or set to their default values. The pertinent media are: the FORTRAN source program to be processed, IOUT; the data base tape, QUIKDB; the output FORTRAN source program which will subsequently be compiled, IOUTDEC; and if requested, the listing medium, ILIST.

Prior to processing the input program, subroutine READDIR is called and the data base directory is read in its entirety. The source program is then processed one record at a time. If the record from the source deck contains CDECLARE(X), the common blocks /EDITAPE/, /EDITERM/, and /PROCESS/ are inserted and written onto output tape IOUTDEC. The array GLOB, which is used to keep track of attributes, is cleared to zeros. If the record from the source deck contains CDECLARE (blank), the common blocks are not written to tape and the GLOB array is not cleared. In either case, the remainder of the CDECLARE card is searched for a list of attributes. If such a list is present, the attributes, one at a time, are matched against the directory, and the appropriate statement equivalencing that attribute to a position in the VALUE array is written onto the IOUTDEC tape. In addition, a type statement for the attribute (REAL or INTEGER) is written. If a specific list of attributes is not included, all attributes are matched to the directory, and the equivalence and type statements for all attributes are inserted into the source program.

If the record from the source deck contains CHANGE (attribute name), the following code is inserted into the source program:

```
NC=INDEX      (where INDEX is the position of the attribute in the
               directory)
CALL CHANGE    (a call to a subroutine to save the present value of
               the attribute).
```

The program is terminated by the occurrence of the word END at the beginning of a record or an end-of-file if the input source program is on tape.

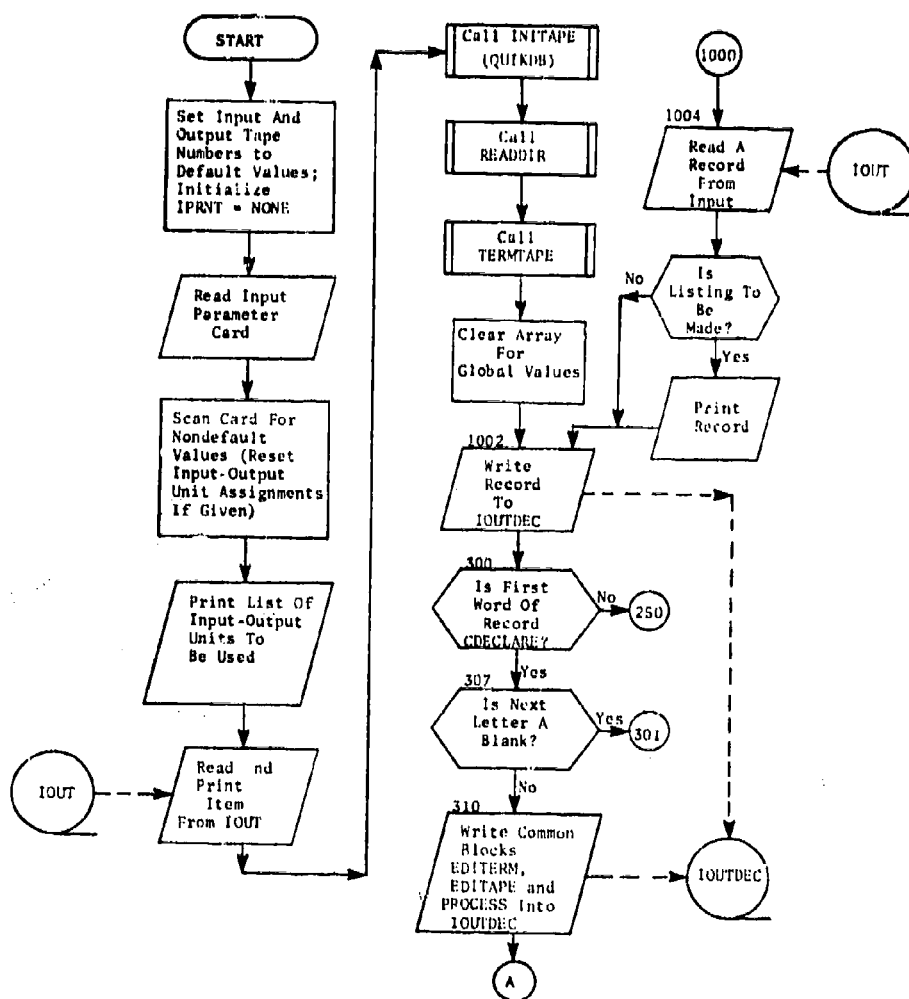


Fig. 28 . Program DECLARES
(Sheet 1 of 3)

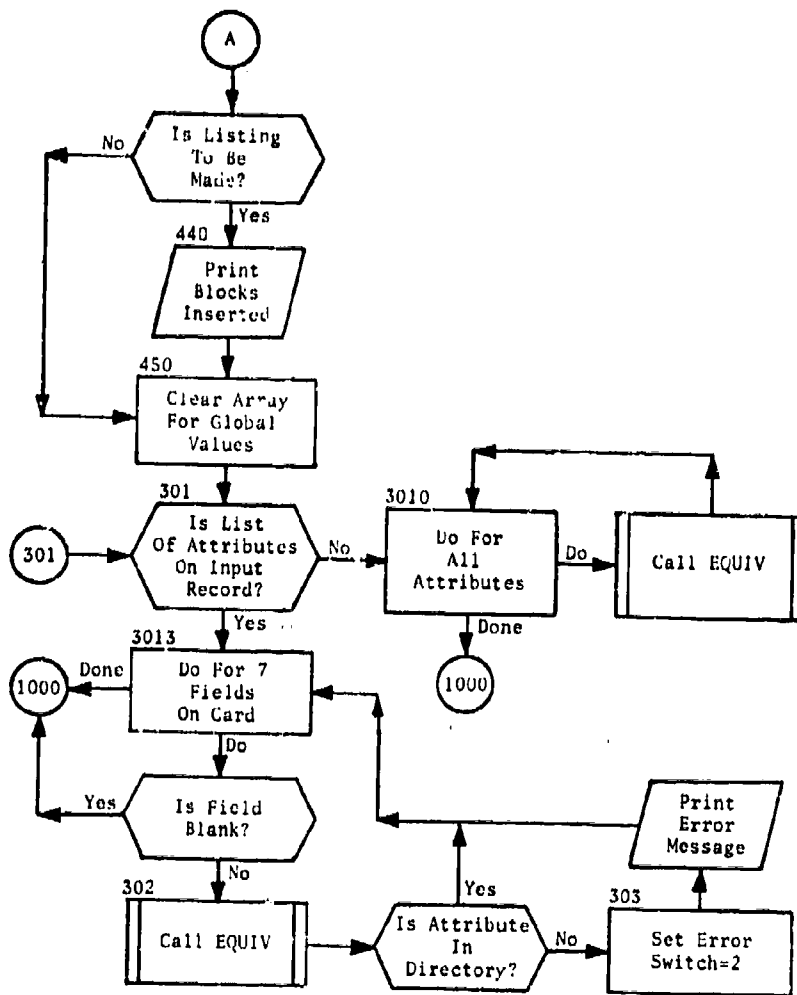


Fig. 28. (cont.)
(Sheet 2 of 3)

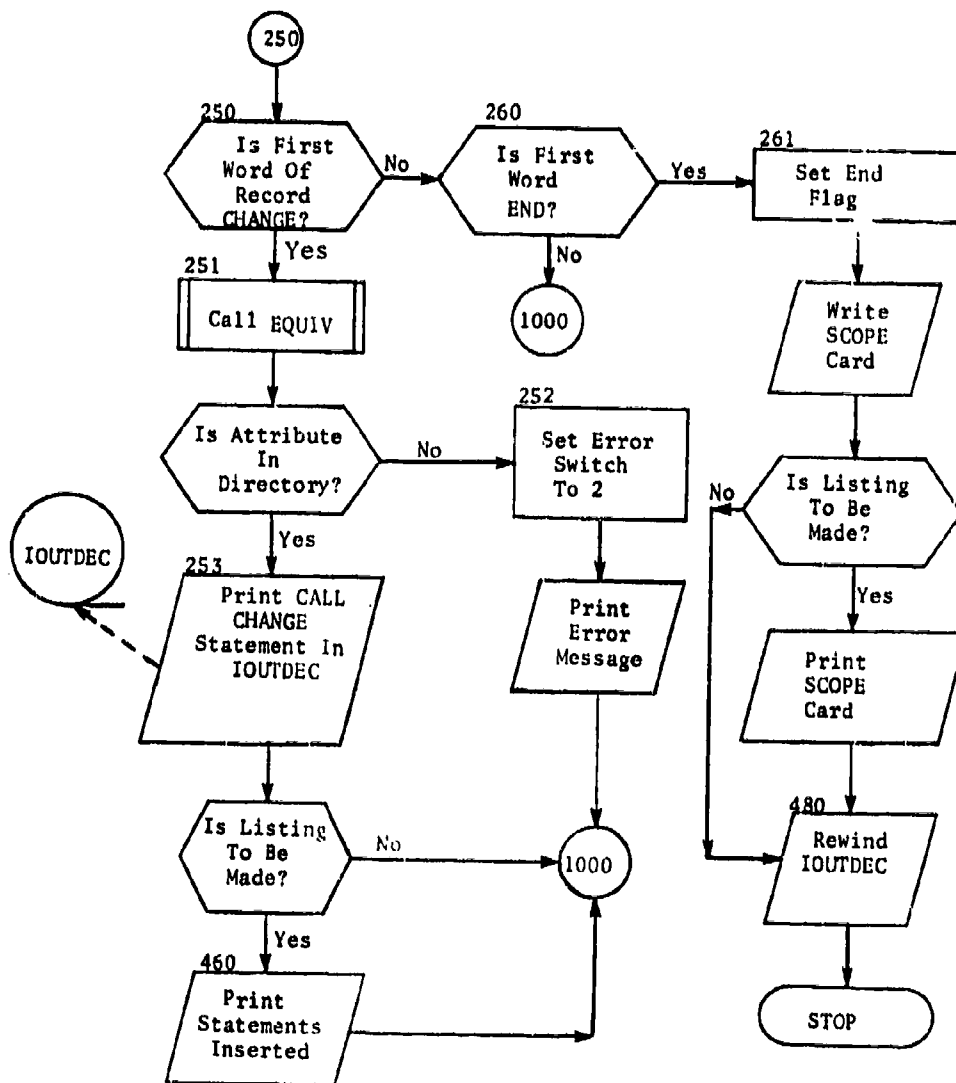


Fig. 28. (cont.)
(Sheet 3 of 3)

SUBROUTINE EQUIV

PURPOSE: To output to tape IOUTDEC the equivalence and type statements required for each attribute defined in the data base directory.

ENTRY POINTS: EQUIV

FORMAL PARAMETERS: ATTNM - The attribute name
J - Flag to indicate if the attribute is in directory: 0 if not found; index of attribute if found

COMMON BLOCKS: DIRECTORY, OUTFILES, XPRT

SUBROUTINES CALLED: ITLE

CALLED BY: DECLARES

Method

EQUIV uses function ITLE to locate the index of the requested attribute in the directory. If the attribute is not found, an error flag is returned to the calling program. If the attribute is located, the attribute name is equivalenced by its index to the VALUE array, and the statement is written to tape. The type of variable is determined by the value of ICODE in the directory, and the appropriate statement is written to tape. If printing has been requested, all items written to tape are also written to the ILIST medium.

Subroutine EQUIV is illustrated in figure 29.

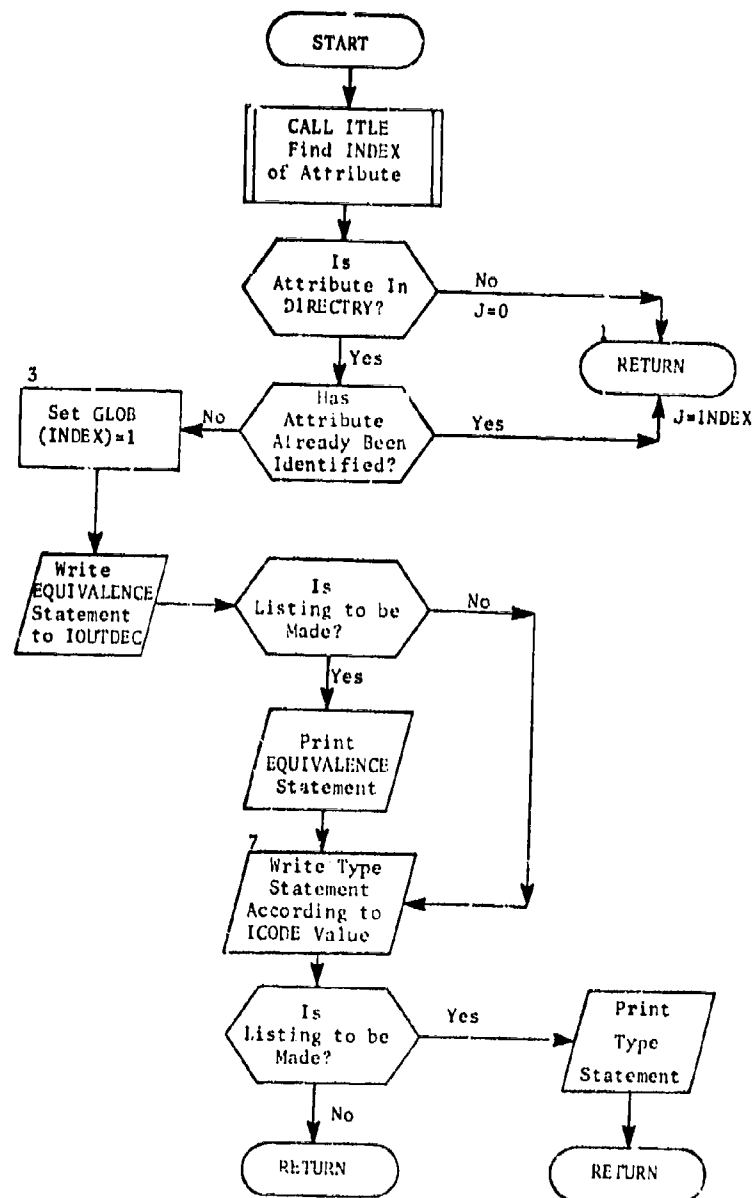


Fig. 29. Subroutine EQUIV

PROGRAM FILEDUMP

PURPOSE: To print specific portions of a magnetic tape, written in binary mode, or a disk file.

ENTRY POINTS: FILEDUMP

FORMAL PARAMETERS:

- MODE - Tape or disk file designator
- ITP - (If tape file), logical tape unit number
- NAME - (If disk file), logical file name
- IFF - First file to be printed
- IFREC - First record to be printed in each file
- IFW - First word to be printed in each record
- ILF - Last file to be printed
- ILREC - Last record to be printed in each file
- ILW - Last word to be printed in each record

COMMON BLOCKS: ITP, IREC, DATA, FILEIN

SUBROUTINES CALLED: NEXTFILE, NUMGET

Method

The investigation of portions of a data file can be a tedious process for two reasons. First, printing an entire file is time-consuming and costly. Second, manual conversion from binary data to other modes is time-consuming and error prone. Program FILEDUMP is used to print portions of a tape (written in binary mode) or a disk file in a readable form.

The portion of the tape or disk file to be printed is specified by the user. He specifies the files, records within files, and words within records to be printed. The user-input parameters to this program are the nine formal parameters listed above. These parameters are input on one data card in the order listed above.

The program prints each word of the requested portion of the file in four formats: octal (O16), fixed point (I16), floating point (E13.6), and alphanumeric (A8).

If the data contain illegal alphanumeric codes, blanks are inserted for these characters in the alphanumeric field. As a further aid, each four-field block of output corresponding to one data word is numbered with its order of occurrence within the record. If a number of

consecutive data words are identical, the program does not print each individual word. Rather, it prints the contents of the words once with a message listing the range of words which have this value.

The program also prints the actual length of each record. The maximum record size that can be accommodated is 20,000 words. If a record equals or exceeds this length, only the first 20,000 words will be printed, and the program will print the message RECORD LENGTH MAY EXCEED 20000.

If a read parity error is encountered on input, the program will attempt to reread the record six times. If the parity error persists, the program prints the message PARITY ERROR. SKIP TO NEXT RECORD and reads the next record on the tape or file.

Program FILEDUMP is illustrated in figure 30.

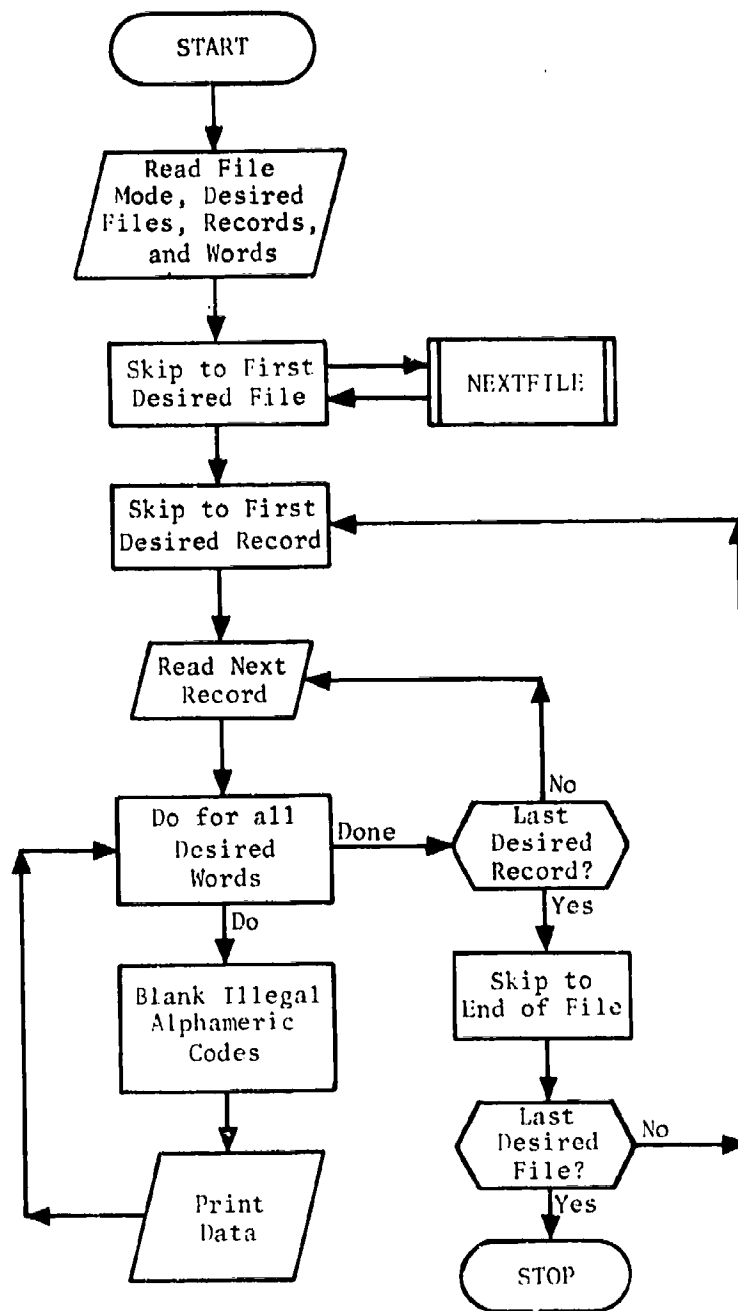


Fig. 30. Program FILEDUMP

CHAPTER 4 GENERAL UTILITIES

This chapter contains descriptions of a variety of subroutines and functions, performing various tasks, used throughout the QUICK system. Common blocks used by these programs are described in appendix A. A list of the programs and/or subroutines which call each utility routine is presented in appendix D.

SUBROUTINE ABORT

PURPOSE: Entry ABORT: To force a core dump on demand.
 Entry WARNING: To print a warning message and
 possible error diagnostics.

ENTRY POINTS: ABORT, WARNING

FORMAL PARAMETERS: None

COMMON BLOCKS: ERRCODE, ERRMESS

SUBROUTINES CALLED: Q8QERROR

Method

This subroutine prints error diagnostics and an optional core memory dump. The error diagnostics include:

- Error message
- Contents of A and Q registers
- Name of routine (ABORT or WARNING)
- A trace of the chain of subprogram calls back to the main program.

The error message is contained in common /ERRMESS/. This block consists of two arrays, each of 10-element length, IABORT and IWARN.

The message to be printed is contained in array IABORT for entry ABORT, and IWARN for entry WARNING. The calling program may place any message in these arrays subject to the following restrictions:

1. The message must end with a period (.) and contain no imbedded periods.
2. The message length must not exceed 80 characters, including the terminating period.

If entry ABORT is used, the error diagnostics will be followed by the phrase EXECUTION DELETED and the job will terminate with a core memory dump.

Common /ERRCODE/ consists of two error codes, KABORT and KWARN. They are preset to KABORT=0 and KWARN=1. A zero code forces job termination and

dump. A nonzero code forces only the diagnostic print and a normal return. Code KABORT is used by entry ABORT and KWARN by entry WARNING.

Subroutine ABORT is illustrated in figure 31.

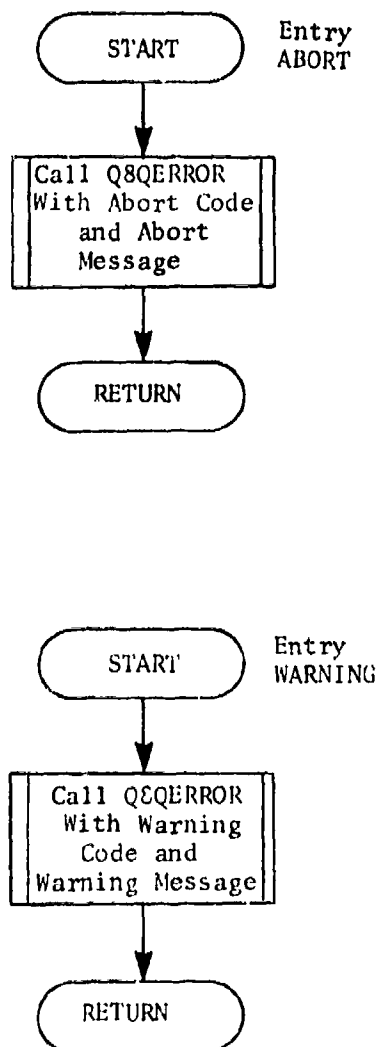


Fig. 31. Subroutine APORT

SUBROUTINE ANOTHER

PURPOSE: To unload a magnetic tape reel while maintaining the assignment of the physical unit.

ENTRY POINTS: ANOTHER

FORMAL PARAMETERS: NTOGO - Logical tape unit number

COMMON BLOCKS: None

SUBROUTINES CALLED: UNLOAD

Method

This assembly language subroutine uses the system macro UNLOAD to rewind and unload a magnetic tape reel. The formal parameter is set to the logical tape unit number of the tape to be unloaded. The physical unit assignment is not released. Therefore, another tape reel can be mounted on this unit.

Subroutine ANOTHER is illustrated below in figure 32.

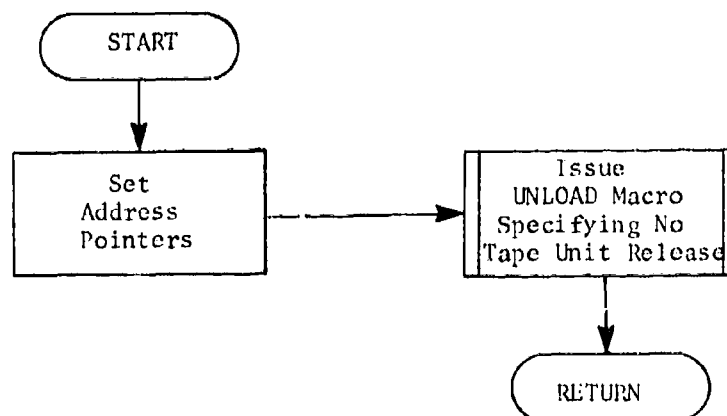


Fig. 32. Subroutine ANOTHER

FUNCTION ATN2PI

PURPOSE: To calculate the arc tangent function over the interval 0 to 2π .

ENTRY POINTS: ATN2PI

FORMAL PARAMETERS: Y, X (floating point numbers)

COMMON BLOCKS: None

SUBROUTINES CALLED: ATANF

Method

This function calculates the arc tangent of the value (Y/X) . The arc tangent returned to the calling program lies within the interval from 0 to 2π . The operating system function ATANF returns the principal value of the arc tangent; i.e., over the interval $-\pi/2$ to $+\pi/2$. ATN2PI uses ATANF to compute the principal value of the arc tangent of Y/X . The signs of X and Y are investigated to determine the quadrant of the arc tangent. The principal value is then modified to return a value within the correct quadrant.

Function ATN2PI is illustrated in figure 33.

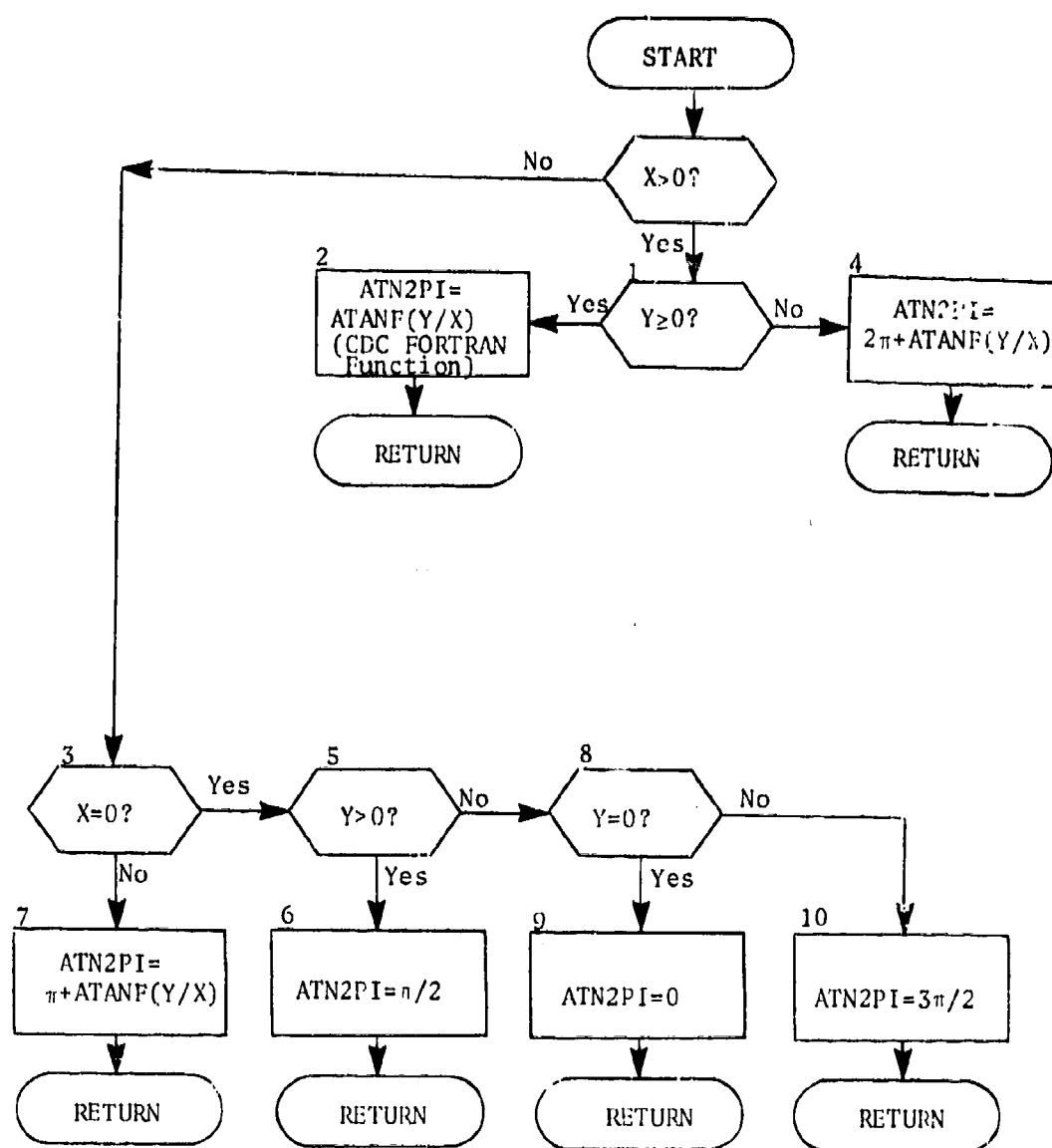


Fig. 33. Function ATN2PI

SUBROUTINE CHANGE

PURPOSE: To prepare for changing the value of any attribute in the data base item (record) currently being processed so that globally defined attributes, if changed, are restored to their original value in the following items.

ENTRY POINTS: CHANGE

FORMAL PARAMETERS: None

COMMON BLOCKS: PROCESS

SUBROUTINES CALLED: None

Method

If the index (NC) of the attribute to be changed is already contained in the list of input attribute-value pairs, the subroutine returns to the calling program. Otherwise, the index is stored in INITEM in the location equal to twice the number of pairs plus one, and the value is stored in the following location. The logical array DEF(NC) is set to one to show that the attribute with index NC is defined. The number of pairs NI is incremented by one and the number of entries by two. This assures that, if the attribute to be changed is globally defined, it will be restored to its original value when a new item is read in.

Subroutine CHANGE is illustrated in figure 34.

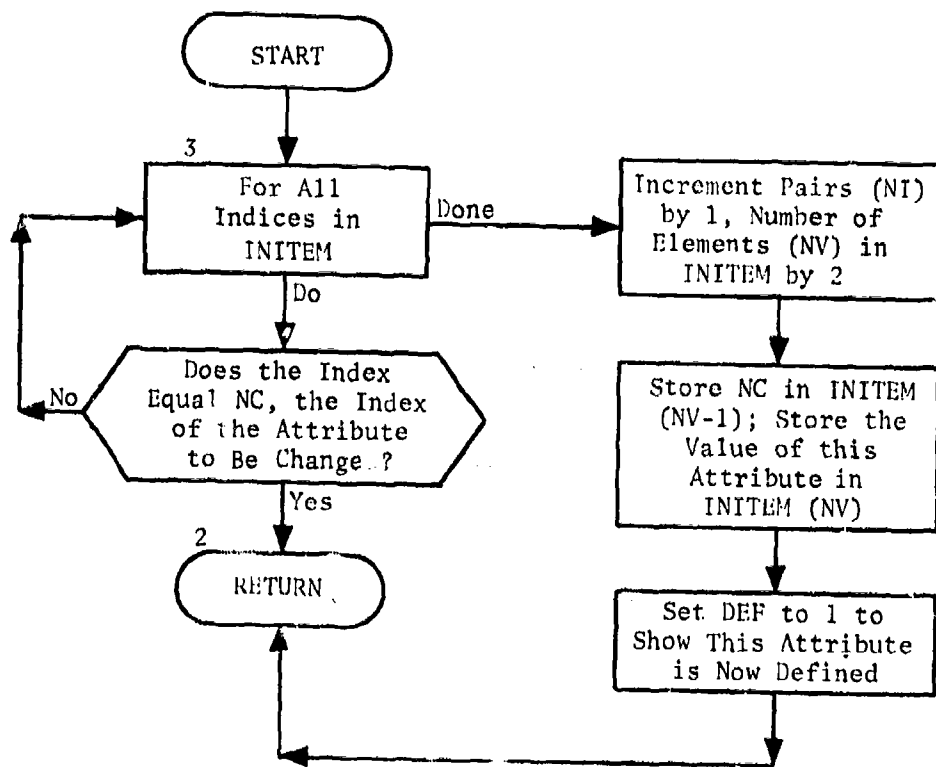


Fig. 34. Subroutine CHANGE

FUNCTION DELLONG

PURPOSE: To compute the signed difference between two longitudes.

ENTRY POINTS: DELLONG

FORMAL PARAMETERS: A - A floating point longitude
B - A floating point longitude

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

The input longitudes are expressed as decimal degrees measured in a westerly direction. West longitudes are in the range 0-180 degrees; east longitudes are in the range 180-360 degrees. Longitudes 0 and 360 are the Greenwich Meridian. This function returns the value of the difference between the two longitudes. The sign of the value is determined as follows:

Positive if $A \geq B$
Negative if $A < B$.

Function DELLONG is illustrated in figure 35.

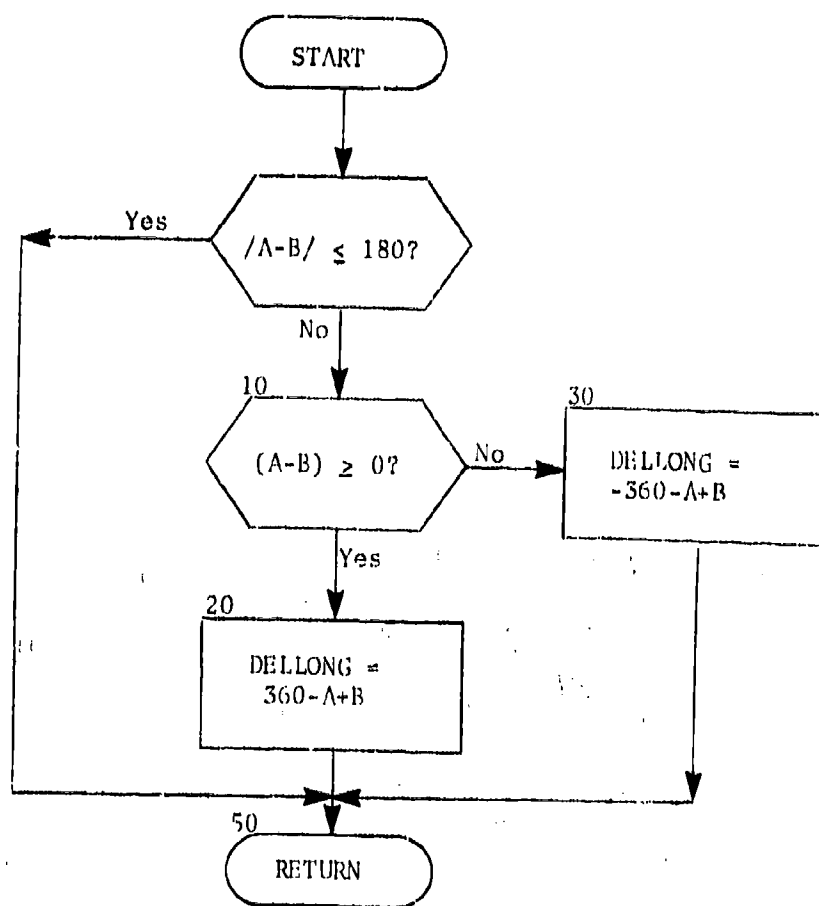


Fig. 35. Function DELLONG

FUNCTION DIFFLONG

PURPOSE: To compute the difference between two longitudes whose sign is determined by the shorter direction of travel from the first meridian to the second.

ENTRY POINTS: DIFFLONG, DIFFLNG*

FORMAL PARAMETERS: X1 - Floating point longitudes
X2 - Floating point longitudes

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

The input longitudes lie in the interval 0-360 degrees with west longitudes in the range 0-180 degrees and east longitudes in the range 180-360 degrees. Longitudes 0 and 360 define the Greenwich Meridian. This function returns a value whose absolute value is equal to the number of degrees of longitude traversed in using the shorter great circle route from meridian X1 to meridian X2. The sign is positive if the direction of travel is eastward and negative otherwise.

Function DIFFLONG is illustrated in figure 36.

*Duplicate entry for DIFFLONG.

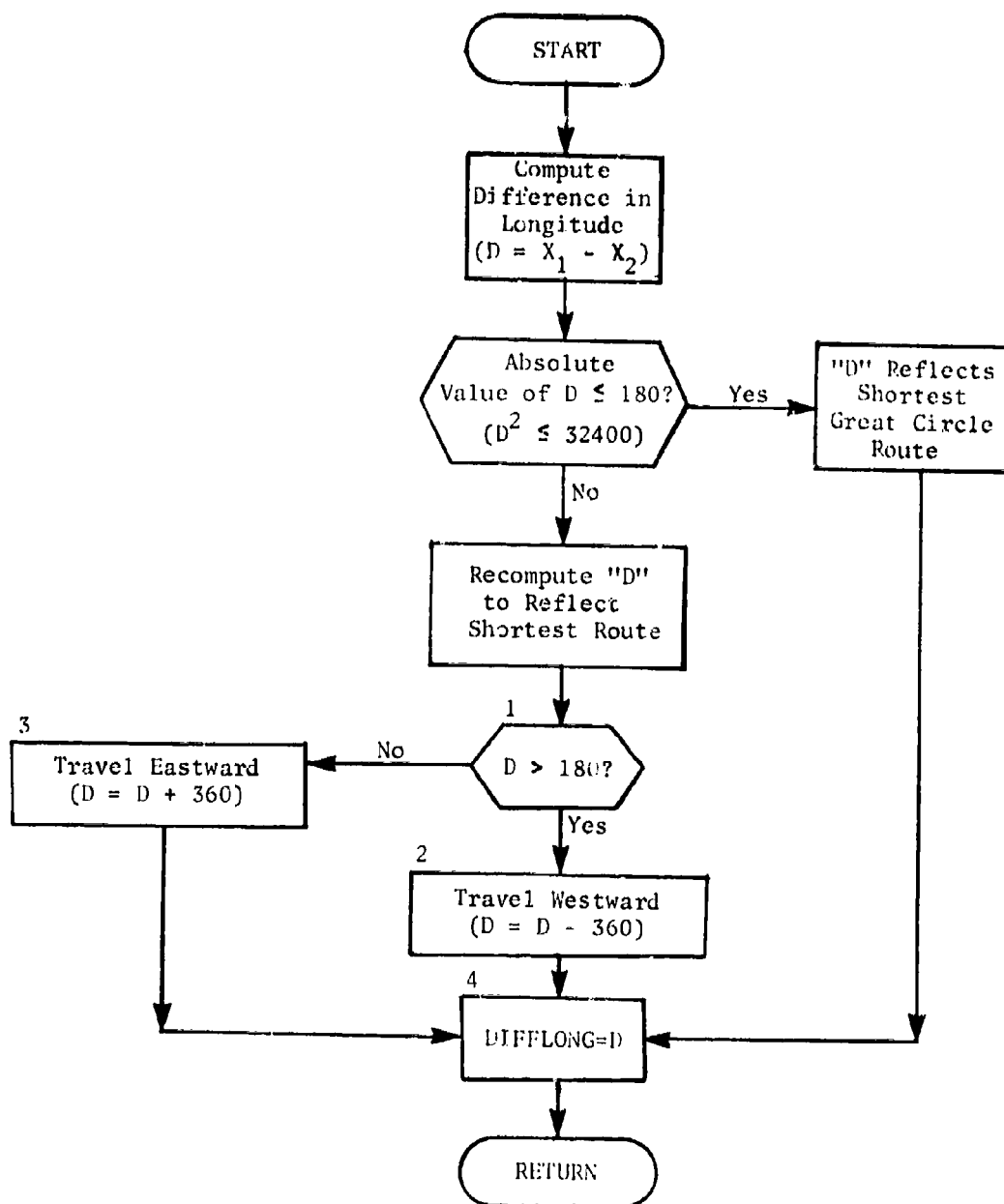


Fig. 36. Function DIFFLONG

FUNCTION DISTF

PURPOSE: To compute great circle distances in nautical miles.

ENTRY POINTS: DISTF

FORMAL PARAMETERS: LAT1 - Latitude of point 1
LONG1 - Longitude of point 1
LAT2 - Latitude of point 2
LONG2 - Longitude of point 2

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

The formal parameters are all type real. The coordinates are input in degrees with south latitude and east longitude coordinates being negative. (Lambda format for longitudes is also acceptable.)

If the difference in longitude is less than ~ 2.8 degrees, a straight-line approximation to the great circle route is used. Otherwise, standard law of cosines for a spherical triangle is applied to compute the great circle distance. The radius of the earth is assumed to be 3437.74677 nautical miles. The units of the result are nautical miles.

Function DISTF is illustrated in figure 37.

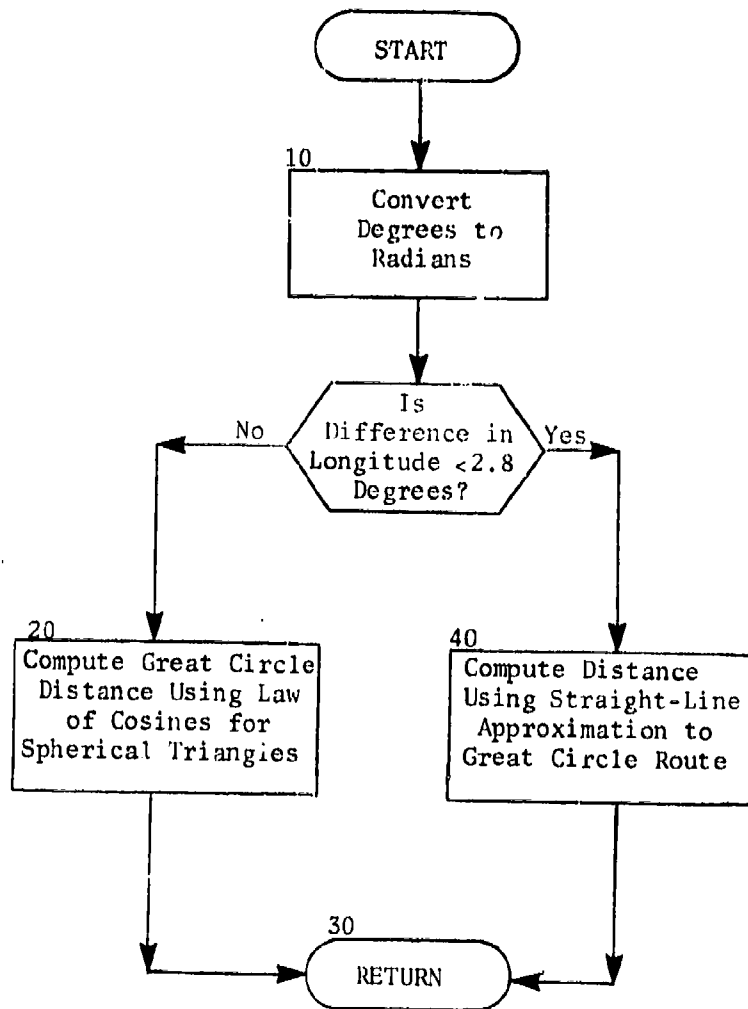


Fig. 37. Function DISTF

FUNCTION DSTF

PURPOSE: To compute the distance over a straight-line approximation to a great circle route between two points.

ENTRY POINTS: DSTF

FORMAL PARAMETERS: XLAT - First point latitude
YLAT - Second point latitude
DLONG - Longitudinal difference between the points

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

The formal parameters are all type real. They are in degrees in standard Lambda format.

This function computes a straight-line approximation to the great circle distance between the points defined by the formal parameters. The Pythagorean law is then used to calculate the distance along that approximation.

The radius of the earth is assumed to be 3437.74677 nautical miles. The units of the result are nautical miles.

Function DSTF is illustrated in figure 38.

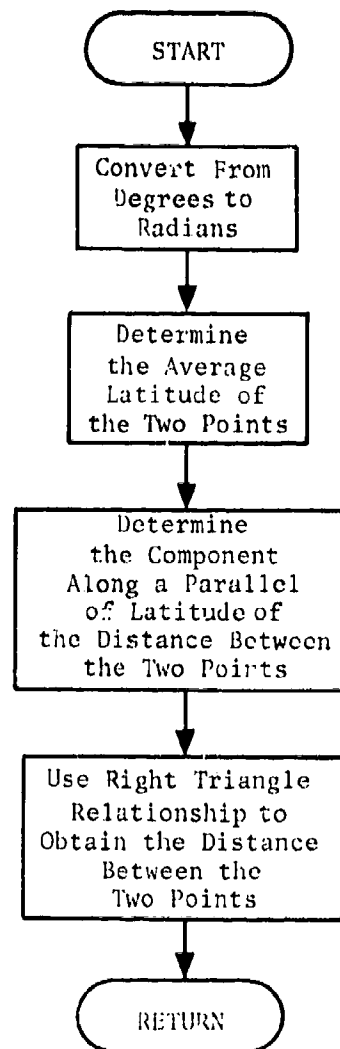


Fig. 38. Function DSTF

SUBROUTINE ENDDATA

PURPOSE: To terminate the data file.

ENTRY POINTS: ENDDATA

FORMAL PARAMETERS: NT1 - Tape to be terminated

COMMON BLOCKS: ITP, TWORD

SUBROUTINES CALLED: TERMTAPE*, WRWORD*

Method

This subroutine terminates an output data base tape by writing the word ENDDATA on it and calling TERMTAPE. ENDDATA is illustrated in figure 39 below.

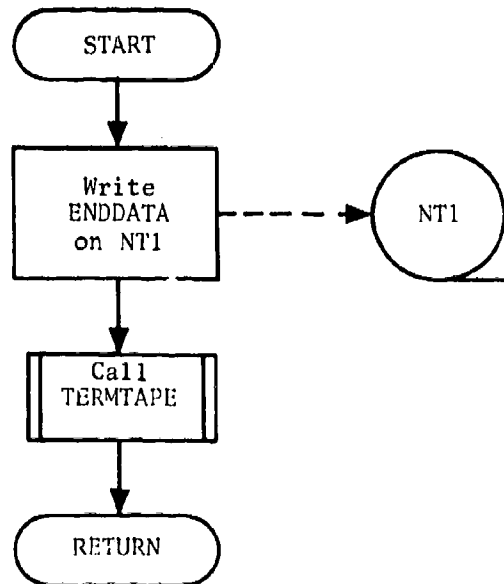


Fig. 39. Subroutine ENDDATA

*See Filehandler subroutines.

SUBROUTINE ERAZE

PURPOSE: To erase six inches on a magnetic tape to reserve space for future use or to skip a bad segment of tape.

ENTRY POINTS: ERAZE

FORMAL PARAMETERS: ITP - Logical tape unit number

COMMON BLOCKS: None

SUBROUTINES CALLED: ERASE

Method

This subroutine uses the system macro ERASE to erase six inches from a magnetic tape. The formal parameter is the logical unit number of the tape to be erased. The subroutine will erase the next six inches of the tape, and leave the tape in position to write beyond the erased portion.

Subroutine ERAZE is illustrated below in figure 40.

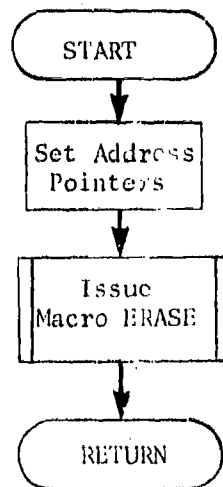


Fig. 40. Subroutine ERAZE

FUNCTION GETCLOCK

PURPOSE: To return the current time in floating point minutes.

ENTRY POINTS: GETCLOCK, GETCLK (duplicate entry)

FORMAL PARAMETERS: X - A dummy parameter

COMMON BLOCKS: None

SUBROUTINES CALLED: TIMEF

Method

This subroutine calls the system function TIMEF to return the current time in milliseconds. A division by 60,000 is used to return the current time in floating point minutes.

Function GETCLOCK is illustrated below in figure 41.

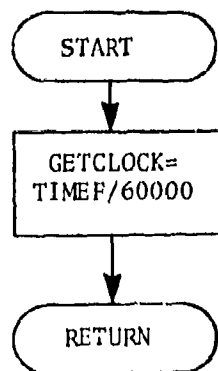


Fig. 41. Function GETCLOCK

FUNCTION GETDATE

PURPOSE: To obtain the current date.

ENTRY POINTS: GETDATE

FORMAL PARAMETERS: X - A dummy parameter

COMMON BLOCKS: None

SUBROUTINES CALLED: DATE

Method

This function calls the system macro DATE to obtain the current date in the format MM/DD/YY. When called in a FORTRAN program, this function will return the date in a floating point format.

Function GETDATE is illustrated below in figure 42.

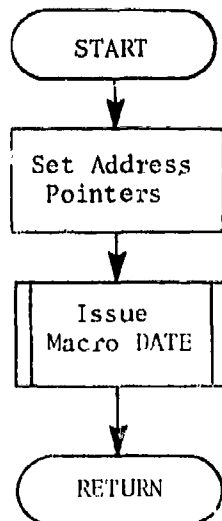


Fig. 42. Function GETDATE

SUBROUTINE GETLIMIT

PURPOSE: To return the limits of available memory in both banks of the CDC 3800

ENTRY POINTS: GETLIMIT

FORMAL PARAMETERS: None

COMMON BLOCKS: BNKBND

SUBROUTINES CALLED: MEMORY

Method

This subroutine is called by subroutine STORAGE to return the limits of available memory in both banks of the CDC 3800. The limits are returned in common /BNKBND/ with the lower limits preceding the upper limits. That is, the four words of storage in this common block are allocated in order: lower limit in bank 0, lower limit in bank 1, upper limit in bank 0, and upper limit in bank 1. The system macro MEMORY is used to determine these limits.

Subroutine GETLIMIT is illustrated in figure 43.

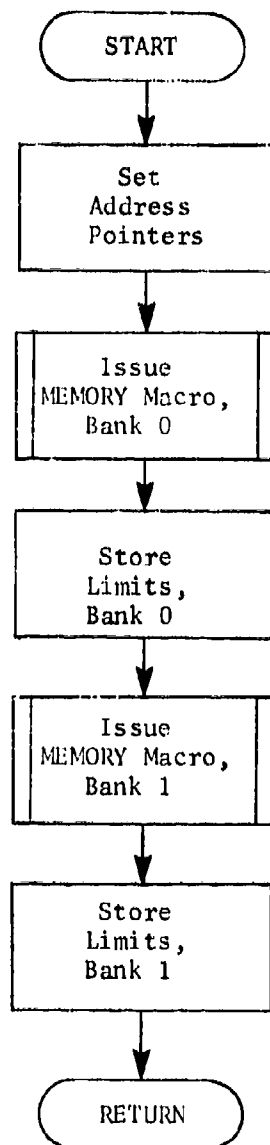


Fig. 43. Subroutine GETLIMIT

SUBROUTINE GETVALU

PURPOSE: To convert unformatted input data into attribute-value form.

ENTRY POINTS: GETVALU

FORMAL PARAMETERS:

- INPUT - An array containing the input data
- NDATA - Number of data items found in input array
- NAMES - Names of variables found in input array
- INVALU - Values assigned to variable in input array
- INDEX1 - Value of first index of array
- INDEX2 - Value of second index of array
- INDEX3 - Value of third index of array
- MORE - Input termination indicator

COMMON BLOCKS: None

SUBROUTINES CALLED: IWANT, NUMGET

Method

This subroutine receives as input an array INPUT which contains 80 BCD characters of information. This input is unformatted but consists of a series of variable names, array indices, and values. Subroutine GETVALU prepares lists of the variables contained in the input, their array indices, if necessary, and their values.

Consider the input array as an 80-column card. GETVALU considers the card to be broken into fields as follows:

1. Parameter Field. Delimited by commas (,) with first field beginning in column 1 and last field terminated by column 80. Each parameter field may contain a number of subfields containing the name of the parameter, the array indices for the parameter, and the value of the parameter.
2. Name Subfield. This subfield gives the name of the variable represented in the parameter field. If the name subfield is missing, the subroutine assumes that the current parameter field is for the succeeding element of the array processed in the preceding parameter field. The name subfield begins at the beginning of the parameter field. The first nonblank character

cannot be numeric. Imbedded blanks are ignored. This subfield is terminated by either an equals sign (=) for simple variables, or by an open parenthesis (()) for array variables.

3. Index Subfield. This subfield gives the indices for an array variable. If it is not present, the variable is assumed to be a simple variable. This subfield must follow a name subfield. It begins with an open parenthesis (()) and ends with a close parenthesis ()). Within the parentheses, the array indices are separated by commas. The maximum number of indices is three.
4. Value Subfield. This subfield may be of two types, alphameric or numeric. An alphameric field is begun and terminated with an asterisk (*). There may be no imbedded asterisks in the field. The maximum number of characters (including blanks) in an alphameric field is 16. A numeric field consists of the characters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, -, ., E, or D. Imbedded blanks are ignored. The maximum number of nonblank characters in a numeric field is 16.
5. Termination Subfield. This subfield consists of a dollar sign (\$). Its appearance in a name subfield as the first nonblank character will terminate processing of the input data. In addition, the value of the termination indicator MORE is set to zero.

Subroutine GETVALU processes these fields to produce the following output:

1. NDATA. Number of valid parameter fields processed from the input data.
2. NAMES. An array containing the names of the variables as determined from the name subfields. Maximum name length is eight characters.
3. INVALU. An array containing the characters in the value subfields. There are two elements in the INVALU array for each element in the NAMES array. For the variable name in the Jth position in the NAMES array, the corresponding value is contained in the (2*J)-1 and (2*J) positions of the INVALU array.
4. INDEX1. An array containing the value of the first index of an array. For simple variables, its value is one.
5. INDEX2 and INDEX3. Arrays used similarly to INDEX1.
6. MORE. A termination indicator, set to 0 if a dollar sign (\$) appears as the first nonblank character of a name subfield; set to 1 otherwise.

The values returned by subroutine GETVALU are in BCD code in the following formats:

1. Alphanumeric. Left-justified; blank filled on the right if field length is less than 16.
2. Numeric. Right-justified with no imbedded blanks; blank filled on the left if field length is less than 16.

In this format, the value field (IVALU) may be decoded using standard FORTRAN formats (I, F, A, E, D).

When an array is input, a number of successive elements can be input without repetition of the name and index subfields. If GETVALU encounters a value subfield prior to a name subfield within any parameter field, it assumes that the current value subfield refers to the array element immediately following the element used in the preceding parameter field. However, the first parameter field in each input array (or on each card) must have a name subfield. Omission of this subfield in the first parameter field will be flagged as an error.

Subroutine GETVALU is illustrated in figure 44.

Note: Error checking
is included in the
processing of each field.

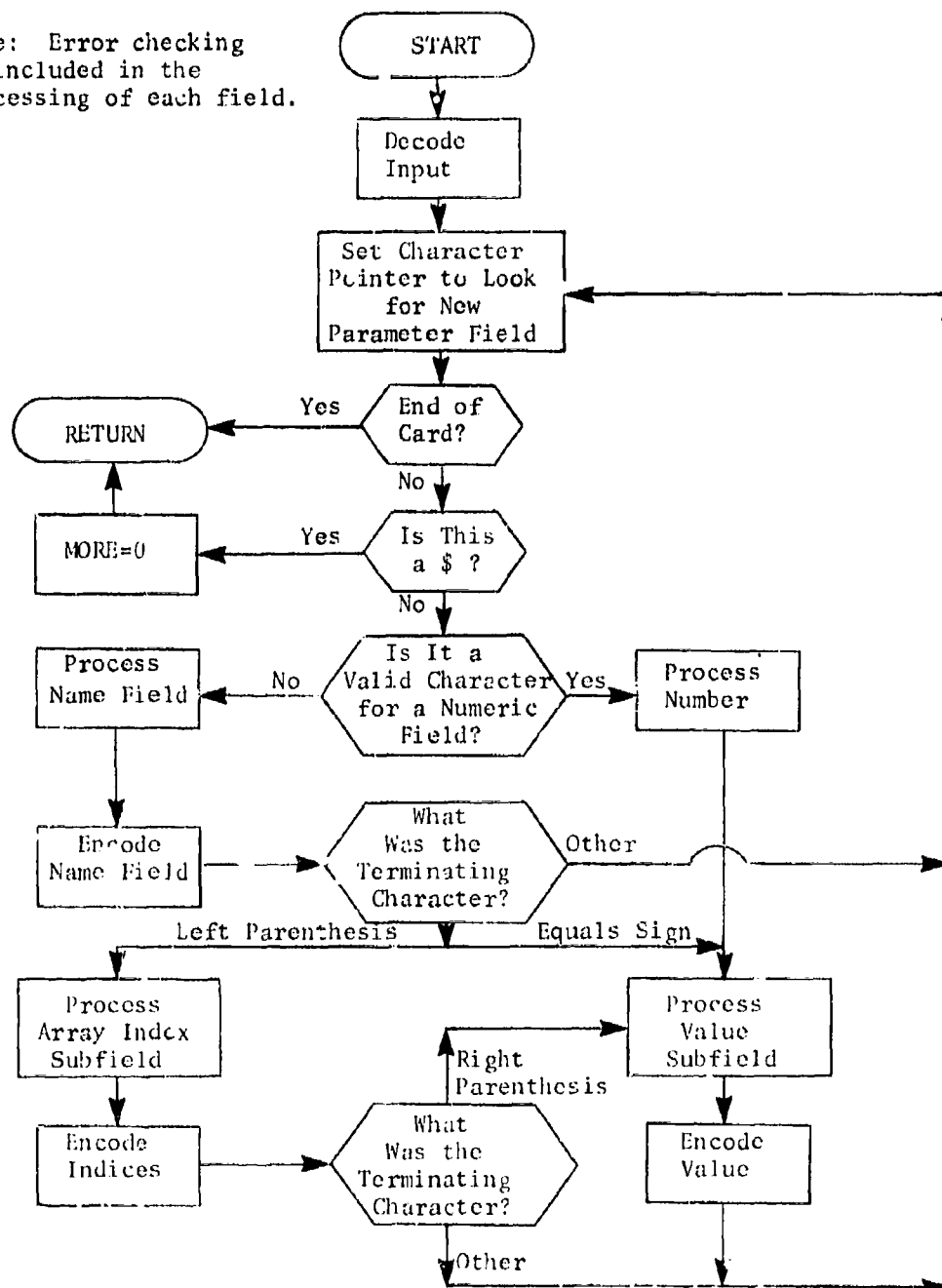


Fig. 44. Subroutine GETVALU

FUNCTION IGET

PURPOSE: To unpack a data item from a storage area according to a specified format.

ENTRY POINTS: IGET

FORMAL PARAMETERS:

- KEY - A key word generated by the function KEYMAKE
- INDEX - An index to the array IARR; the value of IGET will be extracted from IARR(INDEX)
- IARR - The array from which the data will be extracted

COMMON BLOCKS: DATPK

SUBROUTINES CALLED: ABORT

Method

The variable KEY is compared with ISVKEY (the value of KEY on the previous call to IGET). If they are not equal, then the KEY is unpacked and the variables ITYPE, NBITS, and NSHIFT are set (see description of function KEYMAKE). If KEY and ISVKEY are equal, then it is assumed that ITYPE, NBITS, and NSHIFT have been set by a previous call on IGET.

If the data are unpacked (i.e., ITYPE=4), then IGET is set equal to IARR(INDEX) and the routine returns. If ITYPE≠4, then IARR(INDEX) is shifted right NSHIFT bits, and now the rightmost NBITS are placed in IOUT.

If ITYPE=2, then IGET is set equal to IOUT, and the routine returns. If ITYPE=1, then the proper sign is attached to the value IOUT before setting the value of IGET to IOUT and returning control to the calling program.

If ITYPE does not equal 1, 2, or 4, then the subroutine ABORT is called.

Function IGET is illustrated in figure 45.

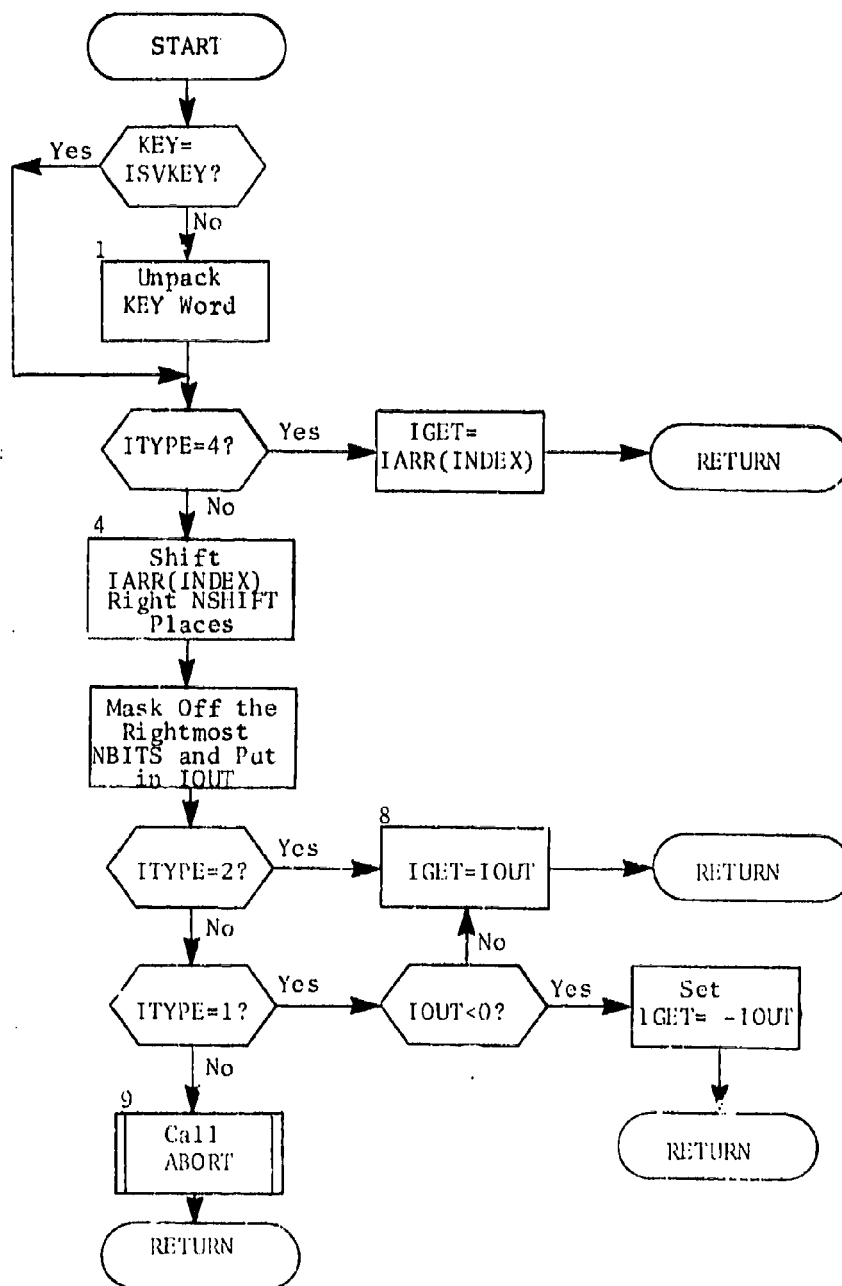


Fig. 45. Function IGET

SUBROUTINE INITEDIT

PURPOSE: To initialize for editing or processing data base files.

ENTRY POINTS: INITEDIT, INITEDT (duplicate entry)

FORMAL PARAMETERS: IT - Input data base file

COMMON BLOCKS: DIRECTRY, EDITAPE, PROCESS

SUBROUTINES CALLED: READDIR, WRITEDIR

Method

This subroutine is the basic initialization routine for any processing of the data base on unit IT, either for ordinary retrieval processing, or for editing to create one or more output data bases.

In the event that there are one or more output data bases, NOUT in common block /EDITAPE/ must be filled, prior to calling this routine, with the number of output bases to be created, and array ITOUT of the same block must contain the unit numbers assigned to the output data base files. If NOUT is not set prior to calling INITEDIT, it automatically assumes the value 0, and no output files will be created.

In addition, if the unit specified by IT is negative, it is a signal to FILEHNR that the input file is a disk file.

The basic actions of this subroutine are therefore to read the directory file from the input data base and to output the directory on the specified output data base files (if any). In addition, the values of all attributes in memory are set to their default values as specified in the directory, and the definition (DEF) and global definition (LGLOB) flags are all set to 0 (FALSE).

Subroutine INITEDIT is illustrated in figure 46.

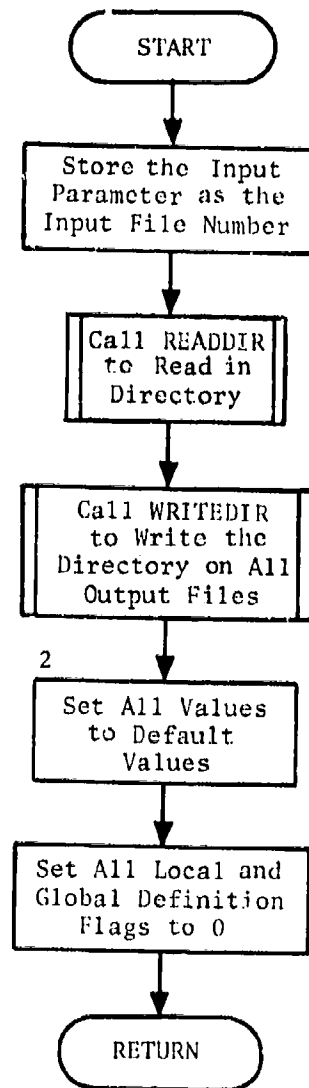


Fig. 46. Subroutine INITEDIT

SUBROUTINE INPITEM

PURPOSE: To read items from a data base file.

ENTRY POINTS: INPITEM, NEXTITEM, NEXTITM*

FORMAL PARAMETERS: None

COMMON BLOCKS: EDITAPE, EDITERM, ITP, PROCESS, TWORD

SUBROUTINES CALLED: ENDDATA, OUTWORDS, RDARRAY**, RDWORD**, TERMTAPE**

Method

INPITEM is used to input the first item (record) of a data base prior to executing the processing logic. Subsequent items beyond the first are read in by calls on a second entry point, NEXTITEM or NEXTITM.

The action of both these routines is as follows. When called, the data base contained on unit INTP (common block /EDITAPE/) is read until the next item has been stored in memory with the values of all attributes associated with it. Any global definitions or undefinitions which occur prior to this item are automatically placed in force by these routines.

After execution of INPITEM or NEXTITEM, the resulting state of memory in common block /PROCESS/ is as follows. The array VALUE is filled with the correct value of every attribute which is currently defined and with the default value of all currently undefined attributes. The logical array DEF has the value TRUE for all currently defined items and the value FALSE for all currently undefined items. Finally, logical array LGLOB has the value TRUE for all attributes which are currently globally defined and the value FALSE for the remainder.

In addition, if there are any output files, all global definitions and undefinitions are automatically transmitted to the set of output data base files at the same time that they are processed here.

Subroutine INPITEM is illustrated in figure 47.

*Duplicate entry for NEXTITEM.

**See filehandler subroutines.

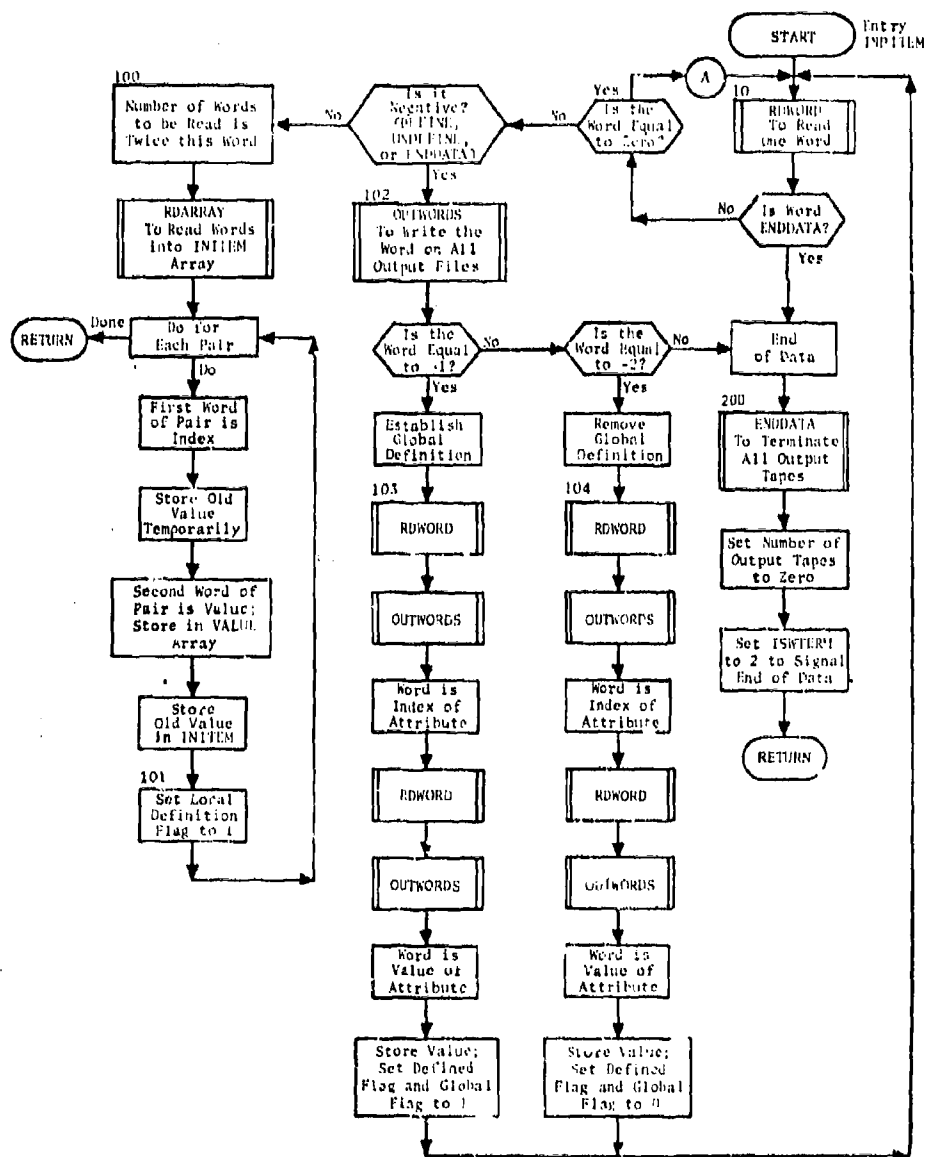


Fig. 47. Subroutine INPITEM
(Sheet 1 of 2)

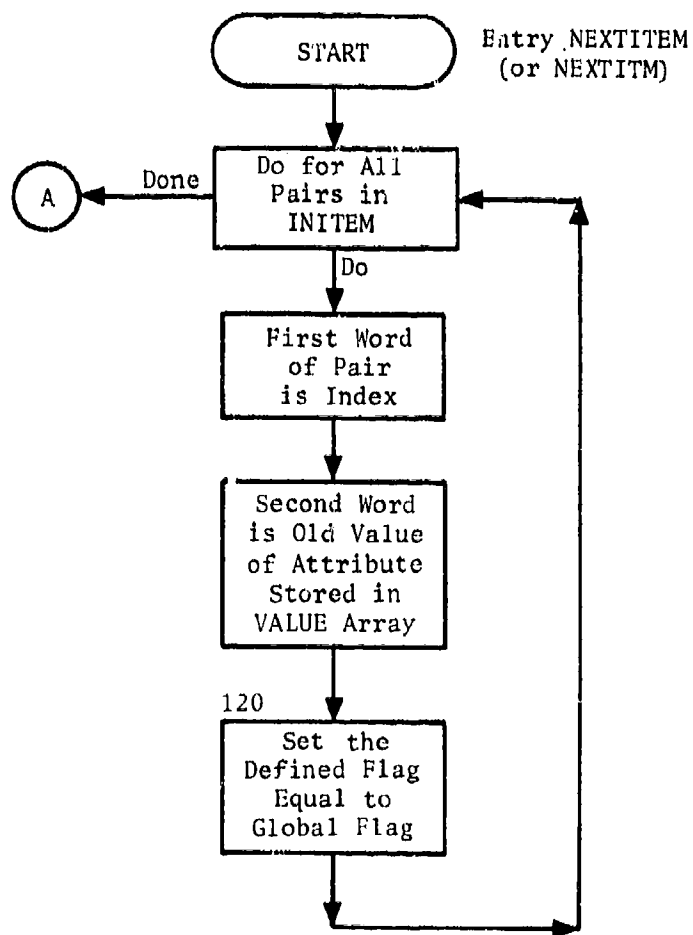


Fig. 47. (cont.)
(Sheet 2 of 2)

SUBROUTINE INTERP

PURPOSE: To find by interpolation the point (SR, TR) located some given fraction of the distance from the point (S1, T1) to the point (S2, T2).

ENTRY POINTS: INTERP

FORMAL PARAMETERS: None

COMMON BLOCKS: POLITE

SUBROUTINES CALLED: INTERPGC

Method

The parameters which control the interpolation are contained in common /POLITE/ as follows.

1. Input parameters:

S1 - latitude of first point
T1 - longitude of first point
S2 - latitude of second point
T2 - longitude of second point
FACTOR - fraction of distance to be interpolated

2. Output parameters:

SR - latitude of interpolated point
TR - longitude of interpolated point

All latitudes and longitudes are carried internally in the QUICK system in the following format:

North latitude	0. (equator) to +90. (North Pole)
South latitude	0. (equator) to -90. (South Pole)
East longitude	180. to 360. (Greenwich Meridian)
West longitude	0. (Greenwich Meridian) to 180

The variable FACTOR determines the fraction of the distance from the first point to the second that is equal to the distance from the first point to the interpolated point.

Subroutine INTERP first determines whether the fraction FACTOR is within the interval $0 < \text{FACTOR} < 1$. If not, the result (SR, TR) is set to (S1, T1) when $\text{FACTOR} \leq 0$, or to (S2, T2) when $\text{FACTOR} \geq 1$, and the subroutine returns.

When FACTOR is within range, however, the interpolation is to be performed. First, the difference in longitude of the two input points, $T12 = T2 - T1$, is computed. If that difference is greater than 2.8 degrees, INTERP calls the utility subroutine INTERPGC to perform the interpolation along the great circle route from (S1, T1) to (S2, T2), and then returns.

If $|T12|$ is less than 2.8 degrees, INTERP performs a straight-line, or Mercator, interpolation between (S1, T1) and (S2, T2) by putting:

$$\begin{aligned} \text{SR} &= \text{S1} + \text{FACTOR}(\text{S2} - \text{S1}) \text{ and} \\ \text{TR} &= \text{T1} + \text{FACTOR}(\text{T2} - \text{T1}). \end{aligned}$$

When the resultant TR is less than zero, 360 degrees are added to it; similarly, 360 degrees are subtracted from a TR which is greater than 360 degrees.

Subroutine INTERP is illustrated in figure 48.

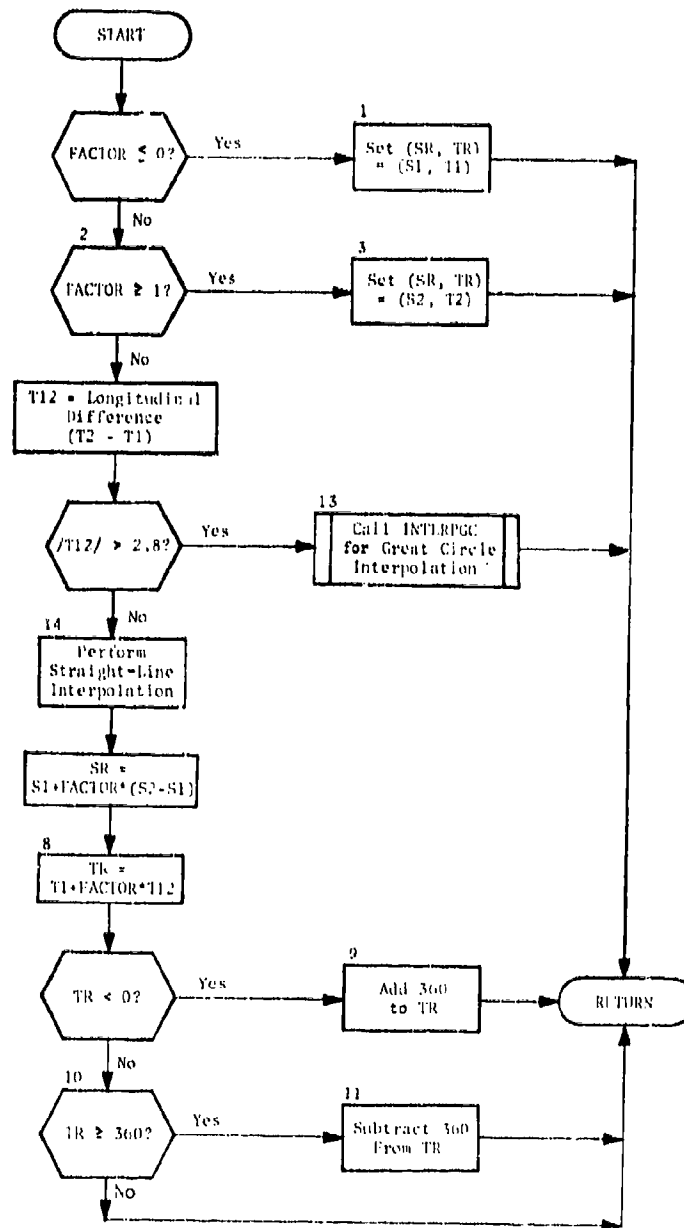


Fig. 48. Subroutine INTERP

SUBROUTINE INTERPGC

PURPOSE: To perform interpolation over great circle routes. It determines a point (SR, TR) located a given fraction FACTOR of the distance from the point (S1, T1) to (S2, T2) where the coordinates are latitude and longitude.

ENTRY POINTS: INTERPGC, INTRPGC (duplicate entry)

FORMAL PARAMETERS: None

COMMON BLOCKS: POLITE, A, B, C

SUBROUTINES CALLED: ATN2PI, DISTF

Method

The parameters which control the interpolation are contained in common /POLITE/ as follows:

1. Input parameters:

S1 - latitude of first point
T1 - longitude of first point
S2 - latitude of second point
T2 - longitude of second point
FACTOR - fraction of distance to be interpolated

2. Output parameters:

SR - latitude of interpolated point
TR - longitude of interpolated point

All latitudes and longitudes are floating point variables giving the data in degrees in standard Lambda format. The variable, FACTOR, determines the fraction of the distance from the first point to the second that is equal to the distance from the first point to the interpolated point.

For the purpose of this description, we assume now a right-handed coordinate system shown in figure 49, where the angle α is the longitude, measured east from the zero meridian (located along the X-axis). Here the latitude is given by the angle δ . Then the unit vector r_i associated with the latitude δ_i and longitude α_i is given by:

$$r_i = \begin{pmatrix} \cos \alpha_i \cos \delta_i \\ \sin \alpha_i \cos \delta_i \\ \sin \delta_i \end{pmatrix}$$

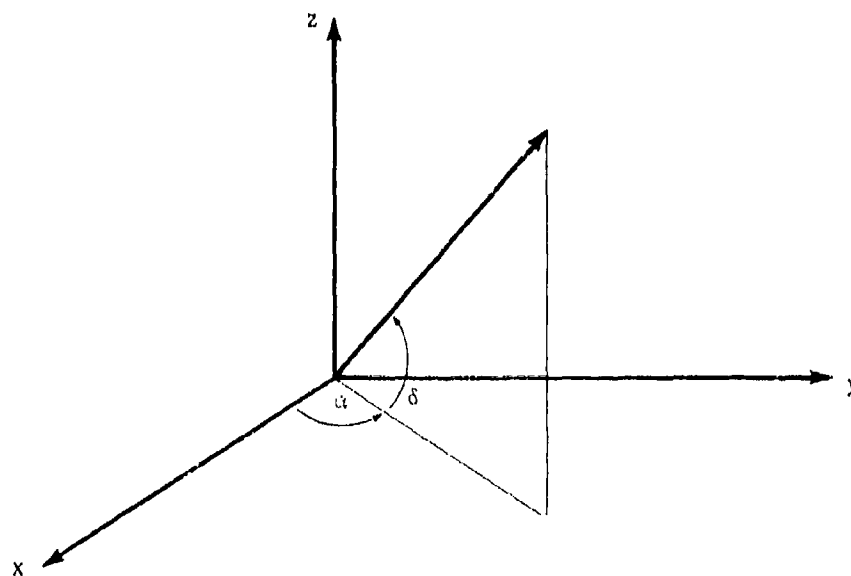


Fig. 49. Coordinate System for INTERPGC

Assume we are given the unit vector r_1 located at (α_1, δ_1) , and the unit vector r_2 determined by (α_2, δ_2) , and the angle θ between them. We wish to find a unit vector r in the plane determined by r_1 and r_2 some given angle θ_1 from r_1 . To do this we make use of the following three vector relations:

$$(r_1 \times r_2) \cdot r = 0$$

$$r_1 \cdot r = \cos \theta_1$$

$$r_2 \cdot r = \cos \theta_2$$

where $\theta_2 = \theta - \theta_1$.

These relations may be expanded as follows:

$$(r_{23}r_{12} - r_{13}r_{22})x + (r_{21}r_{13} - r_{11}r_{23})y + (r_{11}r_{22} - r_{12}r_{21})z = 0$$

$$r_{11}x + r_{12}y + r_{13}z = \cos \theta_1$$

$$r_{21}x + r_{22}y + r_{23}z = \cos \theta_2$$

where $r = (x, y, z)$. This equation has a unique solution in x, y , and z provided $\theta \neq 0$ or $\theta \neq \pi$, since the determinant:

$$(r_1 \times r_2) = \sin^2 \theta$$

is not equal to 0. Now since the resultant vector r is:

$$r = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \cos \alpha \cos \delta \\ \sin \alpha \cos \delta \\ \sin \delta \end{pmatrix}$$

then the desired point (α, δ) is given by:

$$\delta = \sin^{-1} z$$

$$\alpha = \tan^{-1} \left(\frac{y}{x} \right)$$

where the value of the arc tangent is not necessarily its principal value. (Function ATN2PI is used to calculate this value.)

To obtain θ and θ_1 when FACTOR is known, observe that:

if: D is the great circle distance from the point (α_1, δ_1) to the point (α_2, δ_2) and

R is the radius of the earth

then: $\theta = D/R$

$\theta_1 = \text{FACTOR} \cdot \theta$

Subroutine INTERPGC is illustrated in figure 50.

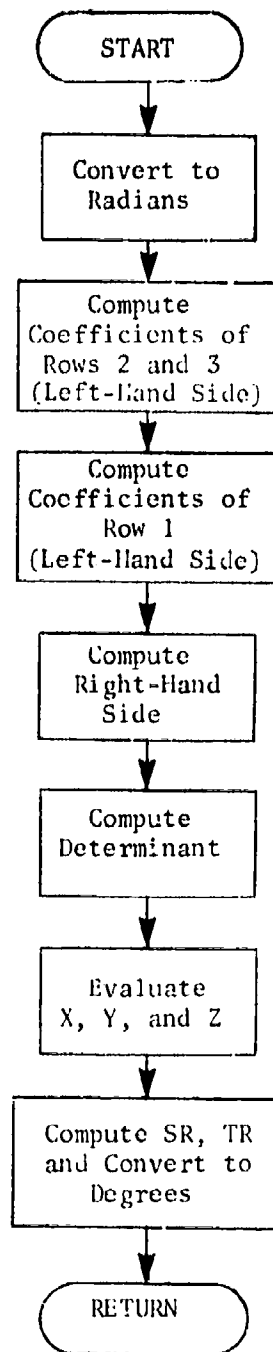


Fig. 50. Subroutine INTERPGC

SUBROUTINE IPUT

PURPOSE: To pack a given data item into a storage area according to a specified format.

ENTRY POINTS: IPUT

FORMAL PARAMETERS:

- KEY - A key word generated by the function KEYMAKE
- INDEX - An index to the array IARR; IVAL will be packed into the word IARR(INDEX)
- IVAL - The data word which is to be packed
- IARR - The array into which these data are to be packed

COMMON BLOCKS: DATPK

SUBROUTINES CALLED: None

Method

The variable KEY is compared with ISVKEY (the value of KEY on the previous call to IPUT). If they are not equal, then the KEY is unpacked, and the variables ITYPE, NBITS, and NSHIFT are set (see description of function KEYMAKE). If KEY and ISVKEY are equal, then it is assumed that ITYPE, NBITS, and NSHIFT have been set by a previous call on IPUT.

If the data are not to be packed (i.e., ITYPE=4), then the variables IN and MSK are set equal to IVAL and 0, respectively. If ITYPE≠4, then IN is set equal to the rightmost NBITS of IVAL and shifted left NSHIFT bits. MSK is set equal to MAST(NBITS), shifted left NSHIFT bits, and complemented.

The bits of IARR(INDEX) where the data are to be stored are set equal to zero. The variable IN (which is IVAL shifted to its proper position) is now added to IARR(INDEX).

Subroutine IPUT is illustrated in figure 51.

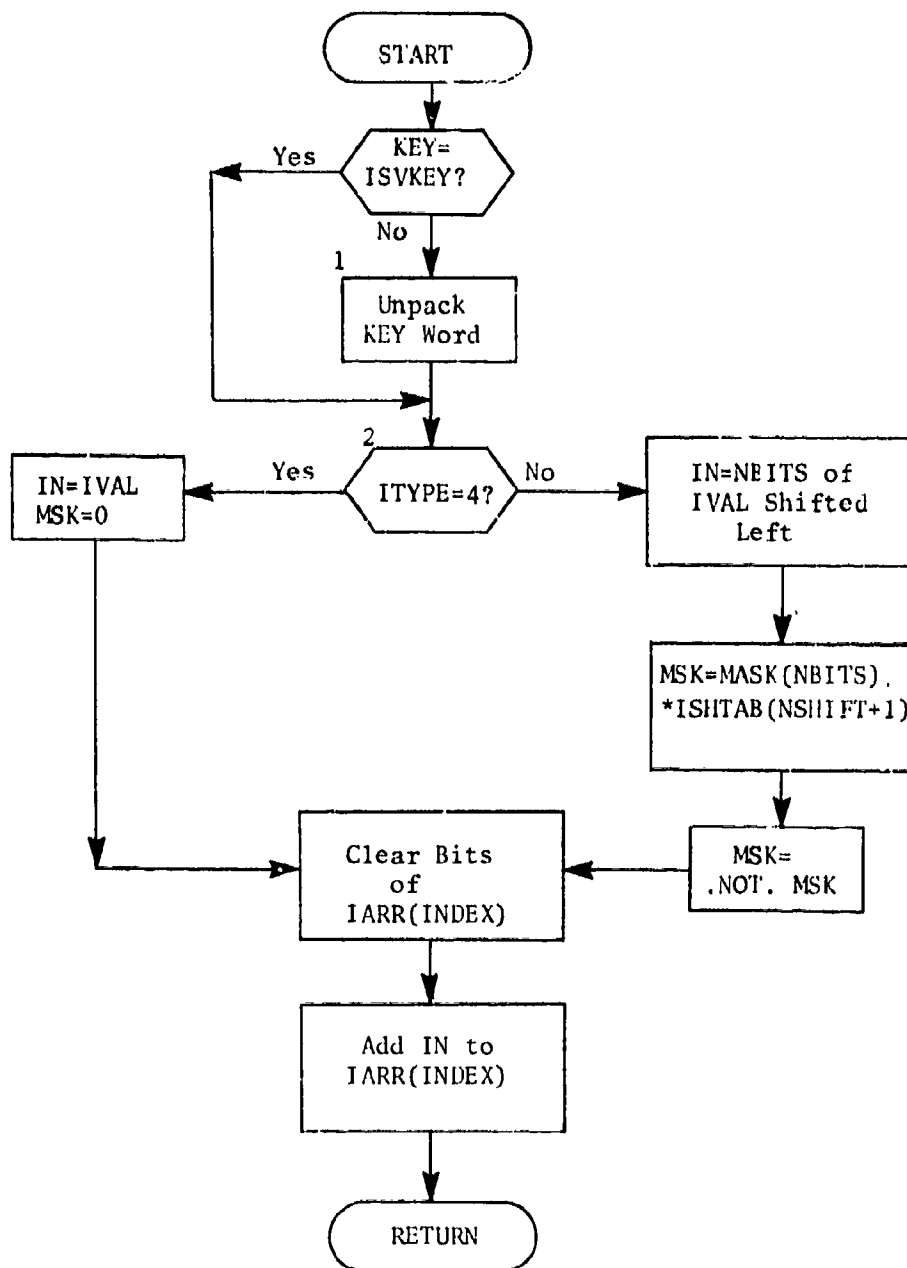


Fig. 51. Subroutine IPUT

FUNCTION ITLE

PURPOSE: To compare an input value with values in a specified array; to return an index of the first match found, and to return zero if no match is found.

ENTRY POINTS: ITLE

FORMAL PARAMETERS: NX - The value for which a match is to be found
NAR - The array with which NX is to be compared
NMB - The number of words to be scanned in NAR

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

The input search value is compared with each member of the array to be scanned. If no match is found, a value of zero is returned. Otherwise, the index of the element of the array which exactly matches is returned. In the case of more than one match, the smallest index value is returned.

Function ITLE is illustrated in figure 52.

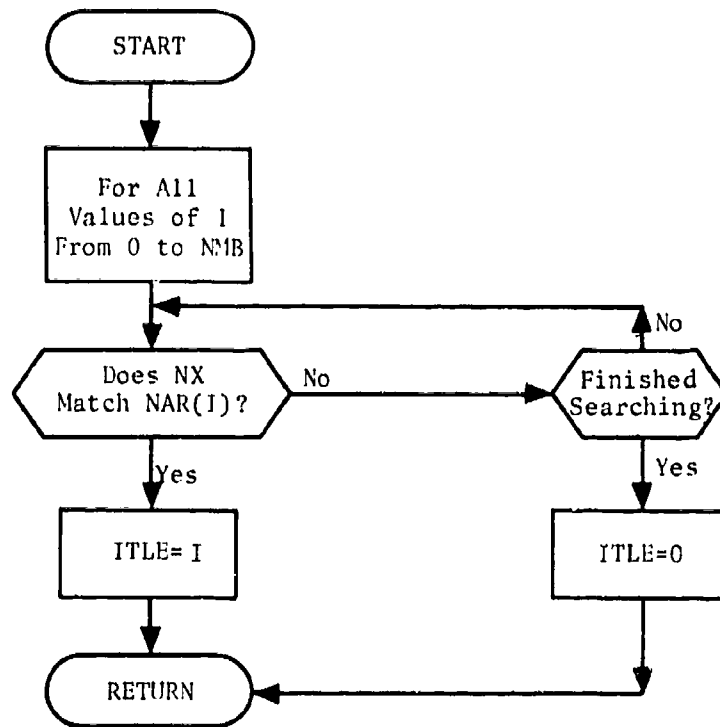


Fig. 52. Function ITLE

FUNCTION IWANT

PURPOSE: To determine the position of an item in an array. The search is modified by specifying items not to be searched.

ENTRY POINTS: IWANT

FORMAL PARAMETERS:

- NEED - The item to be matched
- IAR - The array to be searched
- ISTART - The first element in the array to be searched
- IEND - The last element in the array to be searched

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

This function searches array IAR from element ISTART to element IEND to find a match for NEED. At each occurrence of an element of IAR with the value = 1R*; i.e., one-character BCD code for asterisk right-justified with zero fill to the left, the list is not searched for NEED until after the next occurrence of the right-justified asterisk. This search method corresponds to ignoring alphameric value subfields as defined by subroutine GETVALU. The value returned by the function is the index to the element of IAR which is equal to NEED. If no element meets the prescribed conditions, the function returns the value IEND+1

Function IWANT is illustrated in figure 53.

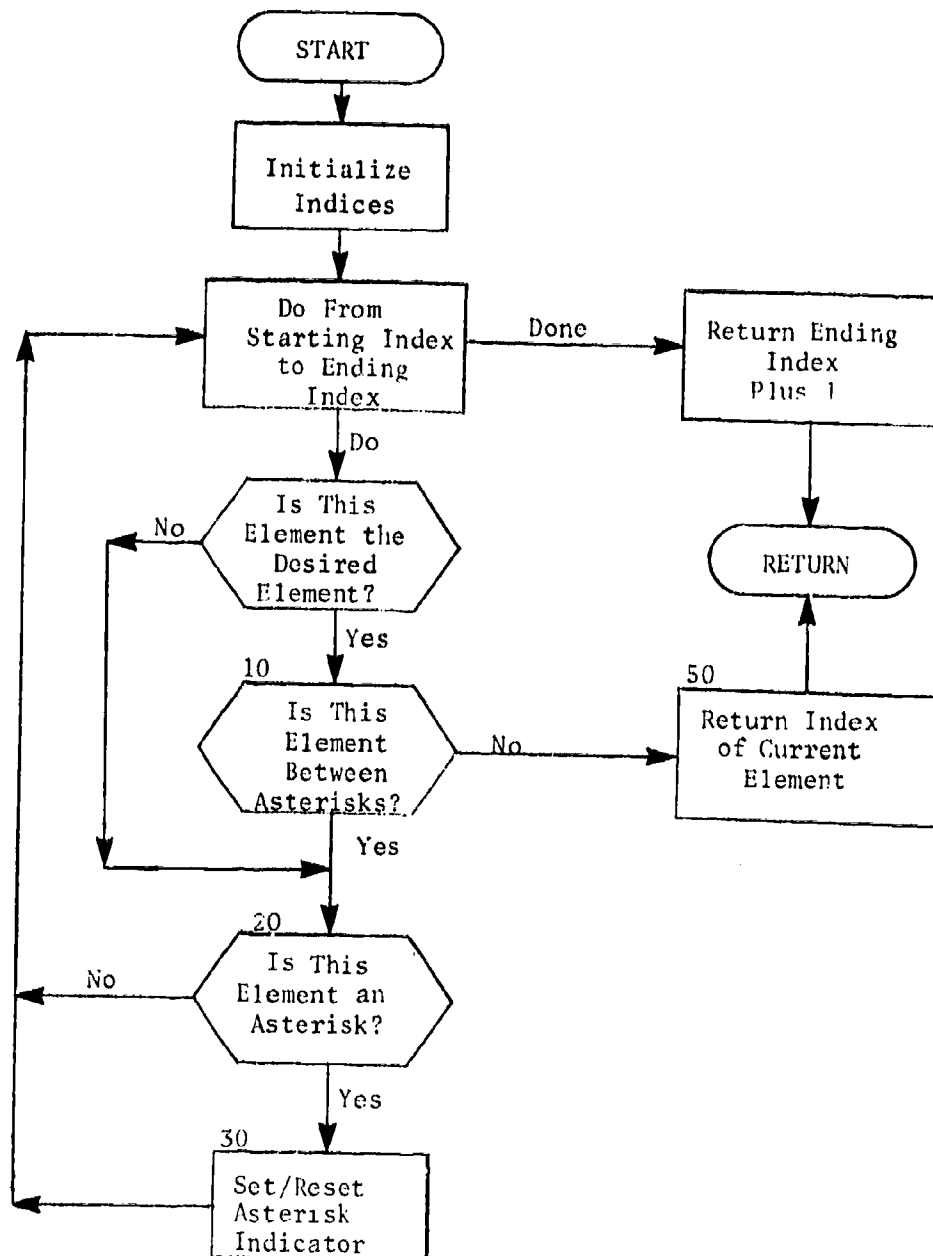


Fig. 53. Function IWANT

FUNCTION KEYMAKE

PURPOSE: To generate a KEY word for input to subroutine IPUT or function IGET.

ENTRY POINTS: KEYMAKE

FORMAL PARAMETERS:

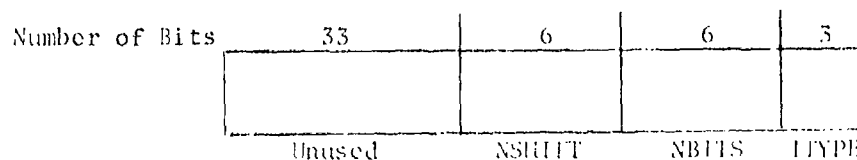
- ITYPE - Specifies the type of packing or unpacking
 - =1 Signed integer
 - =2 Unsigned integer
 - =4 Unpacked word
- NSHIFT - The number of bits from the rightmost bit of the packed word
- NBITS - The number of bits in the packed word

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

Function KEYMAKE generates a key word which contains the necessary packing and unpacking information for input to IPUT or IGET. The format of the key word is as follows:



The variable IWD is set equal to NSHIFT and shifted left six bits. NBITS is added to IWD, and IWD is shifted left three bits. ITYPE is now added to IWD and the result placed in KEYMAKE, which value is returned to the calling program.

Function KEYMAKE is illustrated in Figure 5-1.

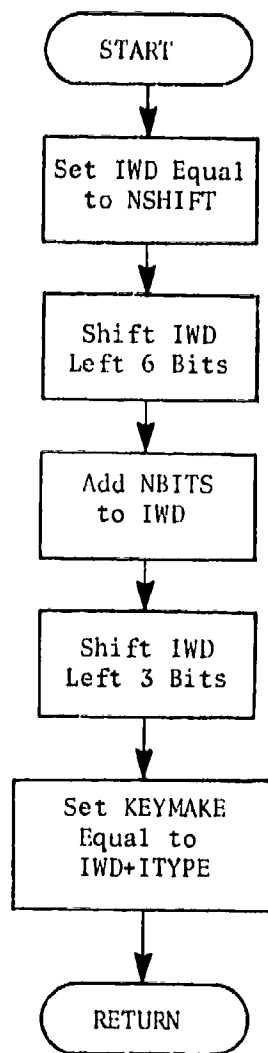


Fig. 54. Function KEYMAKE

FUNCTION LOCF

PURPOSE: To retrieve the machine address of an item (e.g., LOCREAD uses LOCF to determine the limits of core to be used in file reading and writing).

ENTRY POINTS: LOCF

FORMAL PARAMETERS: X - The name of the variable whose address is to be returned

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

This assembly language function determines the address of the variable named in the formal parameter by retrieving it from the word containing the parameters for the call to LOCF. The address is returned right-justified in the accumulator. The bank address is removed before placing the value of the machine address in the accumulator. Thus, function LOCF will not return the bank address of an item.

Function LOCF is illustrated below in Figure 55.

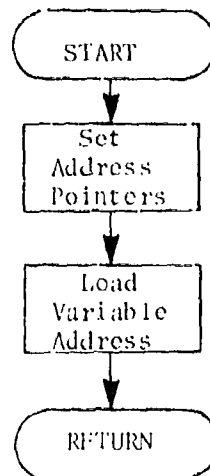


Fig. 55. Function LOCF

SUBROUTINE LOCREAD

PURPOSE: To read information from a file into, or write from, a core area specified by beginning and ending addresses, using the filehandler.

ENTRY POINTS: LOCREAD, LOCWRITE, LOCWRIT*

FORMAL PARAMETERS: A - The name of the last word stored in the block of core to be read or written
B - The name of the first word stored in the block of core to be read or written

COMMON BLOCKS: None

SUBROUTINES CALLED: LOCF, RDARRAY,** WRARRAY**

Method

This subroutine uses function LOCF to determine the limits of core to be used in the file reading or writing. The reading and writing array subroutines in the filehandler (RDARRAY and WRARRAY) are then used to transfer the data into or out of this block of storage. Note that proper usage of this subroutine requires a knowledge of the method of storage allocation by FORTRAN. The programmer should also be aware of the proper use of the filehandler.

Subroutine LOCREAD is illustrated in figure 56.

*Duplicate entry for LOCWRITE.

**See filehandler subroutines.

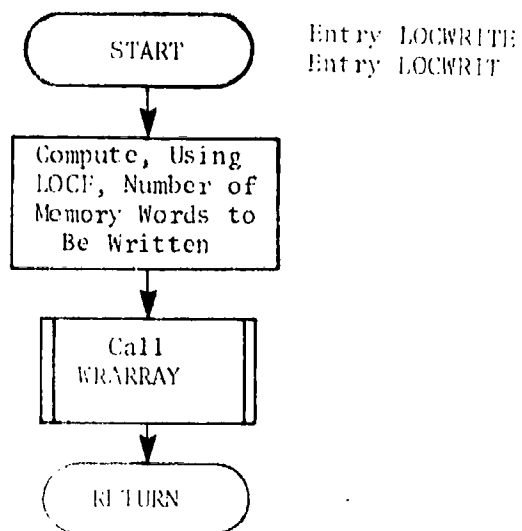
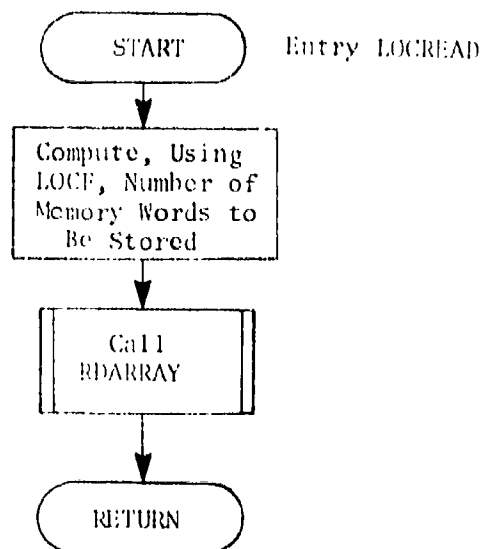


Fig. 56. Subroutine LOCREAD

SUBROUTINE NEWUNIT

PURPOSE: To unload a magnetic tape reel and release the physical unit assignment.

ENTRY POINTS: NEWUNIT

FORMAL PARAMETERS: NTOGO - Logical tape unit number

COMMON BLOCKS: None

SUBROUTINES CALLED: UNLOAD

Method

This assembly language subroutine uses the system macro UNLOAD to rewind and unload a magnetic tape reel and release the physical unit assignment. The formal parameter is set to the logical tape unit number of the tape to be unloaded and released.

Subroutine NEWUNIT is illustrated below in figure 57.

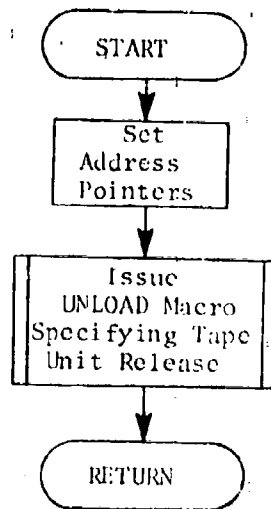


Fig. 57. Subroutine NEWUNIT

SUBROUTINE NEXTFILE

PURPOSE: To skip to the end of the current file on magnetic tape or disk, while keeping track of the number of records skipped.

ENTRY POINTS: NEXTFILE

FORMAL PARAMETERS: None

COMMON BLOCKS: ITP, IREC, FILEIN

SUBROUTINES CALLED: None

Method

This subroutine takes the logical unit of the magnetic tape to be read from common block /ITP/, which contains one variable, the logical tape unit number. If this number is negative, the file is on the disk, and the name given by the variable in common /FILEIN/ is used to retrieve the data from the disk. Common /IREC/ contains one variable which is the number of records currently read from the file. Subroutine NEXTFILE merely buffers in physical records from the tape or disk until it reaches the end of file.* During this buffering process, the number of records read is added to the variable in common /IREC/. After reaching the end of file, the subroutine returns with the file positioned to read the first record after the end of file.

Subroutine NEXTFILE is illustrated in figure 58.

*NEXTFILE positions the I/O device to the next record following the end of the current file. Although the physical location of data areas on a random access device may be in a nonsequential order, the disk I/O routines on the CDC 3800 order all sectors in a disk file sequentially. Thus, "next" record is defined for disk files in the correct manner for NEXTFILE.

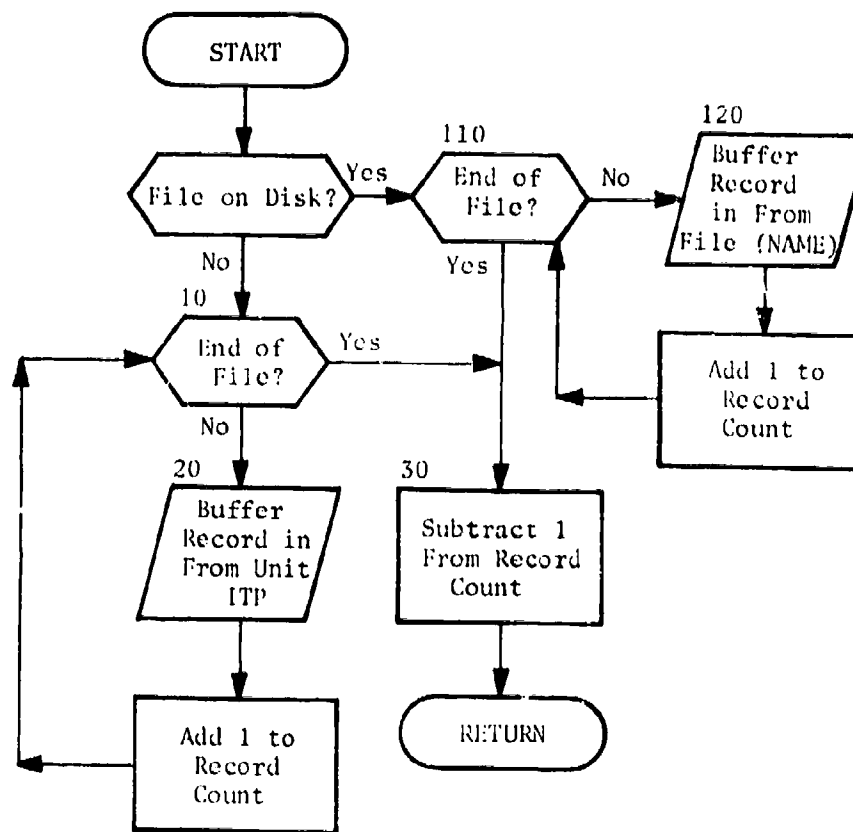


Fig. 28. Subroutine No. CH113

FUNCTION NUMGET

PURPOSE: To convert the BCD contents of an array into a signed integer number.

ENTRY POINTS: NUMGET

FORMAL PARAMETERS:

- NIN - The name of the array or variable where the input data are stored
- NCH - The number of characters to be scanned (the 3800 restricts integers to 15 decimal digits)

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

Function NUMGET (figure 59) converts integer data, which have been read into an array using alphanumeric (A8) format for each word, into signed integer numbers with blanks ignored; i. e., suppressed. The number of characters to be scanned and the array or variable name are specified in the calling sequence. If no sign appears, the number is assumed positive. However, the sign may appear anywhere in the specified input field, and NUMGET places it at the beginning of the number. If two signs appear within the same field, the last sign is assumed correct.

When integer data are read in with I format, standard FORTRAN has a number of limitations. The most annoying and time-consuming in preparing and keypunching data for a program are the requirements that integers be right-justified and that the sign (if any) precede the number. NUMGET overcomes these restrictions. It is not intended to be an error-correcting routine but rather a convenience and a time-saving routine. Whereas a normal read using integer (I) format of a number such as 77/7 or 77M7 would abort the computer run, NUMGET allows the program to continue, translating the number in each example to 77. Thus the burden of checking the accuracy of the data remains on the analyst who has input the data--but he is able to check an entire run at once, rather than having his run abort on the first illegal field encountered.

The program will scan as many contiguous characters as the calling program has requested (formal parameter NCH), moving down the array NIN (dimensioned as 1 in the subroutine, since the real size of this array is determined by

the array size in the calling program), eight characters at a time* until NCH has been satisfied. Since the 3800 has only a 48-bit word and the largest integer that can be translated accurately is $2^{15}-1$ (15 decimal digits), the practical limitation on NCH should be such that, excluding blanks, the translated integer will fit within the word size.

Perhaps the most common example for NUMGET would be the one where the field width has been defined to be 10 characters (normally read as I10 format). When using NUMGET, the calling program would read the field into an array of size two using a format of A8,A2. NUMGET would be called with NCH set to 10, and all 10 characters would be translated to an integer.

*Each word of the 3800 may contain at most eight BCD characters.

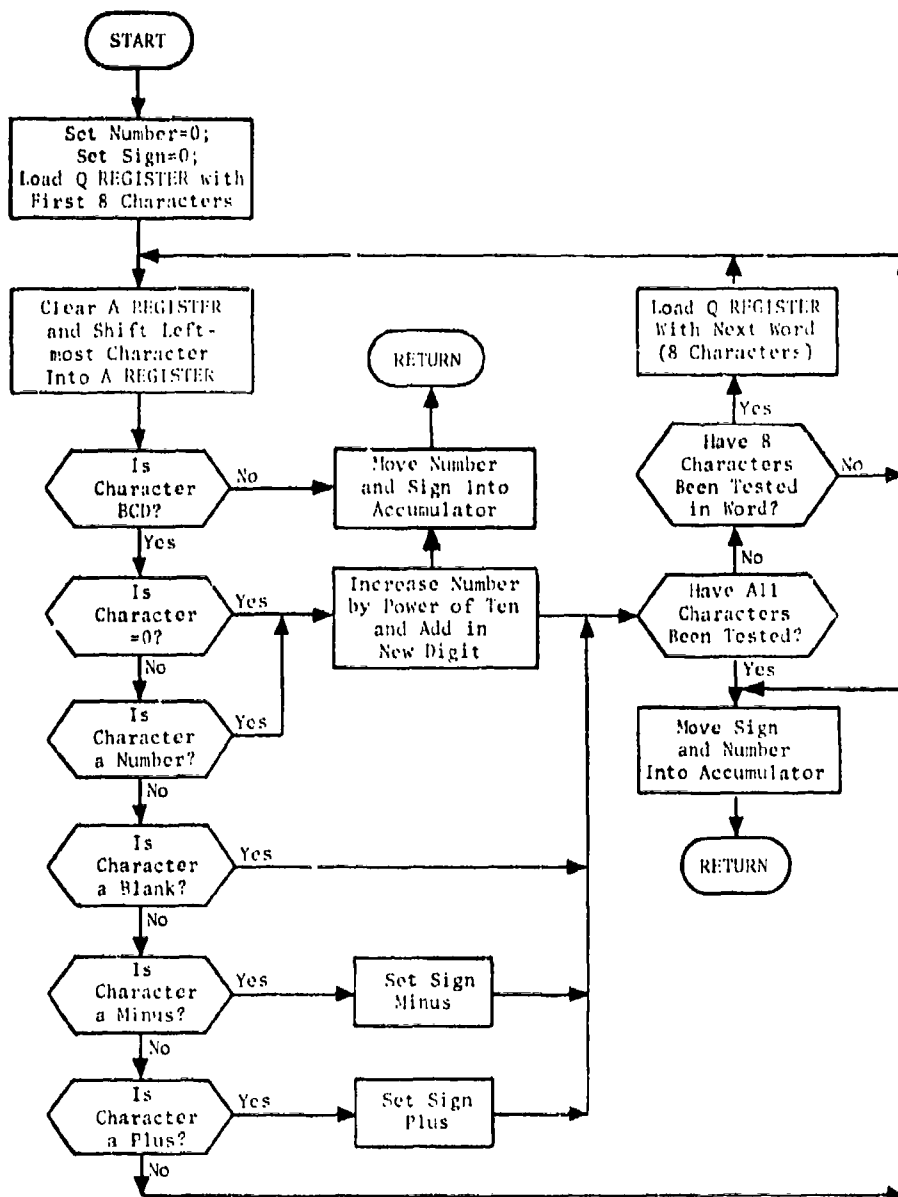


Fig. 59. Function NUMGET

SUBROUTINE ORDER

PURPOSE: To return the indices of the numeric order of the elements in any array.

ENTRY POINTS: ORDER

FORMAL PARAMETERS:

- IA - An array of arbitrary values whose elements are to be indexed
- INDEX - The array which will contain the indices to IA in increasing numerical order
- N - The number of elements in IA to be indexed

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

Given an arbitrary array, IA, of N elements, subroutine ORDER returns in another array, INDEX, the indices of the elements of IA in their numerical order. That is, INDEX(1) contains the index of the smallest element in IA, INDEX(2) the index of the next smallest, and so on. INDEX(N) contains the index of the largest element in IA.

There are restrictions on the values of the formal parameters. The number of elements to be indexed, N, must be greater than zero. The array to be ordered and the indexed array must both have at least this many elements. In addition, the subroutine will not operate correctly unless the array to be ordered and the index array occupy different areas in storage. The array to be ordered, IA, may be in either fixed point or floating point format. The indices, of course, will be returned in fixed point format.

Subroutine ORDER is illustrated in figure 60.

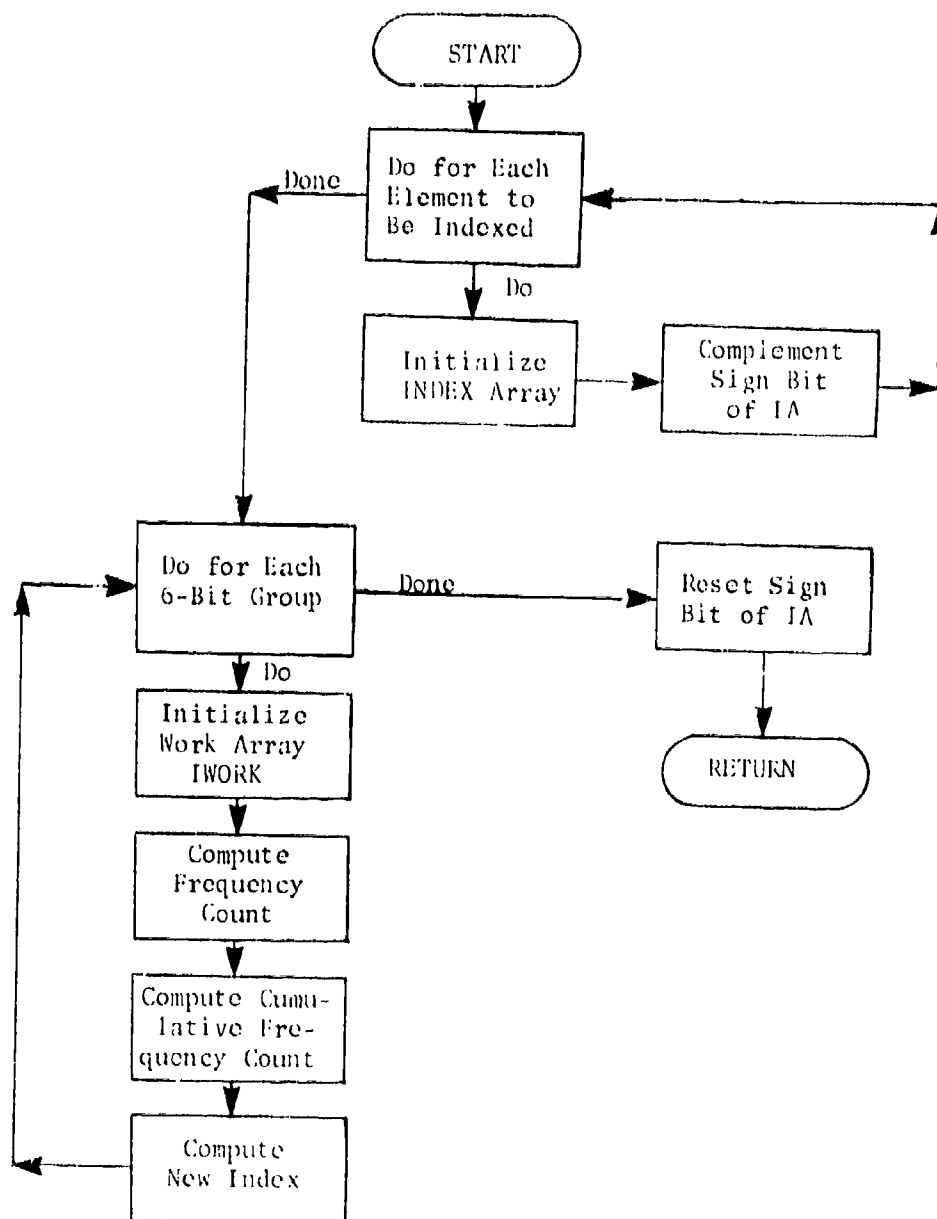


Fig. 60. Subroutine ORDER

SUBROUTINE OUTITEM

PURPOSE: To output the current item (record) to the data base file being created.

ENTRY POINTS: OUTITEM

FORMAL PARAMETERS: None

COMMON BLOCKS: EDITAPE, ITP, PROCESS, TWORD

SUBROUTINES CALLED: WRWORD*

Method

Subroutine OUTITEM uses the filehandler subroutine WRWORD to write the current item (record) on the data base file specified by JOUT in common block /EDITAPE/. The output file and the data to be output are communicated to WRWORD via common variables ITP, of common block /ITP/, and ITWORD, of common block /TWORD/.

As indicated in figure 61, OUTITEM first causes the number of attribute value pairs (variable NI, of common block /PROCESS/) associated with the item to be output. Two calls to WRWORD are then made to output each attribute-value pair, i.e., the index of the attribute within the VALUE array (common block /PROCESS/) is output followed by the attribute value. When all pairs have been written, control is returned to the calling program.

Subroutine OUTITEM is illustrated in figure 61.

*See filehandler subroutines.

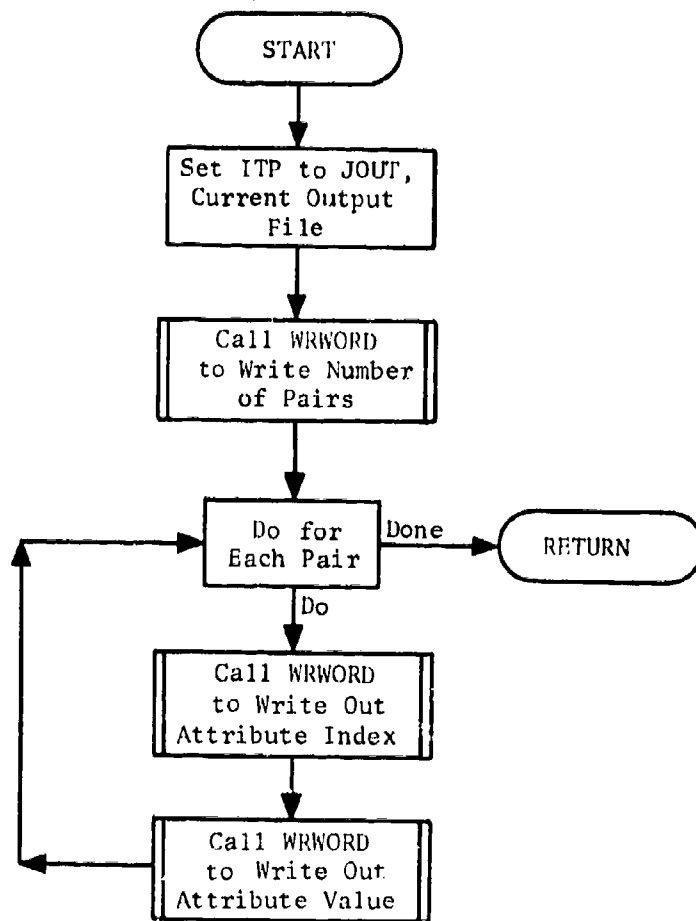


Fig. 61. Subroutine OUTITEM

SUBROUTINE OUTWORDS

PURPOSE: Used in processing a data base file to write one word on each of the specified output files.

ENTRY POINTS: OUTWORDS, OUTWRDS*

FORMAL PARAMETERS: None

COMMON BLOCKS: EDITAPE, ITP

SUBROUTINES CALLED: WRWORD**

Method

Subroutine OUTWORDS is called by the utility routine IMPITEM (NEXTITEM) to write one word on each output file specified in common block /EDITAPE/.

Subroutine OUTWORDS is illustrated in figure 62.

*Duplicate entry for OUTWORDS.
**See filehandler subroutines.

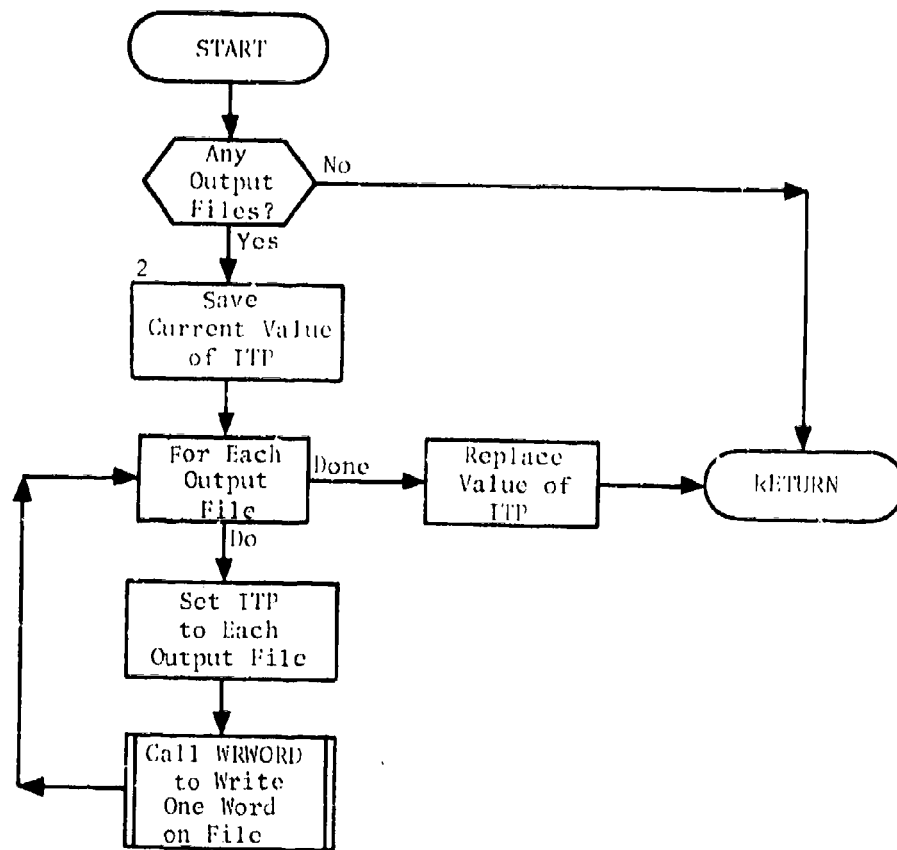


Fig. 62. Subroutine OUTWORDS

SUBROUTINE PAGESKP

PURPOSE: To eject to the top of the next page on the standard output.

ENTRY POINTS: PAGESKP, PAGESKP*

FORMAL PARAMETERS: None

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

This subroutine uses the standard carriage control character (i.e., 1) in column 1 to eject to the top of the next page on the standard output. Subroutine PAGESKP is illustrated in figure 63 below.

* Duplicate entry for PAGESKP

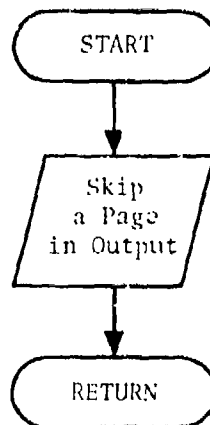


Fig. 63. Subroutine PAGESKP

SUBROUTINE PRITEM

<u>PURPOSE:</u>	To print the defined attribute-value pairs of the data base item currently being processed in a readable format.
<u>ENTRY POINTS:</u>	PRITEM
<u>FORMAL PARAMETERS:</u>	None
<u>COMMON BLOCKS:</u>	DIRECTRY, PROCESS
<u>SUBROUTINES CALLED:</u>	None

Method

This subroutine is used in preparing a printout of the data base. When called (e.g., by PRNTBASE), this subroutine examines all attributes in the directory to determine if they are defined for the data base item currently being processed. If so, the attribute name and value are stored in print arrays. If the definition is global, the pair is marked by an asterisk (*). When six pairs have been stored, the line is printed. This process is repeated until all pairs have been printed.

Subroutine PRITEM is illustrated in figure 64.

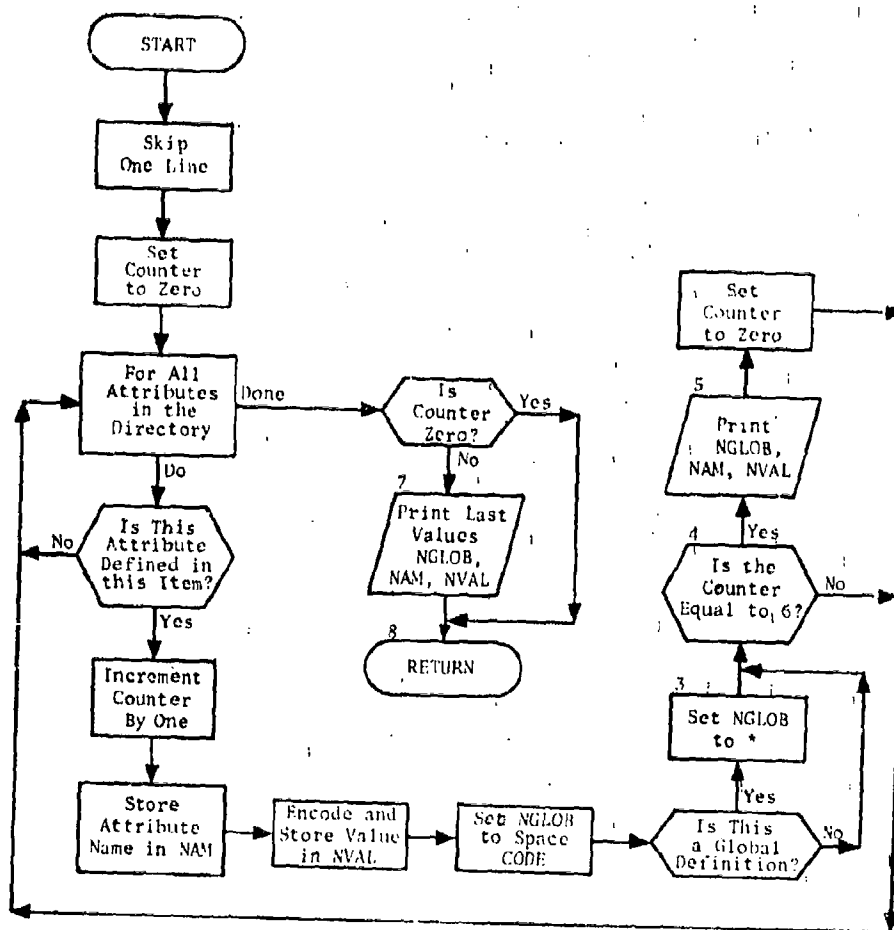


Fig. 64. Subroutine PRITEM

SUBROUTINE PRNTBASE

PURPOSE: To print the directory in readable format.
To print the attribute-value pairs defined for each data base item.

ENTRY POINTS: PRNTBASE, PRNTBAS,* PRNTBSE*

FORMAL PARAMETERS: NTP - Logical unit number of the data base file which is to be printed

COMMON BLOCKS: EDITAPE, EDITERM

SUBROUTINES CALLED: INITAPE,** INITEDIT, INPITEM, NEXTITEM, PRITEM, PRNTDIRC

Method

This subroutine prints out in readable format the entire base contained on tape unit NTP. It prints out the entire directory using subroutine PRNTDIRC. Then each item in the data base is printed by successive calls on subroutine PRITEM. For each item the name of every attribute defined, either globally or locally, for that item is printed together with its value. In addition, an asterisk is printed beside each attribute whose value is currently determined by a global definition.

PRNTBASE first saves NOUT, the number of output tapes, and then sets it to zero. Then through a series of calls on INITEDIT, PRNTDIRC, INPITEM, and PRITEM, the entire data base is printed. NOUT is restored to its original value, and the routine exits.

Subroutine PRNTBASE is illustrated in figure 65.

*Duplicate entries for PRNTBASL.

**See filehandler subroutines.

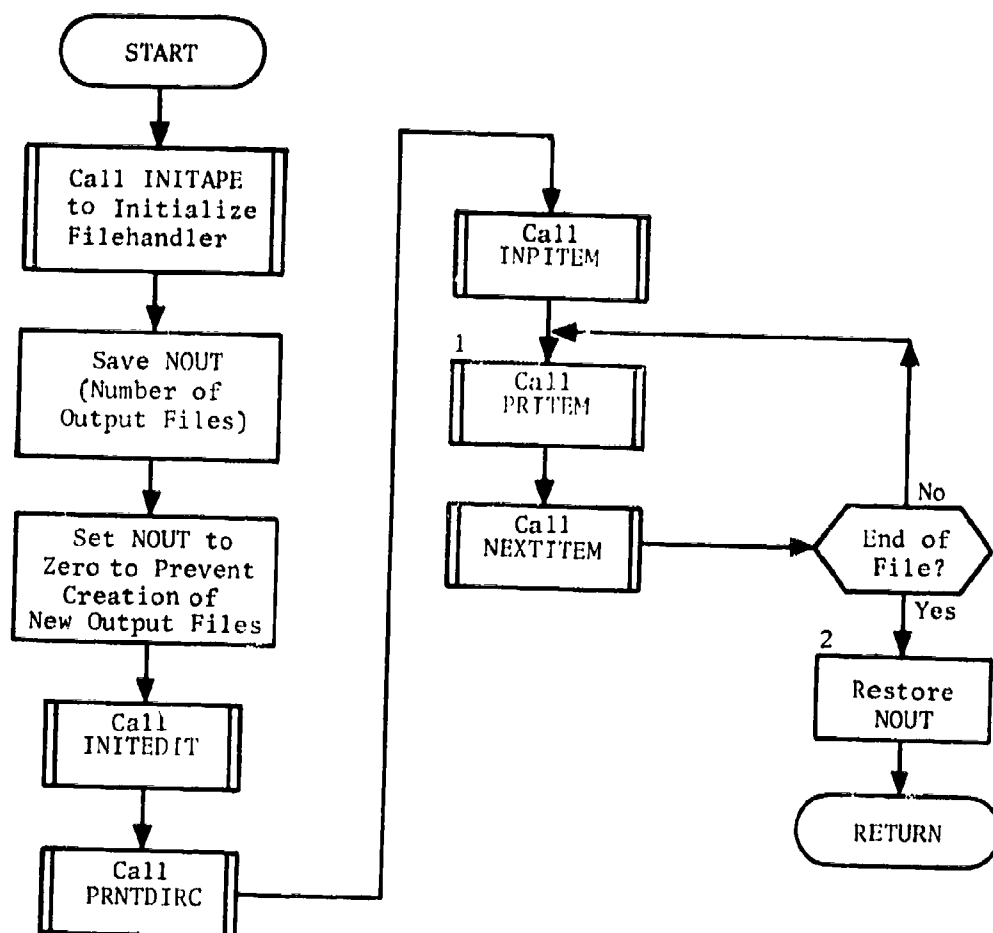


Fig. 65. Subroutine PRNTBASE

SUBROUTINE PRNTDIRC

PURPOSE: To print the directory contents in readable format.

ENTRY POINTS: PRNTDIRC, PRNTDRC (duplicate entry)

FORMAL PARAMETERS: None

COMMON BLOCKS: DIRECTRY

SUBROUTINES CALLED: PAGESKP

Method

Subroutine PRNTDIRC is called to print the contents of the data base directory. The directory consists of a list of all the attributes which can be used to describe the data items defined in the data base. In addition to the mnemonic name assigned to the attribute, the directory includes information about the range of values which may be assigned to the attribute and the value which is given it when it is not specifically defined for an item; i.e., the default value. Information is also included for checking the data input to the data base.

When called, subroutine PRNTDIRC (figure 66) prints all of the information contained in the directory for each attribute. This includes the attribute name, print format, checking code, default value, and the attribute value range limits or a list of allowed values specified for the attribute. The arrays which are examined and printed by PRNTDIRC are:

1. ATTNAME: This array contains the name of each defined attribute. The position of the name in this array becomes the index to all other arrays holding information about this attribute and is the index which is stored with the attribute value in the item portion of the data base tape. A data base file is completely dependent on the order of attributes in the directory.
2. IFORMAT: This array contains the conversion format (e.g., I8) to convert the value of the attribute to output (printer) format.
3. ICODE: The array code numbers (between one and seven) which specify the type value (e.g., floating point number) for each attribute and the method of checking (range, list, or none). The codes currently used are indicated below.

<u>ICODE</u>	<u>TYPE VALUE</u>	<u>CHECKING</u>
1	Floating point numeric	Range (MIN-MAX)
2	Floating point numeric	LIST
3	Fixed point numeric	Range (MIN-MAX)
4	Fixed point numeric	LIST
5	Alphanumeric	LIST
6	Alphanumeric	NOCHECK (none required)
7	Special (see Directory Conventions below)	Range (MIN-MAX)

4. N1, N2 (FN1, FN2): These arrays contain either the minimum or maximum values for the attributes in the case of range checking, or the beginning and ending indices to a list of all possible values for the attributes that specify list checking. These arrays are unused when no checking is called for.
5. DEFAULT: This array contains values which are to be associated with any attribute when it is not defined for an item.
6. LISTCHK: This logical array is set to "true" for list checking and false for range checking or when no checks are required.
7. LISTVALS: This array contains the acceptable values for an attribute to be used in list checking. In this mode of checking, the attribute value is restricted to a specified list. For example, acceptable values of the attribute SIDE might be limited to RED and BLUE.

The subroutine first retrieves the attribute name and determines if there is a default value associated with it. If there is, the default value is encoded using the format statement for the selected attribute. If there is no default value, the word BLANK is substituted for the value in the print line.

The routine next ascertains if list checking has been specified. If it has, the word LIST is inserted in the output line. If list checking was not applicable, the routine determines if "no checking" (i.e., ICODE=6) was specified for the attribute. If so, the word NOCHECK is inserted in the output line. If neither the list nor "no checking" options were specified, it is presumed that range checking is desired. In this case, the minimum and maximum values specified for the attribute value are retrieved from the arrays

N1 and N2. These values are printed using the format obtained from the IFORMAT array. Should N1 and/or N2 be found to contain blanks, the word BLANK is inserted in the print line. The printed line, assembled by one of the above procedures, is of the form: attribute name, conversion format, code for checking, default value, first check word, and second check word.

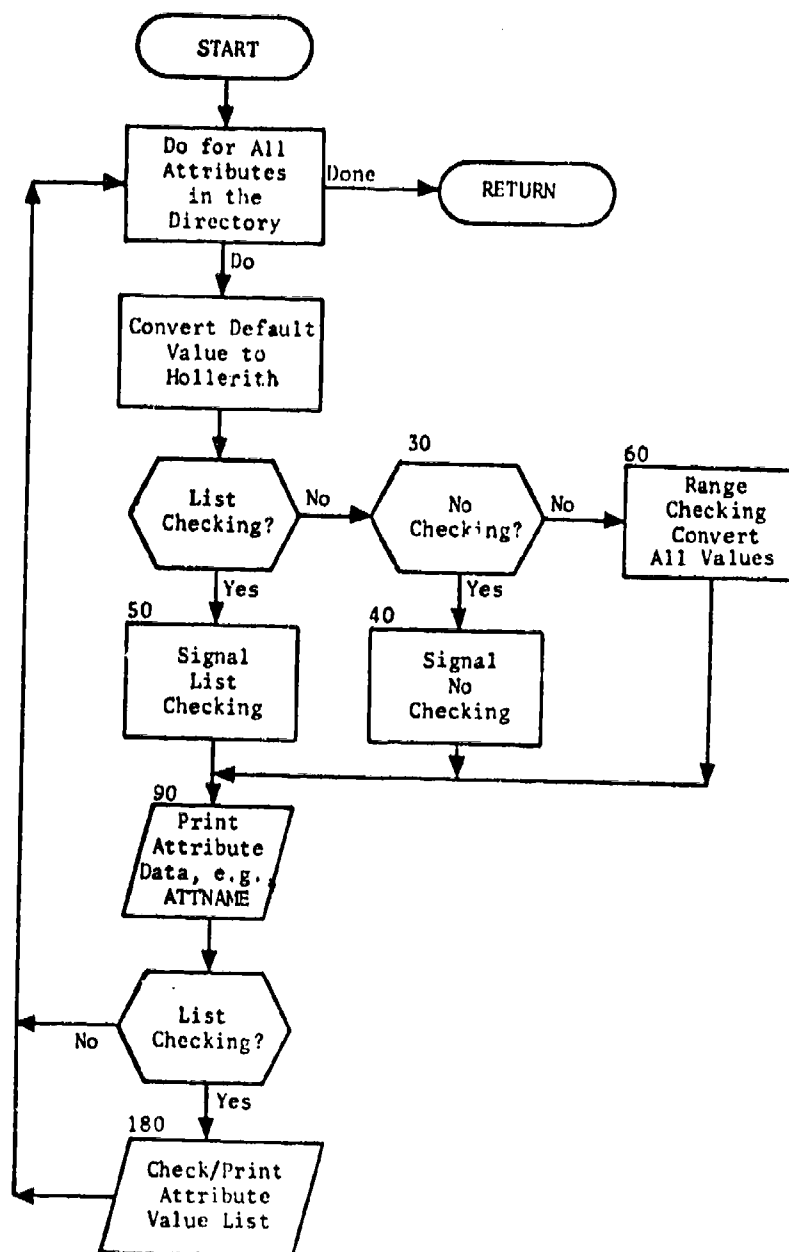


Fig. 66. Subroutine PRNTDIRC

SUBROUTINE PRNTDTA

PURPOSE: To print the contents of a data base file in a compact and readable format.

ENTRY POINTS: PRNTDTA, PRNTDATA (duplicate entry)

FORMAL PARAMETERS: NT1 - Logical unit number of the file containing the data base

COMMON BLOCKS: DIRECTRY, EDITAPE, EDITERM, PRNTCOMM, PROCESS

SUBROUTINES CALLED: INITAPE,* INITEDIT, INPITEM, NEXTITEM, PAGESKP, PRNTDIRC, PRNTPGE

Method

This subroutine prints the contents of any data base file in as compact and readable a format as possible.

Attributes which have the same value for items on a print page are printed along with the value in a COMMON ATTRIBUTE block at the top of the page. Attributes which have different values for items on a print page are printed in columns in an OTHER ATTRIBUTE block beneath the COMMON ATTRIBUTE block. The column headings are the attribute names, with each item requiring a separate print line. If an attribute is defined for some but not all items on the page, a BCD slash (/) is inserted in the undefined locations. If an attribute is undefined for all attributes on a page, it is not printed. A print page is complete and another begun when:

- (1) the number of items on a page reaches 35,
- (2) the number of columns in the OTHER ATTRIBUTE block exceeds 11, or
- (3) a class change occurs.

Subroutine PRNTDTA is illustrated in figure 67.

*See filehandler subroutines.

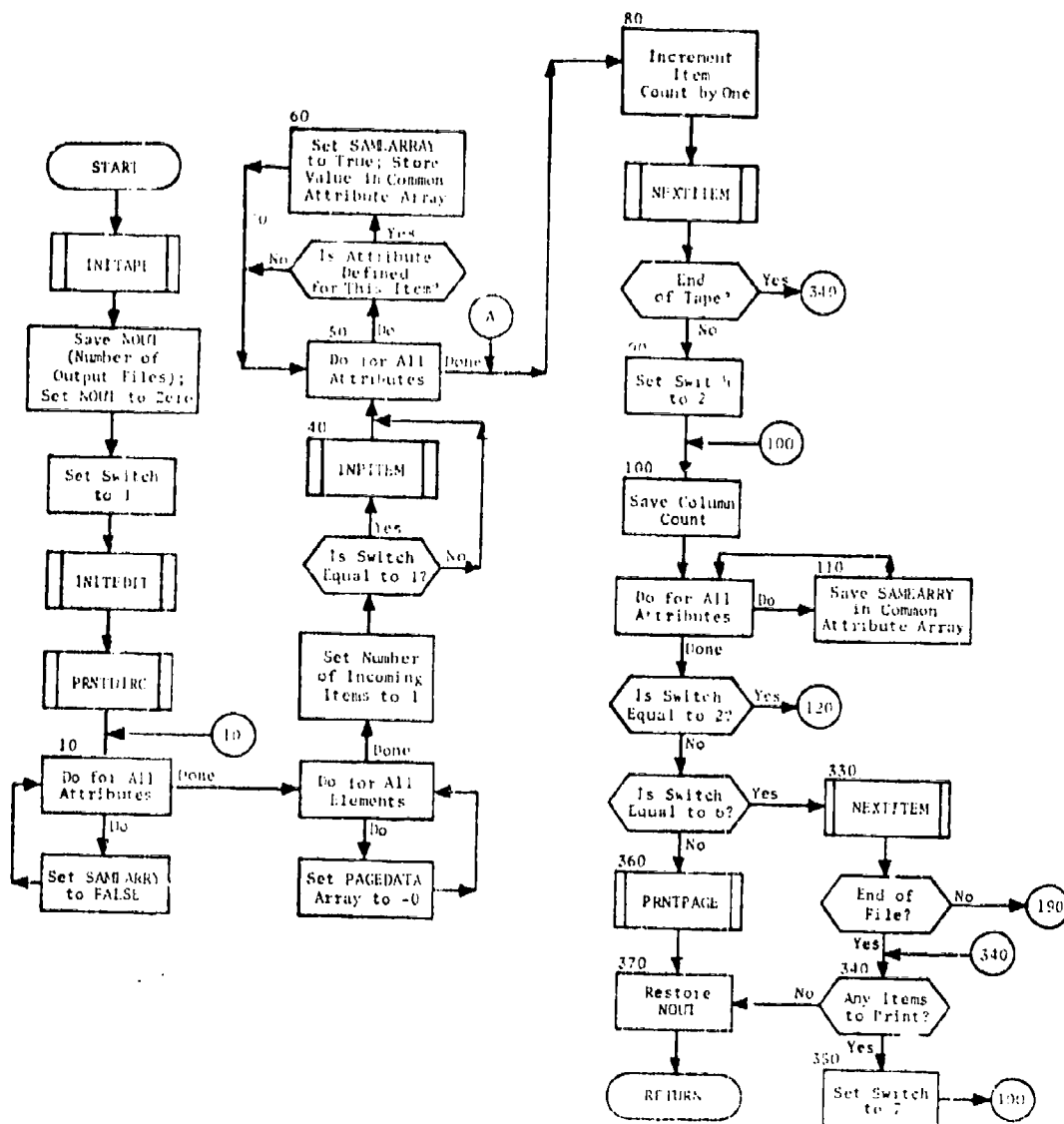


Fig. 67 Subroutine PRNTDTA
(Sheet 1 of 2)

SUBROUTINE PRNTPGE

PURPOSE: To perform the printing of data base information as directed by subroutine PRNTDTA.

ENTRY POINTS: PRNTPGE, PRNTPAGE (duplicate entry)

FORMAL PARAMETERS: J - Number of attributes defined on this page

COMMON BLOCKS: DIRECTRY, PROCESS, PRNTCOMM

SUBROUTINES CALLED: PAGESKP

Method

This subroutine prints a page of data base information, as directed by subroutine PRNTDTA. The variables in common /PRNTCOMM/ are interpreted to determine the number of "common" and "other" attributes. These are printed on the standard input in the following format.

Attributes which have the same value for items on a print page are printed along with the value in a COMMON ATTRIBUTE block at the top of the page. Attributes which have different values for items on a print page are printed in columns in an OTHER ATTRIBUTE block beneath the COMMON ATTRIBUTE block. The column headings are the attribute names, with each item requiring a separate print line. If an attribute is defined for some but not all items on the page, a BCD slash (/) is inserted in the undefined locations.

Subroutine PRNTPGE is illustrated in figure 68.

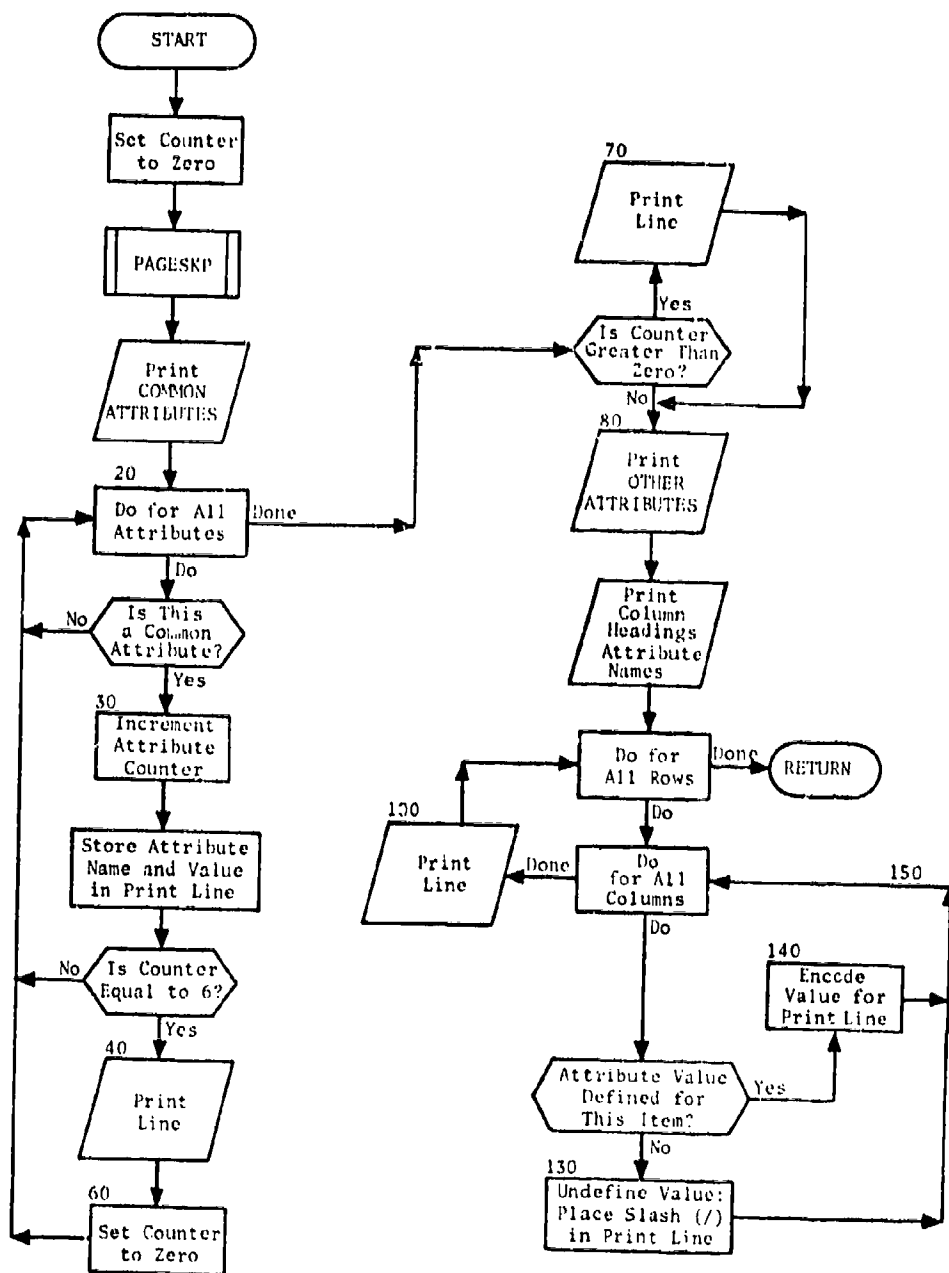


Fig. 68. Subroutine PRNTPGE

SUBROUTINE READDIR

PURPOSE: To read a directory contained on a data base file into common block /DIRECTRY/.

ENTRY POINTS: READDIR

FORMAL PARAMETERS: ITAPX - Logical unit number of the file containing the directory

COMMON BLOCKS: DIRECTRY, ERRORM, ITP, TWORD

SUBROUTINES CALLED: ABORT, RDARRAY*, RDWORD*, SETREAD*

Method

Subroutine READDIR reads the directory from unit ITAPX, which is assumed to contain a standard data base, into memory in common block /DIRECTRY/.

This subroutine is shown in figure 69. The routine begins by setting the buffer number ITP to indicate the logical unit number of the file containing the directory. The filehandler subroutine SETREAD is then called to initialize this file for reading. Subroutine RDWORD is then called twice to read the first two words from the directory. The first word is stored in IDEF and specifies the number of attributes contained in the directory. The second word is stored in LASTLIST and indicates the number of entries in the value list array LISTVALS. The routine then computes the number of words in the logical arrays LISTCHK and GLOB (equivalenced to LOG1 and LOG2).

A series of calls on subroutine RDARRAY are then made to read in the contents of the arrays described below.

1. ATTNAME: This array contains the name of each defined attribute. The position of the name in this array becomes the index to all other arrays holding information about this attribute and is the index which is stored with the attribute value in the item portion of the data base tape. A data base file is completely dependent on the order of attributes in the directory.
2. IFORMAT: This array contains the conversion format (e.g., I8) to convert the value of the attribute to output (printer) format.
3. ICODE: The array code numbers (between one and seven) which specify the type value (e.g., floating point number) for each attribute and

Associated filehandler subroutines:

the method of checking (range, list, or none).

4. DEFAULT: This array contains values which are to be associated with any attribute when it is not defined for an item.
5. N1, N2: These arrays contain either the minimum or maximum values for the attributes in the case of range checking, or the beginning and ending indices to a list of all possible values for the attributes that specify list checking. These arrays are unused when no checking is called for.
6. LISTCHEK: This logical array is set to "true" for list checking and "false" for range checking or when no checks are required.
7. GLOB: This logical array is set to "true" if a global definition is in force; otherwise, it is set to "false."
8. LISTVALS: This array contains the acceptable values for an attribute to be used in list checking. In this mode of checking, the attribute value is restricted to a specified list.

After transferring these data, the routine checks to ensure that the directory as read was of the proper length. This is accomplished by determining if the next word read from the input file is ENDDIRECT. If so, control is returned to the calling program. If not, an error message is printed and subroutine ABORT is called to terminate the run. The occurrence of this error signifies either a machine or file error during the process of reading the directory or an incorrectly mounted file.

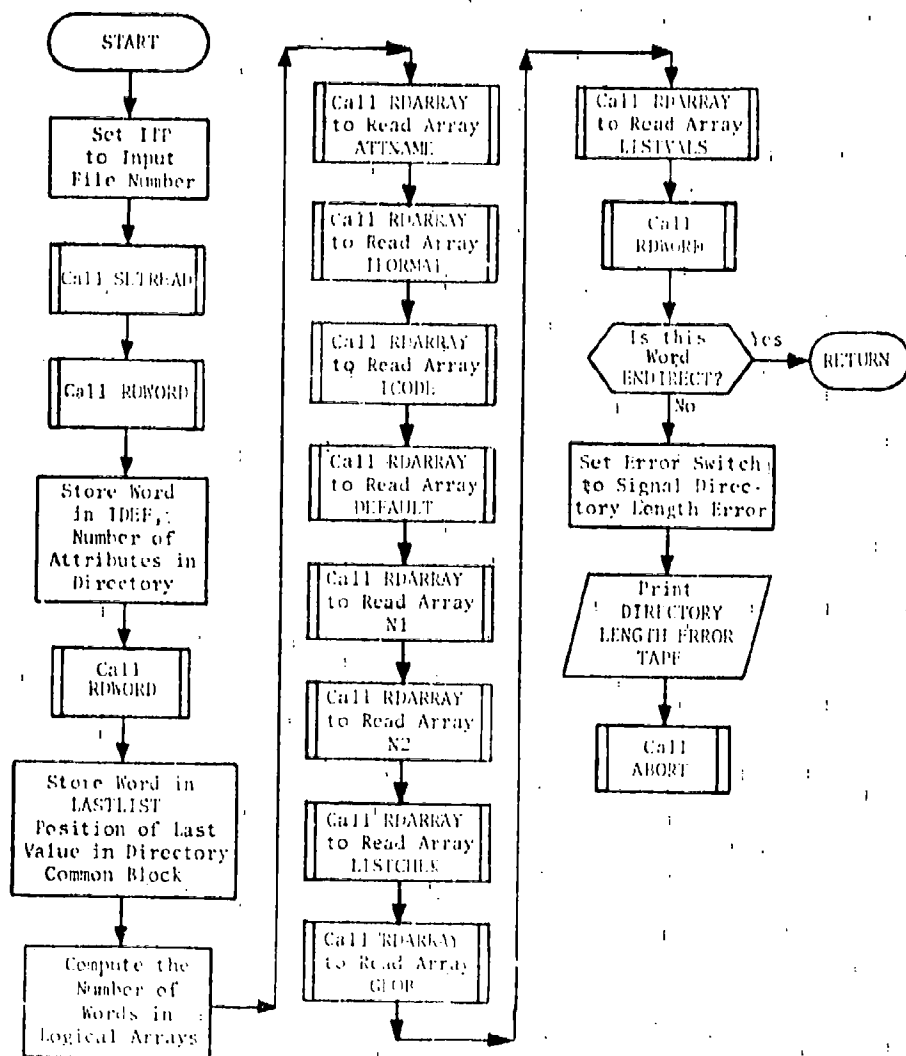


Fig. 69. Subroutine READDIR

SUBROUTINE REORDER

PURPOSE: To rearrange from one to seven arrays into the sequence specified by the elements of an index array.

ENTRY POINTS: REORDER

FORMAL PARAMETERS:

- ISEQ - (Fixed point) sequence key array of the type produced by subroutine ORDER
- NEL - (Fixed point) number of elements to be reordered in each array
- NAR - (Fixed point) number of arrays which are to be reordered
- L1-L7 - (Fixed point or floating point) names of the arrays which are to be reordered; if the number of these arrays is less than seven, the remaining positions must be filled by trailing dummy arguments

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

Subroutine REORDER operates in the following manner. First, it stores one element from each array in a temporary location. It then reads from the array ISEQ the element which should go in that position (which may now be considered empty) and moves it, filling that position and creating a new blank. This is repeated for the corresponding element of each array being reordered. Each new blank spot is filled with its proper contents as soon as its original contents have been moved, until the element currently in temporary storage is required. When this happens, subroutine REORDER finds another element which is not already in its sequence and puts it into temporary storage and continues as before until no elements are out of sequence.

The contents of the array ISEQ are returned to the calling program unchanged, so that subroutine REORDER may be called again, using the same sequence key array if more than seven arrays are to be reordered.

Subroutine REORDER is illustrated in figure 70.

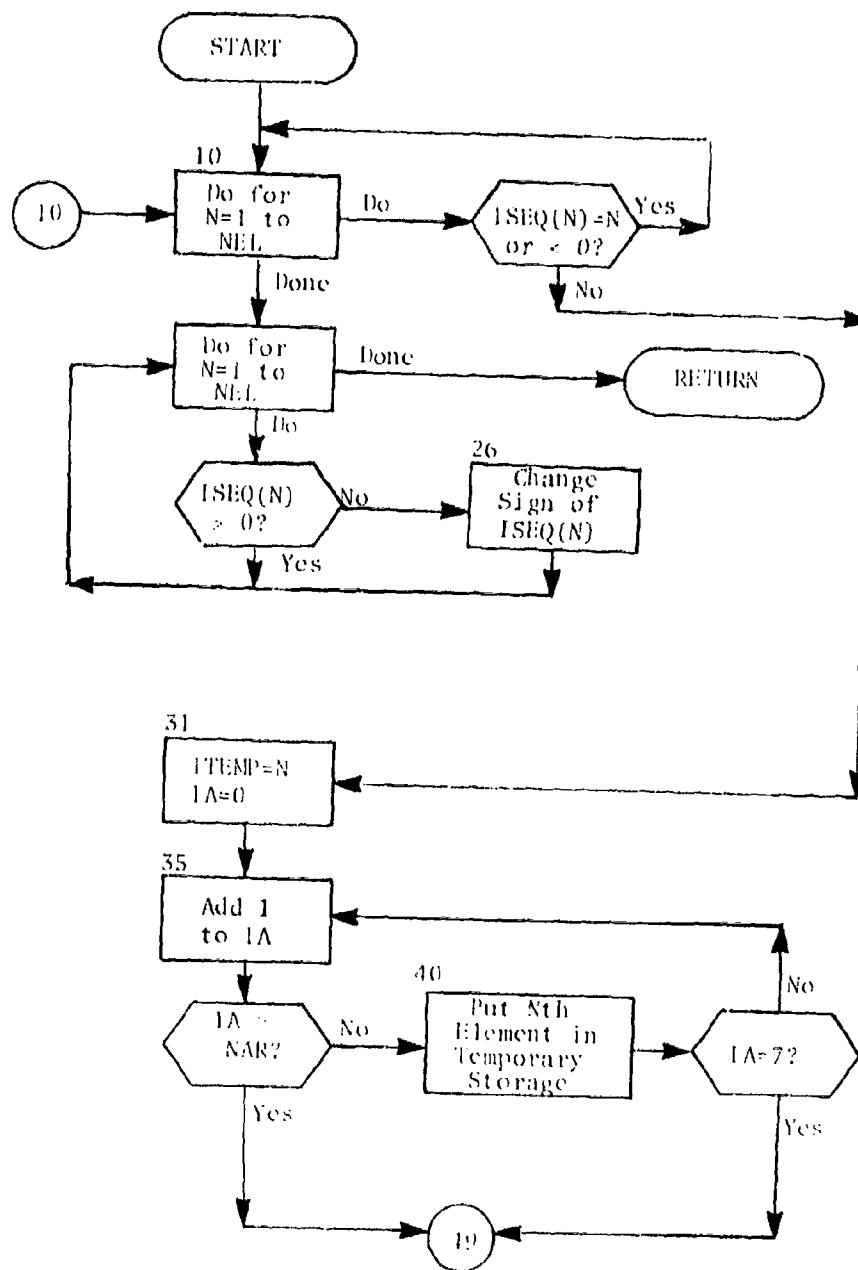


Fig. 70. Subroutine REORDER
(Sheet 1 of 2)

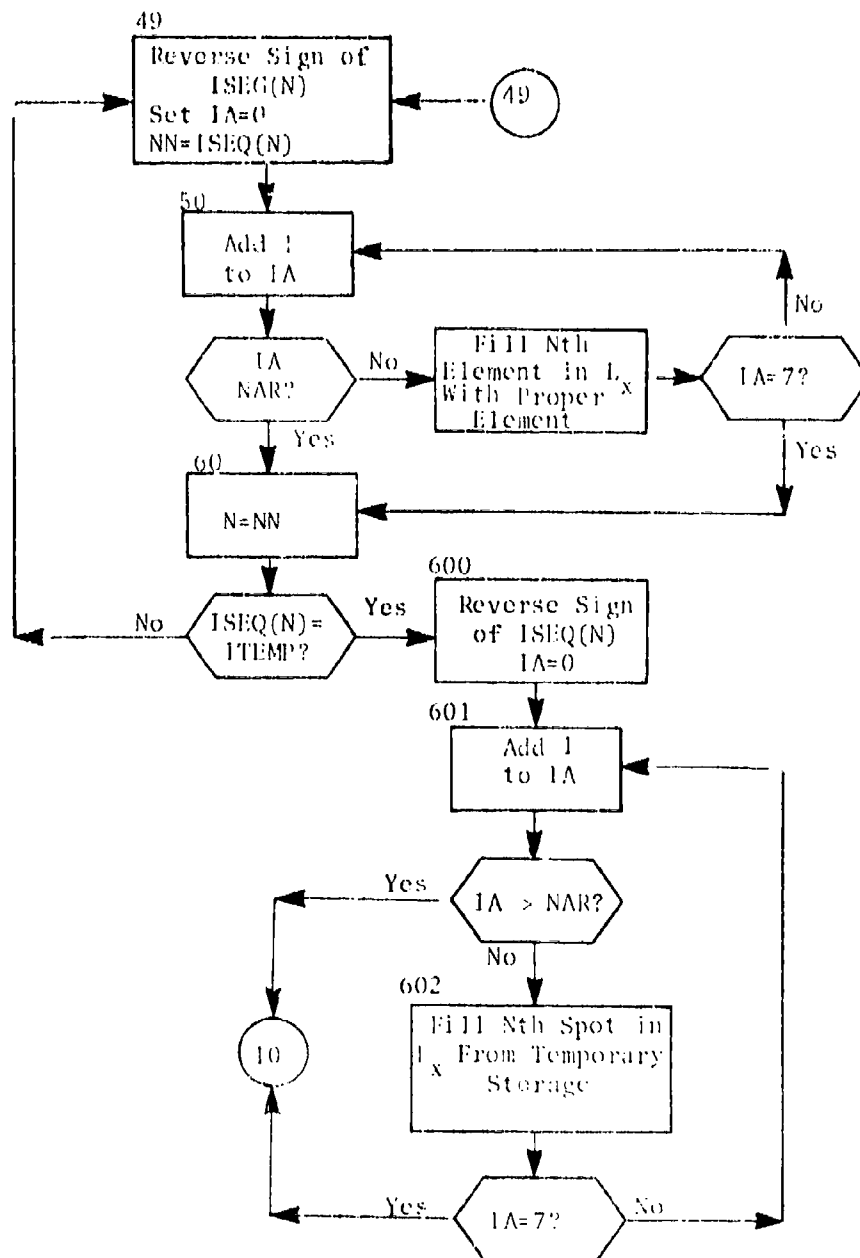


Fig. 70. (cont.)
(Sheet 2 of 2)

SUBROUTINE SKIP

PURPOSE: To skip a number of words on an input file using the filehandler.

ENTRY POINTS: SKIP

FORMAL PARAMETERS: NWORDS - The number of words to be skipped on the tape

COMMON BLOCKS: None

SUBROUTINES CALLED: RDARRAY*

Method

This subroutine is used to ignore (skip) a number of words on an input file while that file is being read by the filehandler. The subroutine assumes the file is in read status. This subroutine has a small amount of internal storage used for temporary data storage. Subroutine SKIP merely calls the filehandler subroutine RDARRAY to fill and refill this temporary storage area until the required number of words have been read. This leaves the input file in position to read the next word of data after skipping the number of words specified in the formal parameter.

Subroutine SKIP is illustrated in figure 71. As indicated, RDARRAY reads 100 words per call except for the final call in which NREM words are read ($NWORDS = (M \times 100) + NREM$).

*See filehandler subroutines.

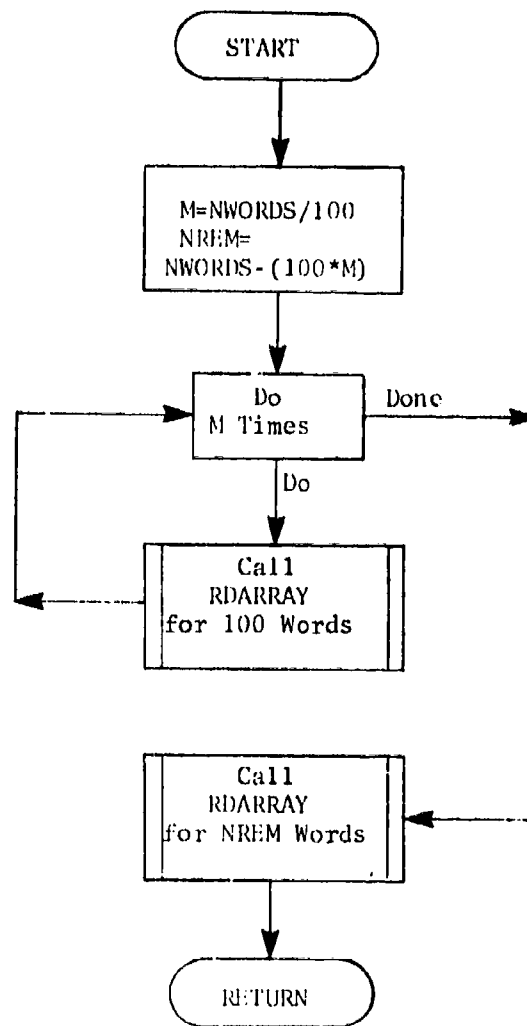


Fig. 71. Subroutine SKIP

SUBROUTINE SKIPFILE

PURPOSE: To skip to the end of a file on a magnetic tape.

ENTRY POINTS: SKIPFILE, BACKFILE

FORMAL PARAMETERS: LTN - Logical tape unit number

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

This subroutine merely reads records from the tape unit specified in the formal parameter LTN until it encounters an end-of-file mark. It then returns control to the calling program, leaving the tape in position to read or write the first record following the end-of-file.

Entry BACKFILE is used to read the tape backwards until the previous end-of-file mark is reached. When this entry is used, the tape is positioned at the beginning of the end-of-file mark.

This subroutine is part of the CDC 3800 SCOPE system.

FUNCTION SSKPC

PURPOSE: To determine the single shot kill probability assuming a circular normal delivery distribution.

ENTRY POINTS: SSKPC

FORMAL PARAMETERS: MOD - Integer (currently 1, 3, or 6); MOD determines the number of terms used to approximate SSKPC
 A - Lethal kill radius (positive real number)
 SXY - Standard deviation
 RO2 - Offset aim point distance, squared

COMMON BLOCKS: None

SUBROUTINES CALLED: None

Method

SSKPC determines the single shot kill probability PKW using the following formula:

$$PKW = \frac{e^{(-RO2/2 \cdot SXY^2 + Z)}}{2 \cdot SXY^2 \cdot \beta} \left[1 + \left(\frac{\gamma}{\beta}\right) (Z + 1) + \frac{\left(\frac{\gamma}{\beta}\right)^2}{2!} (Z^2 + 4Z + 2) + \frac{\left(\frac{\gamma}{\beta}\right)^3}{3!} (Z^3 + 9Z^2 + 18Z + 6) + \frac{\left(\frac{\gamma}{\beta}\right)^4}{4!} (Z^4 + 16Z^3 + 72Z^2 + 96Z + 24) + \frac{\left(\frac{\gamma}{\beta}\right)^5}{5!} (Z^5 + 25Z^4 + 200Z^3 + 600Z^2 + 600Z + 120) \right]$$

where $\gamma = \frac{MOD}{\Lambda^2}, \quad \beta = \frac{MOD}{\Lambda^2} + \frac{1}{2 \cdot SXY^2}$

and $Z = \frac{RO2}{4 \cdot \beta \cdot SXY^4}$

The flow diagram in figure 72 indicates the way in which this formula is used to calculate PKW for different values of MOD (MOD is the equivalent of the parameter W as described under Derivation of Kill Probability Function, Analytical Manual, Volume II). Note that BETA = β , GAM = γ , GB = GAM/BETA = γ/β , and

$$C = \frac{e^{(-RO2/2 \cdot SXY^2 + Z)}}{2 \cdot SXY^2 \cdot BETA}$$

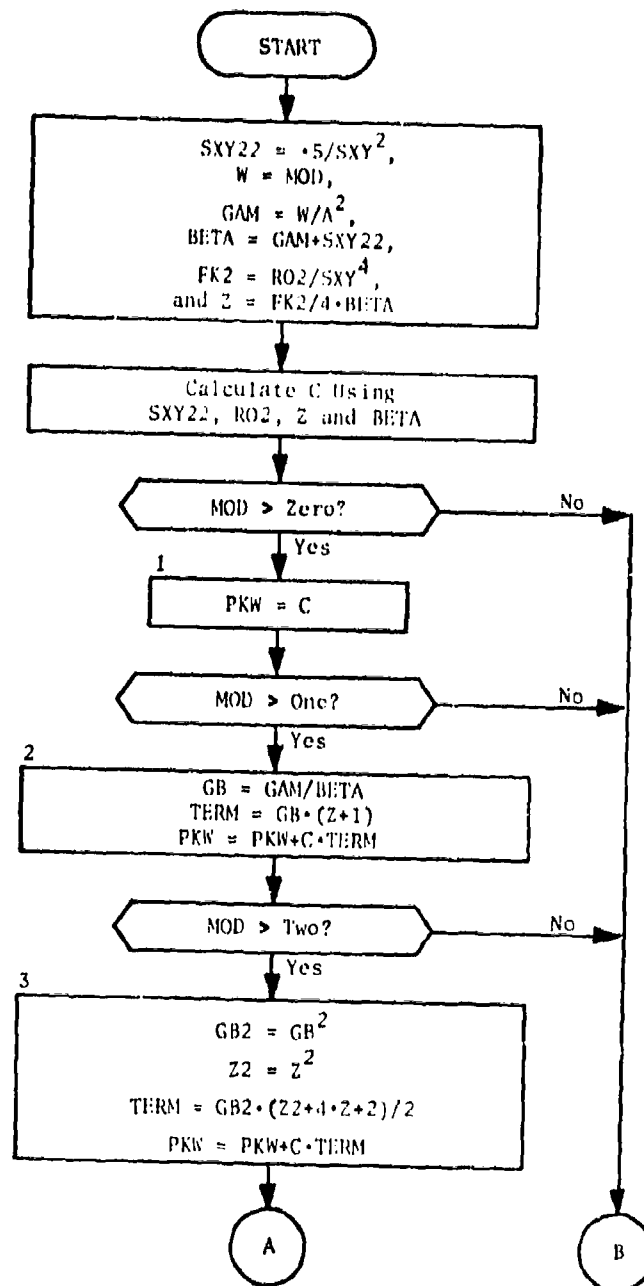


Fig. 72. Function SSKPC
(Sheet 1 of 2)

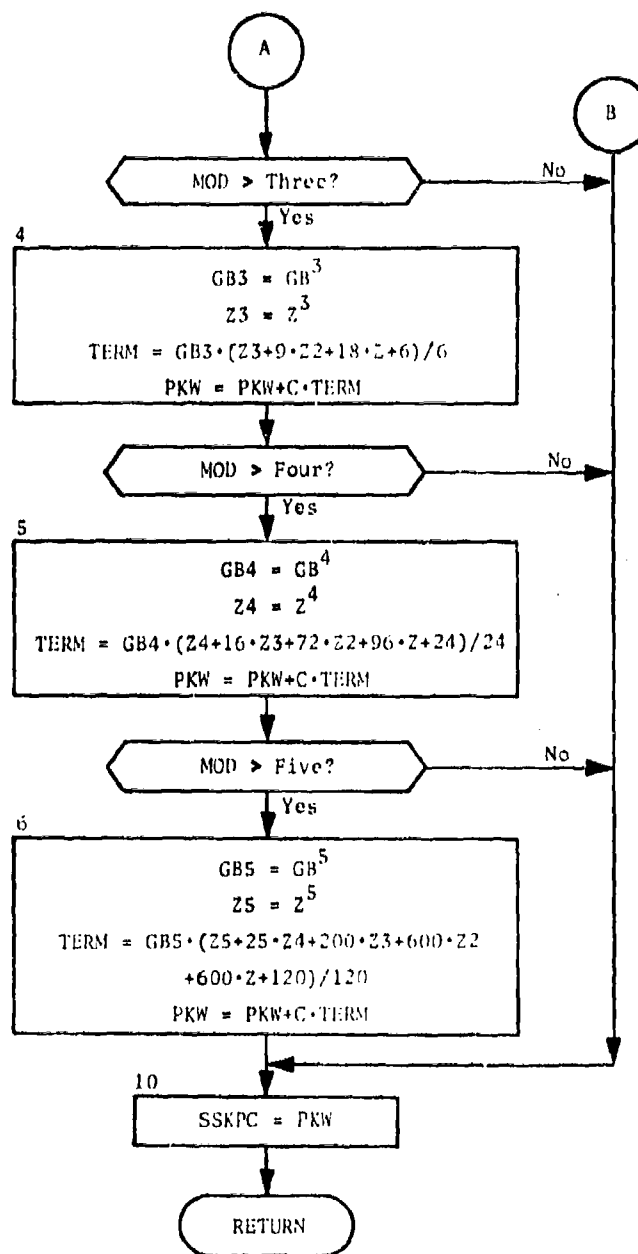


Fig. 72. (cont.)
(Sheet 2 of 2)

SUBROUTINE STORAGE

PURPOSE: To calculate and print the number of available memory locations.

ENTRY POINTS: STORAGE

FORMAL PARAMETERS: None

COMMON BLOCKS: BNKBND5

SUBROUTINES CALLED: GETLIMIT

Method

This subroutine calculates the number of words of available memory in both banks of the CDC 3800. It calls subroutine GETLIMIT, which returns the limits of available memory in both banks into common /BNKBND5/. STORAGE then computes the difference of the upper and lower limits and prints the number of words available for programmer use in both banks. In addition, the addresses of the first word and last word of the available area are written on the standard output.

In the FORTRAN language under the SCOPE PFS Operating System, all available core not utilized by the problem program and its associated system subroutines is used for input/output buffer storage. Thus, the information printed by subroutine STORAGE reflects the amount of available core after the buffers have been allocated. As a result, the most useful information returned by STORAGE is the upper limit in both banks. The true amount of available storage (i. e., ignoring the arbitrary length FORTRAN I/O buffers) is equal to the amount of core from the lowest available storage location* to the upper bound printed by STORAGE, less the amount of numbered common in the program.

For example, if STORAGE prints upper bounds of 40001 (octal) and 31645 (octal) in banks 1 and 0, respectively, and the lower bounds are 1 and 21645 (octal) in the banks, the amount of available core is 20480 words (decimal) i.e., $40001_8 - 1 + 31645_8 - 21645_8$. Any numbered common in the program must be subtracted.

Subroutine STORAGE is illustrated in figure 73.

*In bank one, the lowest available location is 1. In bank zero, the lowest available location is the first location following the resident monitor in bank zero.

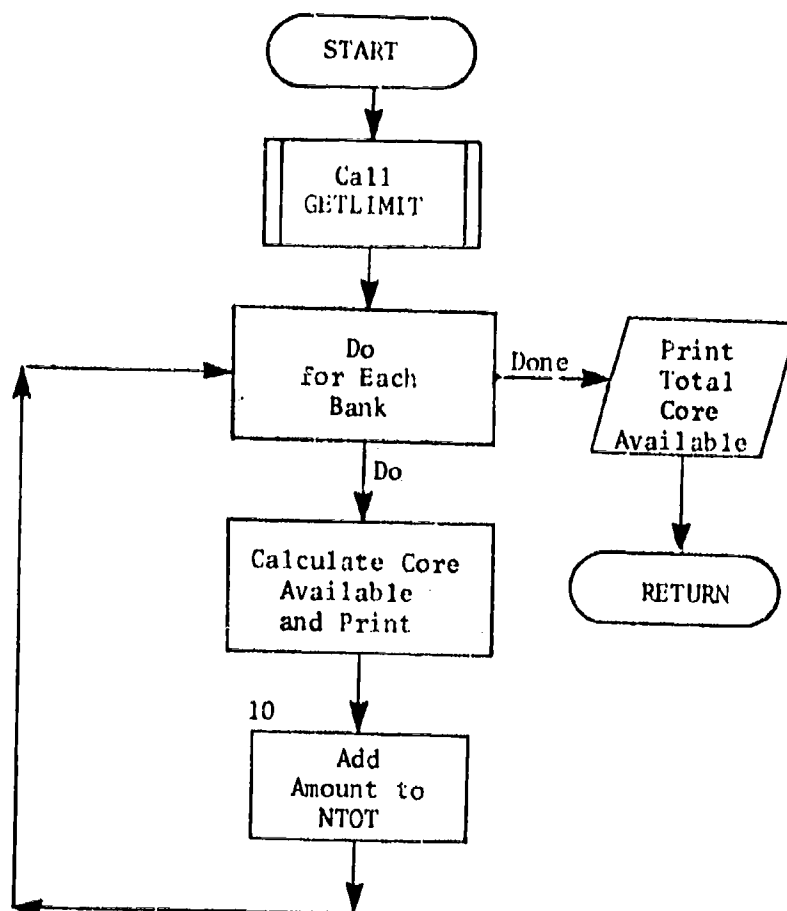


Fig. 73. Subroutine STORAGE

FUNCTION TIMEDAY

PURPOSE: To obtain the current time of day.

ENTRY POINTS: TIMEDAY

FORMAL PARAMETERS: X - A dummy parameter

COMMON BLOCKS: None

SUBROUTINE CALLED: TIME

Method

This assembly language function is used to get the current time of day in 24-hour clock format. The eight characters that are returned are used as follows: first character, blank; two characters, hour; two characters, minute; two characters, seconds; and one character, blank. The system macro TIME is used to retrieve the time of day in this format. When called from a FORTRAN program, the result is returned in floating point.

Function TIMEDAY is illustrated below in figure 74.

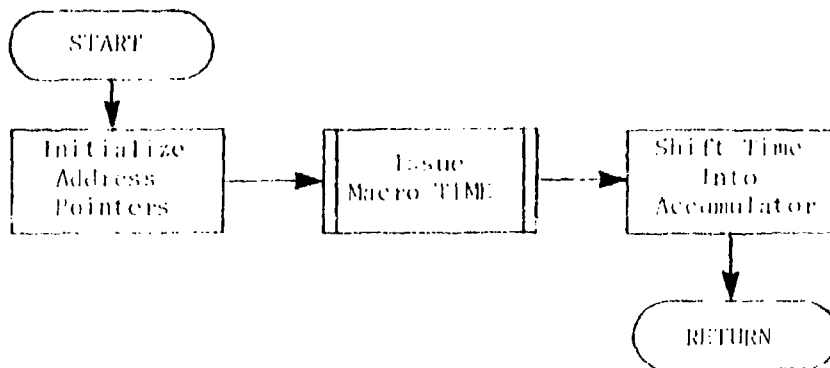


Fig. 74. Function TIMEDAY

SUBROUTINE TIMEME

PURPOSE: To record and print differential and cumulative differential time intervals.

ENTRY POINTS: TIMEME

FORMAL PARAMETERS: I - Integer variable used as described below

COMMON BLOCKS: None

SUBROUTINES CALLED: GETCLOCK

Method

The formal parameter is used as follows:

- I = -3 Reactivate clock
- I = -2 Deactivate clock
- I = -1 Initialize TIMEME
- I = 0 Print differential, cumulative differential, and elapsed times
- I < 10 Place differential and cumulative differential times in cell I.

TIMEME is used in QUICK to obtain the differential and cumulative execution times of different stages of a program. A call on TIMEME with I = -1 initializes its internal clock. Subsequent calls with I, 1:10 will record the differential time in the Ith differential time cell and will add the differential time into the cumulative differential time cell. For example, to obtain the separate running times for three stages of a program, one would call TIMEME with I = -1 at the beginning of the first stage to initialize the timing. A call with I = 1 at the end of this stage would cause the elapsed time to be recorded in cell 1 of the differential time array and to be added into cell 1 of the cumulative time array. Calls with I = 2 and I = 3 at the end of the second and third stages, respectively would similarly fill the second and third cells of the two arrays. Notice that I is used as a label for the cells, not as an iteration index.

It is possible to exclude time used for irrelevant operations by using calls with I = -2 and I = -3 at the beginning and end of each irrelevant operation to "turn off" the internal clock. For example, if one has a print subroutine which can be called at any time to produce debug prints, these print times

can be excluded from the running times by making a call with $I = -2$ at the beginning of the print subroutine and a call with $I = -3$ at the end. This causes the time elapsed during the print routine to be subtracted from the differential time of the calling stage of the program.

A call with $I = 0$ causes the differential and cumulative differential time arrays to be printed, as well as the total time, the time lost, and the sum of these two, or the total elapsed time.

Three error warnings may be issued from TIMEME:

1. ILLEGAL TIMEME CALL $I = (X)$
(if $I > 10$)
2. ERROR - STOP ATTEMPTED WITH CLOCK ALREADY STOPPED
(if two successive calls with $I = -2$ are made without an intervening call with $I = -3$)
3. ERROR - RESTART ATTEMPTED WITH CLOCK ACTIVE
(if two successive calls with $I = -3$ are made without an intervening call with $I = -2$)

After each of these messages, the subroutine returns with no further processing.

Subroutine TIMEME is illustrated in figure 75.

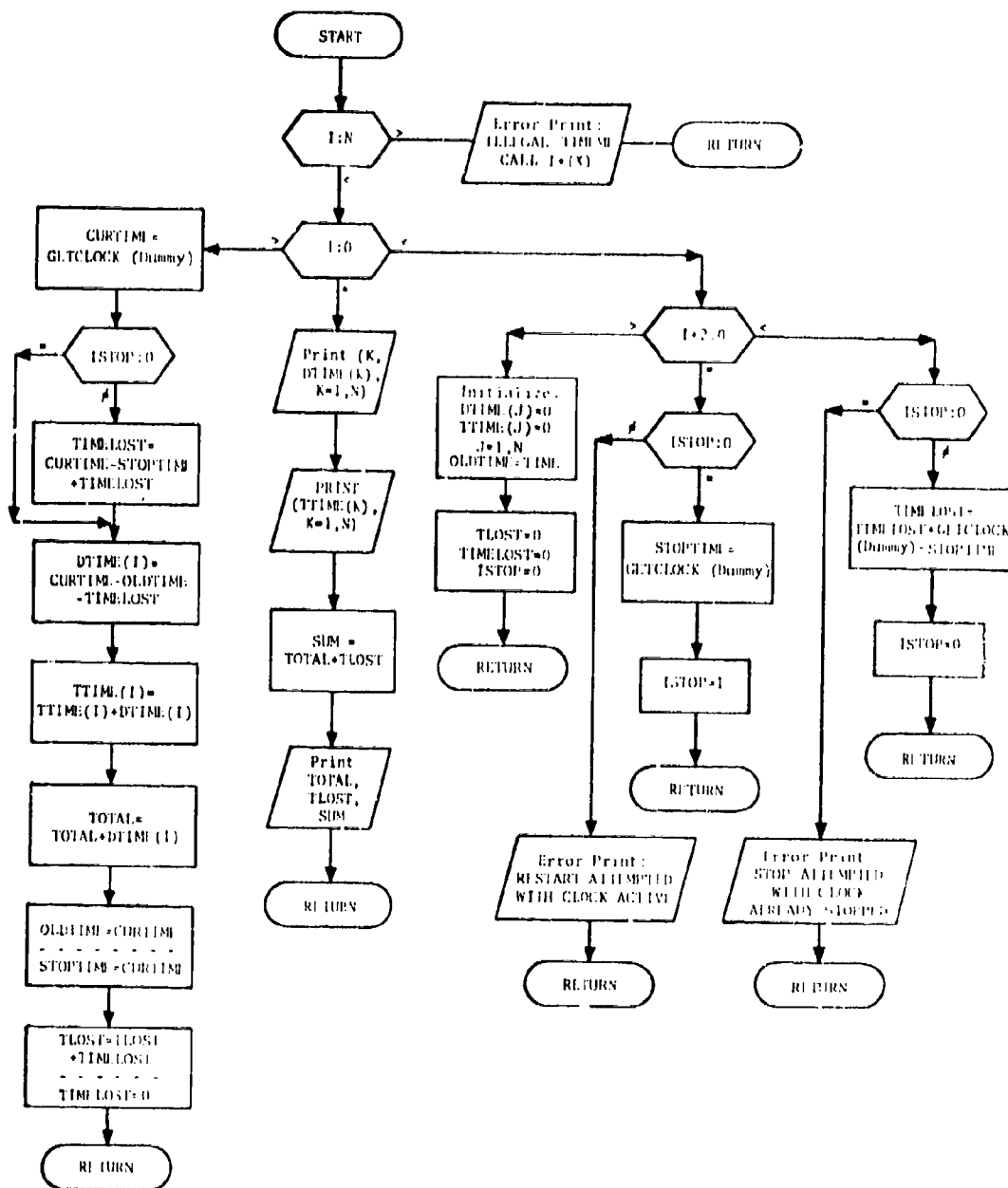


Fig. 75. Subroutine TIMMEM

SUBROUTINE WRITEDIR

PURPOSE: To write a data base directory as contained in common block /DIRECTORY/ onto a specified file.

ENTRY POINTS: WRITEDIR, WRITEDR

FORMAL PARAMETERS: ITAPX - Logical unit number of the file on which the directory is to be written

COMMON BLOCKS: DIRECTORY, ITP, TWORD

SUBROUTINES CALLED: SETWRITE*, WRARRAY*, WRWORD*

Method

This subroutine writes the contents of common block /DIRECTORY/ on unit ITAPX. Subroutine WRITEDIR is illustrated in figure 76. As indicated, the filehandler subroutine WRARRAY is called to write the contents of the arrays described below onto the designated unit. Having transferred these data, the end of the directory is signalled by writing the word ENDIRECT on the output file using subroutine WRWORD.

1. ATTNAME: This array contains the name of each defined attribute.
2. IFORMAT: This array contains the conversion format (e.g., 18) to convert the value of the attribute to output (printer) format.
3. ICODE: The array code numbers (between one and seven) which specify the type value (e.g., floating point number) for each attribute and the method of checking (range, list, or none).
4. DEFAULT: This array contains values which are to be associated with any attribute when it is not defined for an item.
5. N1, N2: These arrays contain either the minimum or maximum values for the attributes in the case of range checking, or the beginning and ending indices to a list of all possible values for the attributes that specify list checking. These arrays are unused when no checking is called for.
6. LISTCHK: This logical array is set to "true" for list checking and "false" for range checking or when no checks are required.

*See filehandler subroutines.

7. LGLOB: This logical array is set to "true" when a global definition is in force and set to "false" otherwise.
8. LISTVALS: This array contains the acceptable values for an attribute to be used in list checking. In this mode of checking, the attribute value is restricted to a specified list. For example, acceptable values of the attribute SIDE might be limited to RED and BLUE.

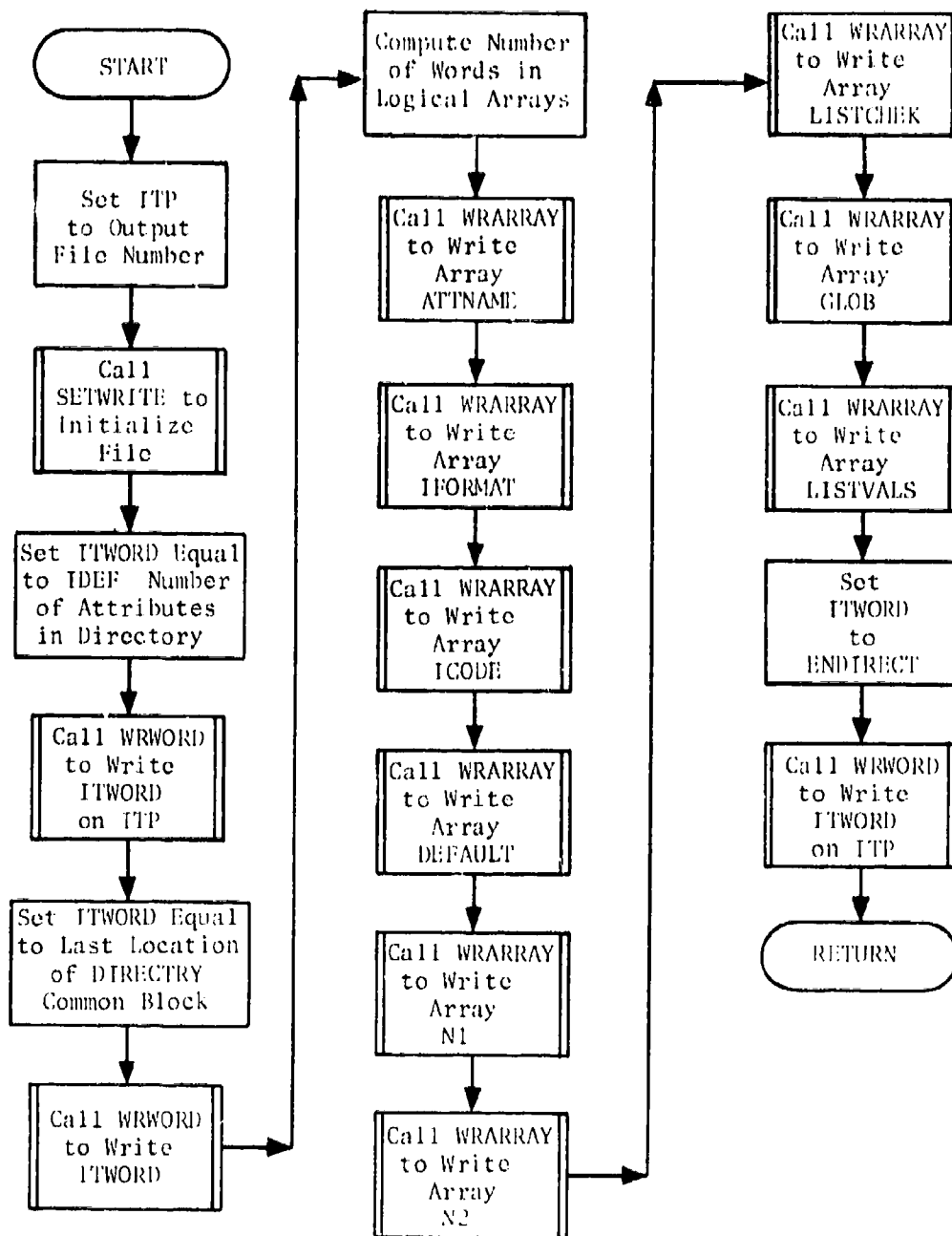


Fig. 76. Subroutine WRITEDIR

CHAPTER 5 PROGRAM QUIKBASE

PURPOSE

The main purpose of program QUIKBASE is to create a game base file, composed of a number of items defined by a set of attribute-value pairs, which defines the data base that will be used by succeeding programs in the QUICK system. In addition, program QUIKBASE has the capability to create or to update a data library file, as well as the capability to print the game base file in a format which is meaningful to the user.

INPUT

The input to program QUIKBASE can be separated into four major categories: program option cards, a data library file, update command cards, and item update files. These inputs are merged to create the game data base file.

Program Option Cards

These user-input parameter cards specify the mode of operation of program QUIKBASE. The four major options are: creation of a data library file from card images (SETID option); updating of a data library file to create a new library file (UPDATE option); generation of a game base file from the data library file (QUIKDBG option); and printing of the game base file (PRINTDB option). These options may be exercised singly or in groups to perform the desired operations. Each option has a series of subparameters which control program operation (e.g., printing) during the execution of each option.

Data Library File

This file, named DATADB, contains a series of card images with the addition of card identifiers consisting of a set number, a line number, and an update identifier (e.g., the date). The card images are divided into sets, each with a unique number. Within each set, each card is assigned a consecutive line number. Thus, each card is uniquely defined

by its set and line number. The update identifier is an eight-character word which aids the user in determining the run which introduced the card into the data library file.

In some applications of program QUIKBASE, there is no existing data library file. In this case, the SETID option may be exercised to create this file from a card deck or a tape containing BCD card images. The input file is then called the DATAFIL.

The card images on the data library file (DATADB) are data base creation cards. This set of cards is constructed as follows:

1. Directory Card Images. The set of directory cards is in the format of eight words of 10 columns each with all quantities, including integers, left-justified. Two commands appearing in word one are recognized: ADD and ENDIRECT. The ADD command is used to add a new attribute to the directory and is followed by cards containing information about the QUICK data base attributes. The ENDIRECT card terminates the deck of directory cards.
2. Data Base Card Images. The ENDIRECT card is followed by the deck of item cards. Four commands are recognized: DEFINE, ITEM, UNDEFINE, and ENDINPUT. The DEFINE command is used to produce global definitions; i.e., attribute values which will remain in force until removed by an UNDEFINE command. Words 2, 4, and 6 contain attribute names, while words 3, 5, and 7 contain their associated values. A blank word terminates the sequence. The ITEM command is used to provide values of attributes for the current item only. Words 2, 4, and 6 of the card contain attribute names, and words 3, 5, and 7 contain their associated values. Subsequent cards contain attribute names in words 1, 3, 5, and 7 and values in words 2, 4, 6, and 8. The sequence is terminated only upon detection of a command in the first word of a card. UNDEFINE removes global definitions. The names of the attributes to be undefined occur in the remaining words of the card. ENDINPUT terminates the deck of items.

Update Command Cards

These user-input parameter cards specify the mode of operation in the UPDATE option. They specify the placement of new cards in the existing data sets, as well as the input medium for update information. The basic UPDATE commands are REPLACE, DELETE, and ADDAFTER. The REPLACE command requests the update card image to replace the existing card

image on the data library file. The DELETE command removes a set of card images from the data library file. The ADDAFTER command inserts a number of update card images onto the data library file. These major commands have several subcommands which control the printing of information and the input mode for the update card images.

Item Update Files

These files (there may be up to four files used in any single run) contain update card images to be inserted into the data library file. The format of these files is either buffered data (56-word physical records) prepared by program DELDESIG or CARD images on tape prepared by program DATABACK. These programs support, but are not a part of, the QUICK system. Therefore, they are not discussed further in this manual.

OUTPUT

There are two major outputs from program QUIKBASE: an updated data library file (updated DATADB), and a game base file (QUIKDB).

The updated DATADB file is in the same format as the original DATADB file. It consists of a series of card images with two words appended to each image (see table 4). The two words are the set number and line number (both contained in one word), and the update identifier.

The QUIKDB file is the data base that will be used by the succeeding QUICK programs. It consists of a directory, which defines the structure of the attribute-value pairs, and a series of items defined by a set of attribute-value pairs. (Table 5 shows the format of the data blocks on this file.) This file contains most of the information on the weapon forces and target systems that will be used in the game. The conversion of the data library cards to the data base items is discussed in the next section.

Table 4. Data Block Format for DATADB File

<u>WORD</u>	<u>DESCRIPTION</u>
1	Columns 1-8 of card image.
2	Columns 11-18 of card image.
3	Columns 21-28 of card image.
4	Columns 31-38 of card image.
5	Columns 41-48 of card image.
6	Columns 51-58 of card image
7	Columns 61-68 of card image.
8	Columns 71-78 of card image.
9	Set and line number. First four characters are set number; last four characters are line number.
10	UPDATE identifier.

CONCEPT OF OPERATION

The main processing flow of program QUIKBASE is best viewed as consisting of four separate steps or options. Not all steps need be accomplished to create the game base file. The selection of the options to be exercised in each run depends on the form of the input and the desired output. Since a complete run of QUIKBASE could include all four options, this section describes the operation as if all four were to be exercised.

Creation of a Data Library File (SETID) Option.

This option reads an input file of card images and creates a data library file. The input set of card images is divided into subsets, hereinafter called sets. The division of the input file into sets is not merely for addressing convenience. The QUICK data base concept allows for "global"

Table 5. Data Base File (Logical Record Format)

<u>BLOCK TYPE</u>	<u>ARRAY</u>	<u>LENGTH</u>	<u>DESCRIPTION</u>
Directory	IDEF	1	Number of attributes
	LASTLIST	1	Number of entries in value list
	ATTNAME	IDEF	Attribute names
	IFORMAT	IDEF	Hollerith format codes
	ICODE	IDEF	Error checking code
	DEFAULT	IDEF	Attribute default value
	N1	IDEF	Minimum allowable attribute value
	N2	IDEF	Maximum allowable attribute value
	LISTCHEK	IDEF	Logical array to specify list checking
	LGLOB	IDEF	Logical array to specify global definitions
	LISTVALS	LASTLIST	List of values to be checked
Item	NI	1	Number of attributes defined locally for this item
	INITEM	2*NI	Array containing attribute index (from directory) in odd elements and attribute value in even elements
Define	NI	1	= -1 as DEFINE block indicator
	L	1	Attribute index from directory
	VALUE	1	Attribute value
Undefine	NI	1	= -2 as UNDEFINE block indicator
	L	1	Attribute index from directory
	VALUE	1	New attribute value
Terminator	NI	1	= ENDDATA as terminator block indicator

definitions of attribute values. That is, an attribute can be defined (by a DEFINE command) to retain a value for a number of consecutive items. This capability allows compression of the DATADB file without loss of information, since the value for globally defined attributes need not be repeated for every item. The global definition remains in effect until either an UNDEFINE command is encountered or the end of a set is reached. Thus, division of the input file into sets determines the maximum range of global definitions. A global definition cannot exceed one set unless it is recreated by a new DEFINE command.

In addition to division of the input file into sets, this option appends a line number and an update identifier to each card image. The line numbers run consecutively within each set, beginning at one at the start of each set. Each set may contain no more than 9,999 card images. The update identifier is an eight-character word which identifies the run which inserted the card image. Table 4 shows the format of the data blocks in the data library file, DATADB. There is one data block for each card image.

The user options concerning set division are discussed in the description of subroutine SETID.

Updating a Data Library File (UPDATE Option)

This option replaces, deletes, or inserts information on the data library file, DATADB. The output is an updated DATADB file in the same format as the original DATADB file. The update commands must be in the same order as the information on the DATADB file. Therefore, the process used in this option is very simple.

The program reads the DATADB file one image at a time. If the set and line number of the input image does not match the set and line number of the next update command, the current image is written on the updated DATADB file. The next card image is then read from the DATADB file. If the set and line number does match the set and line number on the next update command, the program proceeds to revise the card image.

If the command is REPLACE, the update image replaces the input image on the updated DATADB file. If the command is DELETE, no information is written on the DATADB file. If the command is ADDAFTER, information is inserted after the current card image on the updated DATADB file.

Data Base Generation (QUIKDBG Option)

This option produces the QUIKDB file. It may be run concurrently with the UPDATE option, since a data base is generated on a card-by-card basis.

The QUIKDB file consists of a number of data blocks as defined in table 5. The first block on the file is the DIRECTORY block. There is only one DIRECTORY block on the file. This block defines the attribute names and characteristics. It provides a summary of information which will be used to determine attribute values. In particular, the DIRECTORY defines the position of each attribute in a value array (VALUE). In addition, the mode of error checking for each attribute is determined by information in the DIRECTORY.

The DIRECTORY block is followed by a number of ITEM, DEFINE, and UNDEFINE blocks. Each separate data base item generates one ITEM block. Each global definition generates one DEFINE block. Each removal of a global definition generates an UNDEFINE block. (These blocks are generated by the data base commands of the same names.) The DATADB file is terminated by a terminator block. The command ENDDATA generates this block.

The process used to generate these blocks is straightforward. The directory commands ADD and ENDIRECT to generate the DIRECTORY block. The description of subroutine NEWDIR explains the use of these commands. The ITEM command generates one ITEM block. Each DEFINE or UNDEFINE command generates one block (DEFINE or UNDEFINE, respectively) for each attribute listed. The description of subroutine NEWBASE explains these commands.

Printing of a Data Base File (PRINTDB) Option

The data base file QUIKDB may be printed in a readable format as it is created in the UPDATE option, or separately through the PRINTDB option. In either case, either subroutine PRNTBASE or subroutine PRNTITA is used to print the attribute-value pairs. The desired level of detail determines which subroutine is used. The description of these utility subroutines explains the printing process.

IDENTIFICATION OF SUBROUTINE FUNCTIONS

Program QUIKBASE is the main control program. This subprogram reads the program option cards and calls the appropriate subroutines. Figure 77 is a flowchart of program QUIKBASE.

SETID Option (Creation of Data Library File)

All the functions of this option are performed by subroutine SETID.

UPDATE Option (Updating of Data Library File)

Subroutine FASTSET is the main controller for this option. It calls subroutines INITFAST and NEWDATA to initialize the variables and arrays used in this option. Subroutine INPRCTL reads and interprets print requests. The update commands are read by subroutine CARDCK. They are interpreted by subroutine NEWDATA (entry NEWCARDS), and their correctness is checked by subroutine CARDCK. If some of the input information is contained on files from program DELDESIG, subroutine BUFEIT is used to read this information. Subroutines MOVEIT, IPRINT, ADDSET, OUT, COUNTDS, COPYDB, PRICONT, and ILOOK are used to perform minor utility functions for this option.

QUIKDBG Option (Data Base File Creation)

If the data base is created concurrently with a data library file update, subroutine MAKEIT controls the data base generation. Otherwise, subroutine MAKEBAS is utilized. The data base directory commands are interpreted by subroutine NEWDIR. (The DIRECTORY block is written by utility subroutine WRITEDIR.) The data base item commands are interpreted by subroutine NEWBASE. (The data base file blocks for these commands are written by utility subroutines OUTITEM and OUTWORDS.) The terminator block is output by utility subroutine ENDDATA.

PRINTDB Option (Printing of a Data base File)

If the only option exercised is the PRINTDB option, subroutine PRONLY controls the mode of operation. Otherwise, subroutines FASTSET or MAKEBAS will control printing. The actual printing of the data base file is performed by utility subroutines PRNTBASE, PRNTITA, and PRITEM.

COMMON BLOCK DEFINITION

External Common Blocks

Program QUIKBASE references the following utility routine common blocks, which are described in appendix A of this manual: /DIRECTRY/, /ERRORM/, /FILABEL/, /IFTPRNT/, /ITP/, /MPRTOPT/, /MYIDENT/, /MYLABEL/, /NOPRINT/, /PRTOPT/, /TODAY/, and /TWORD/.

Internal Common Blocks

Table 6 lists the common blocks used within program QUIKBASE and identifies the associated arrays.

Table 6. Program QUIKBASE Common Blocks
(Sheet 1 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
1	NSAVE	Array to store defined attributes and values during default run
	NS	Array for storage of attribute names from directory
HIST	NOPSUSD	An array in which the option specifiers (e.g., UPDATE and SETID) are stored in the order they are executed
	NUSED	The number of options called for during one QUIKBASE run
ICKTST	ICKTST	Set to 0 if attribute values are to be checked against the data base directory; set to 1 if input data card indicates checking not desired
ICONTROL	ICOM	1 if ITEM; 2 if DEFINE; 3 if UNDEFINE; 4 if ENDINPUT; 5 if none of the above
IDESIGS	IDESIGS	Array where alphabetic part of target designators is stored
	DESIGNO	Array where targets are counted by region
IENDSET	IENDSET	True if all DATA0B has been read; false if not

Table 6. (cont.)
(Sheet 2 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
ITSTUFF	ITSTUFF	Array where buffered data are read and stored
IWSIDE	IWSIDE	Hollerith print constants RED and BLUE
JDESTEST	JDESTEST	Position of DESIG attribute in ATTNAM array
KKSET	KKSET	Set to 1 or 2 to control storage of target designator codes
LOGFLAG	LAST	TRUE if LAST update item has been read
	ICNT	TRUE if there is an unprocessed update card
	TAPEIN	TRUE if BCD tape data to be added
	IADD	TRUE if last control card was ADDAFTER
	DEL	TRUE if last control card was DELETE
	REPL	TRUE if last control card was REPLACE
	ADDIT	TRUE if card in buffer is to be added to base
	IERROR	Not used
MYGOODS	NLINE	Line number of current DATADB line number
	ISF	Set number from update tape item; i.e., set to be modified

Table 6. (cont.)
(Sheet 3 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
MYGOODS (cont.)		
	JSET	Set number of current DATADB set
	NEWDATE	Run identification (if given) from update tape item
MYINPUT	INSTUFF	Array where auxiliary data are read if whole items or sets are being added
MYOUT	OUTSTUFF	Array where DATADB item is read and used
MYPRINT	ISTAR	Blank if old data item; * if new data item
	NDEFINE	TRUE (or FALSE) if last item was a define card
	NUNDEF	TRUE (or FALSE) if last item was an undefine card
MTAPES	INUIT	Tape number of auxiliary data; read in on edit command card
	JTIN	DATADB tape number; set from INITAPL
	LOUT	DATADBUP tape number; set from KOUTL (LIN 6)
	KTAPOUT	QUICKDE tape number (data set to 7)
	KINCARDS	Temporary storage tape for edit command cards (data set to 10)

Table 6 . (cont.)
(Sheet 4 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
NEWSET	NEWSET	Set number to be assigned to new set, if present
NODESTCS	NODESTCS	Total count by side of target designator codes
NOERRORS	NOERRORS	Error count of edit command card errors
	NAMEOF	2nd, 3rd, and 4th fields on LAST card of edit deck; usually name and phone number of person who submitted jobs
	NRPHONE	
	IWANTBU	
NOTEST	NOTEST	Switch to control data checking; 0 to check, nonzero if no checking required
OPTIONS	NCON	Array into which option card is read; NCON(6) passed on as date evident
	INTAPE	Input (primary) tape for the run
	NOUT1	Output (primary) tape for the run
	NOUT2	Second output tape (if any) for the run
	NPR	1 if print selected; 2 if not
	NUN	Beginning line number of SETID set
	NOPSET	Option switch for SETID; 1=default, 2=side, 3=class, 4=side or class, 5>manual

Table 6 . (cont.)
(Sheet 5 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
OPTIONS (cont.)		
	ISETSIZ	Largest allowable line number in a set made by default option of SETID
SETIDD	ID	Array where set number to be printed is stored
	INDEX	9999 if all sets are to be printed; 0 if none are to be printed; count of number of sets to be printed if list read in
	JINDEX	0 if no sets to be printed; equal to INDEX if list read in; equal to number of sets updated if IAUTO=1
	IAUTO	1 if updated sets are to be printed; 0 if not

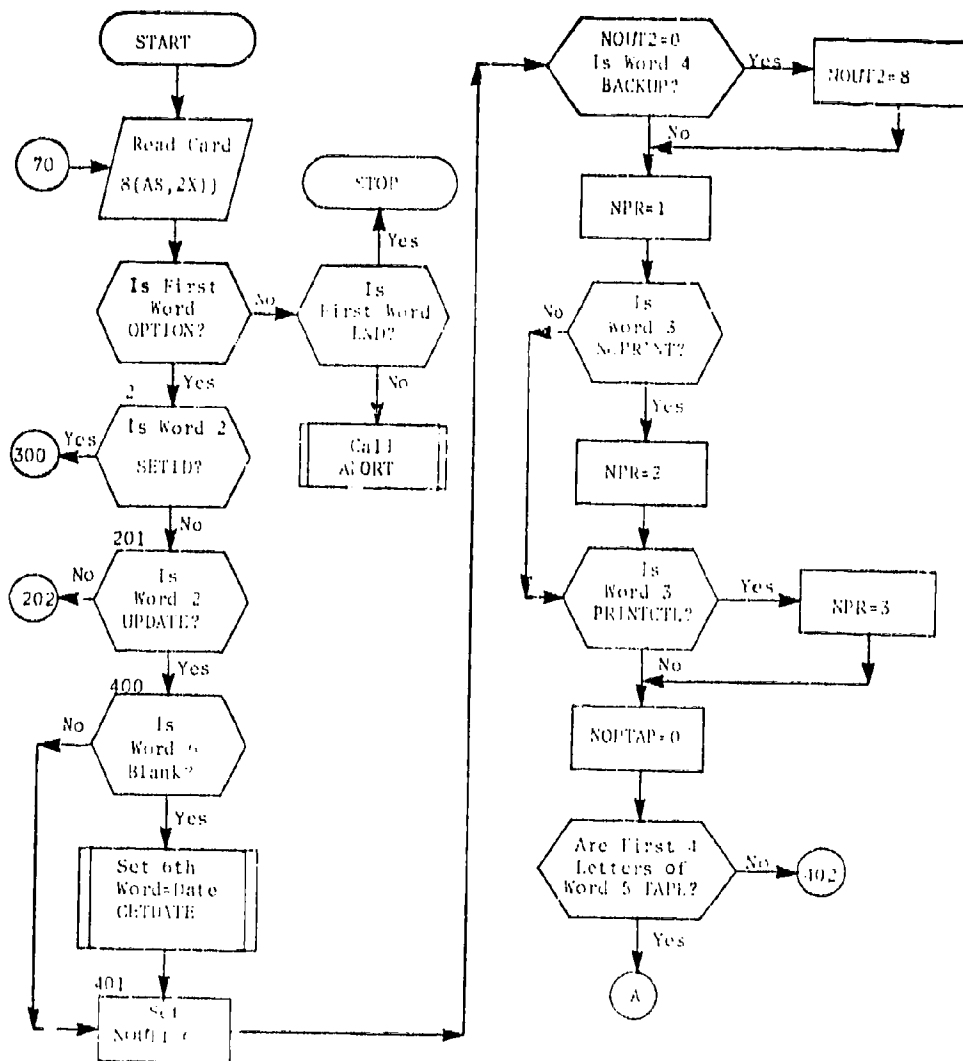


FIG. 7. Program ALERASH
(Sheet 1 of 1)

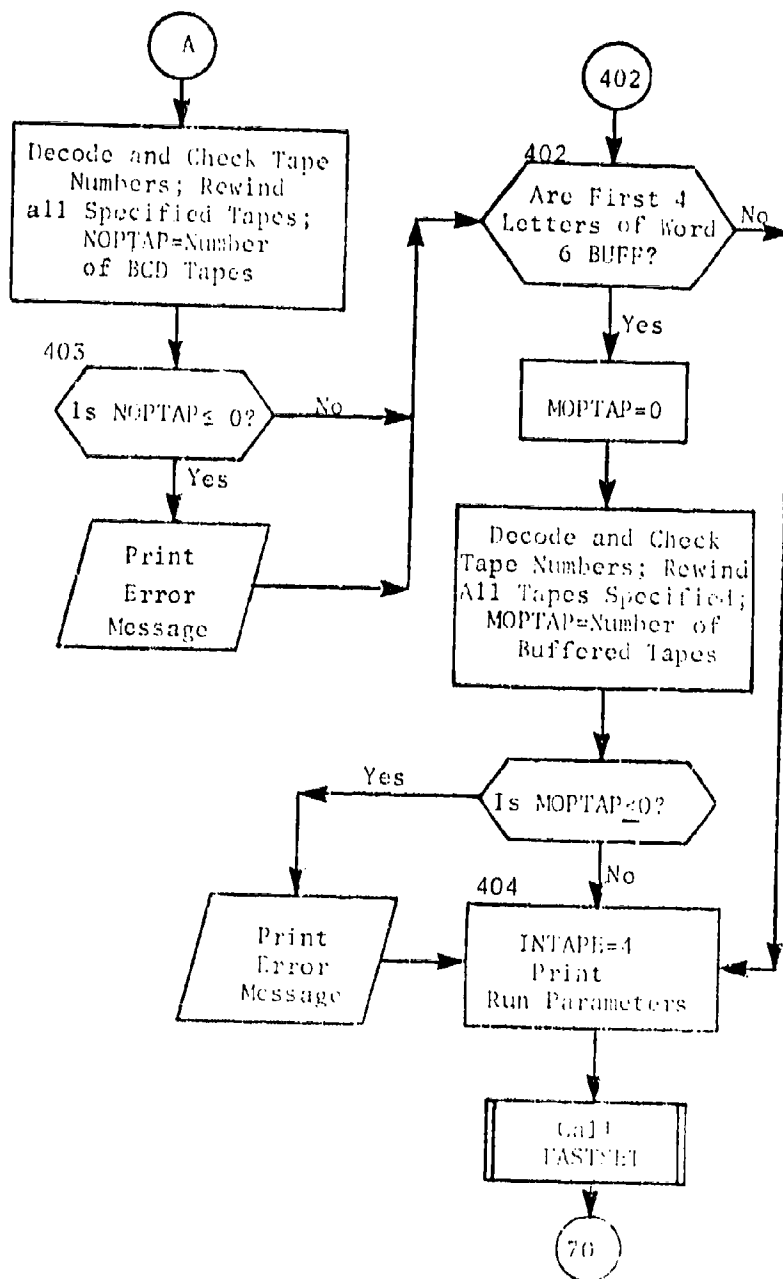


Fig. 77. (cont.)
(Sheet 2 of 4)

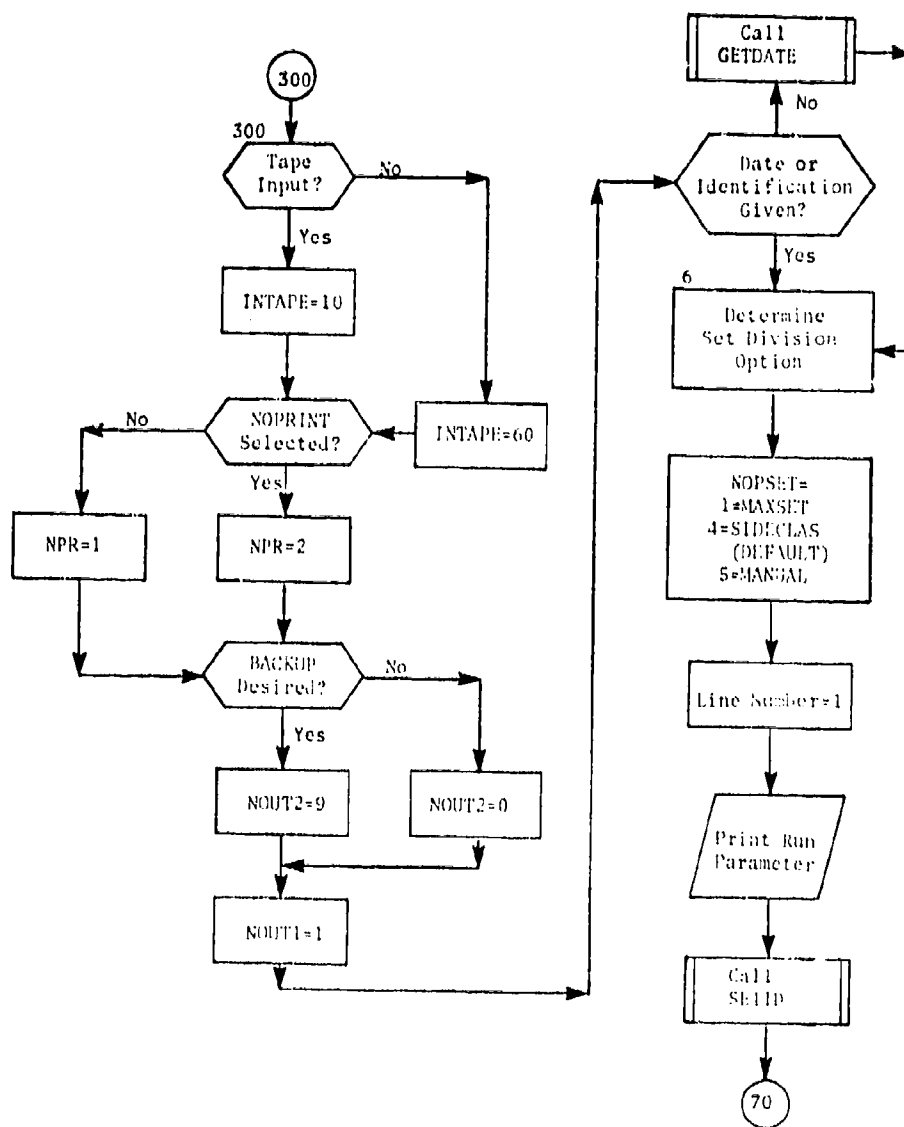


Fig. 77. (cont.)
(Sheet 3 of 4)

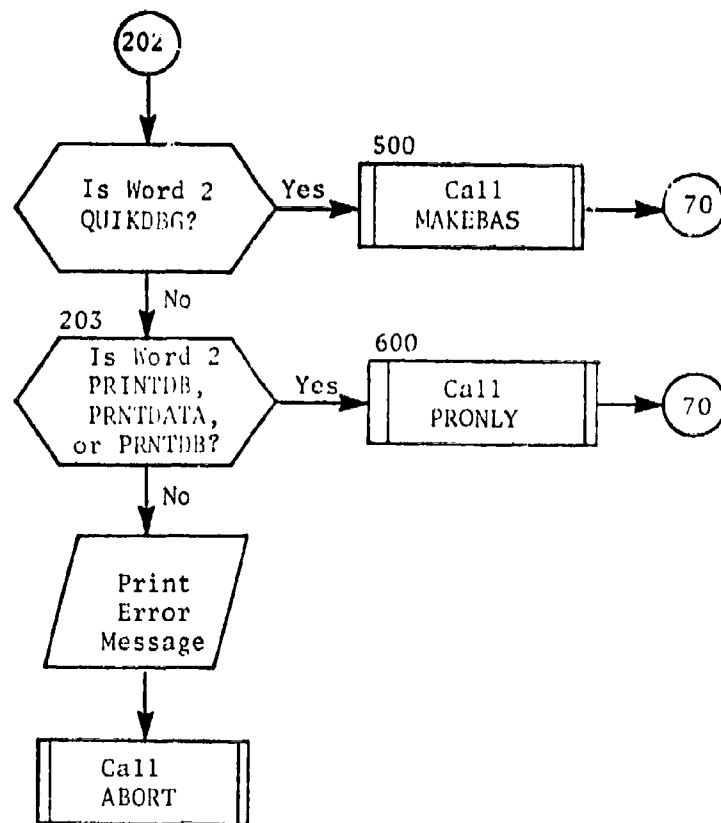


Fig. 77. (cont.)
(Sheet 4 of 4)

SUBROUTINE ADDSET

PURPOSE: To add a set number to the list of sets to be printed.

ENTRY POINTS: ADDSET

FORMAL PARAMETERS: I - The set number

COMMON BLOCKS: SETIDD

SUBROUTINES CALLED: ILOOK

CALLED BY: CARDCK

Method

Subroutine ADDSET adds a set number (I) to the array ID and increases the pointer (JNDEX) under which the set number is stored. An error message is printed if more than 50 sets are requested. Subroutine ADDSET is illustrated in figure 78.

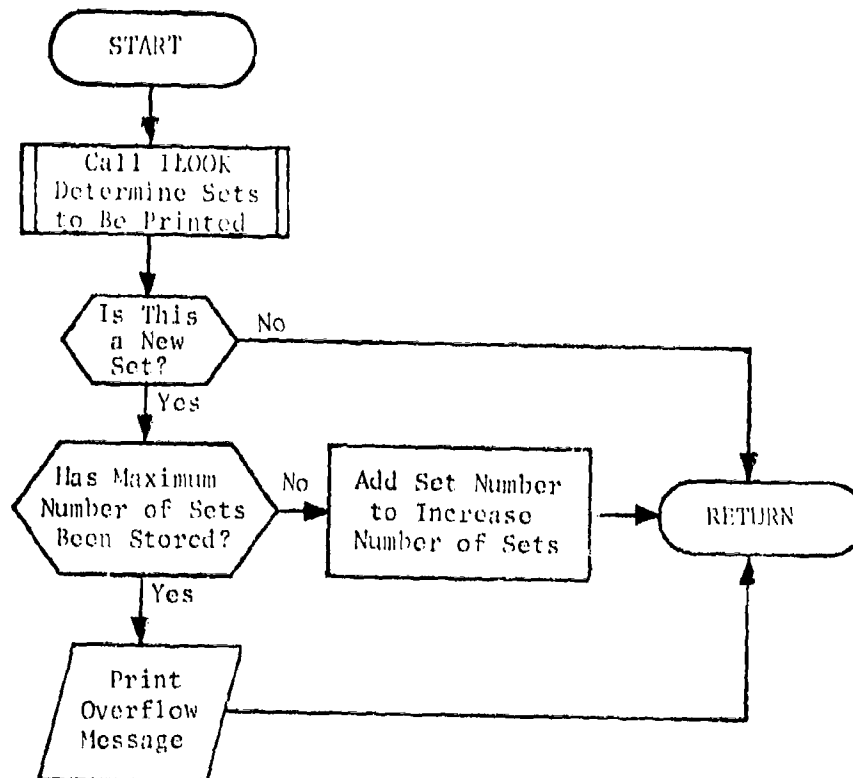


Fig. 78. Subroutine ADDSET

SUBROUTINE BUFFIT

PURPOSE: To read in buffered data and to return one logical record (eight words) to calling program.

ENTRY POINTS: BUFFIT

FORMAL PARAMETERS: IDONE - Flag to show end of tape data;
0 if not done
1 if done

COMMON BLOCKS: ITSTUFF, MYTAPES, MYGOODS, MYINPUT, NEWSET

SUBROUTINES CALLED: None

CALLED BY: NEWCARDS

Method

Subroutine BUFFIT reads a physical record of 56 words (seven eight-word items), compares the set number of buffered data with the desired set number and, if they are the same, returns a data record to the calling program through common block /MYINPUT/. If a record is being returned, the formal parameter IDONE is set to 0. When the data for the set are exhausted, either because the end of set or the end of the data is reached, IDONE is returned as a 1. If a parity error is encountered on the input tape, an error message is printed and the job is terminated. Subroutine BUFFIT is illustrated in figure 79.

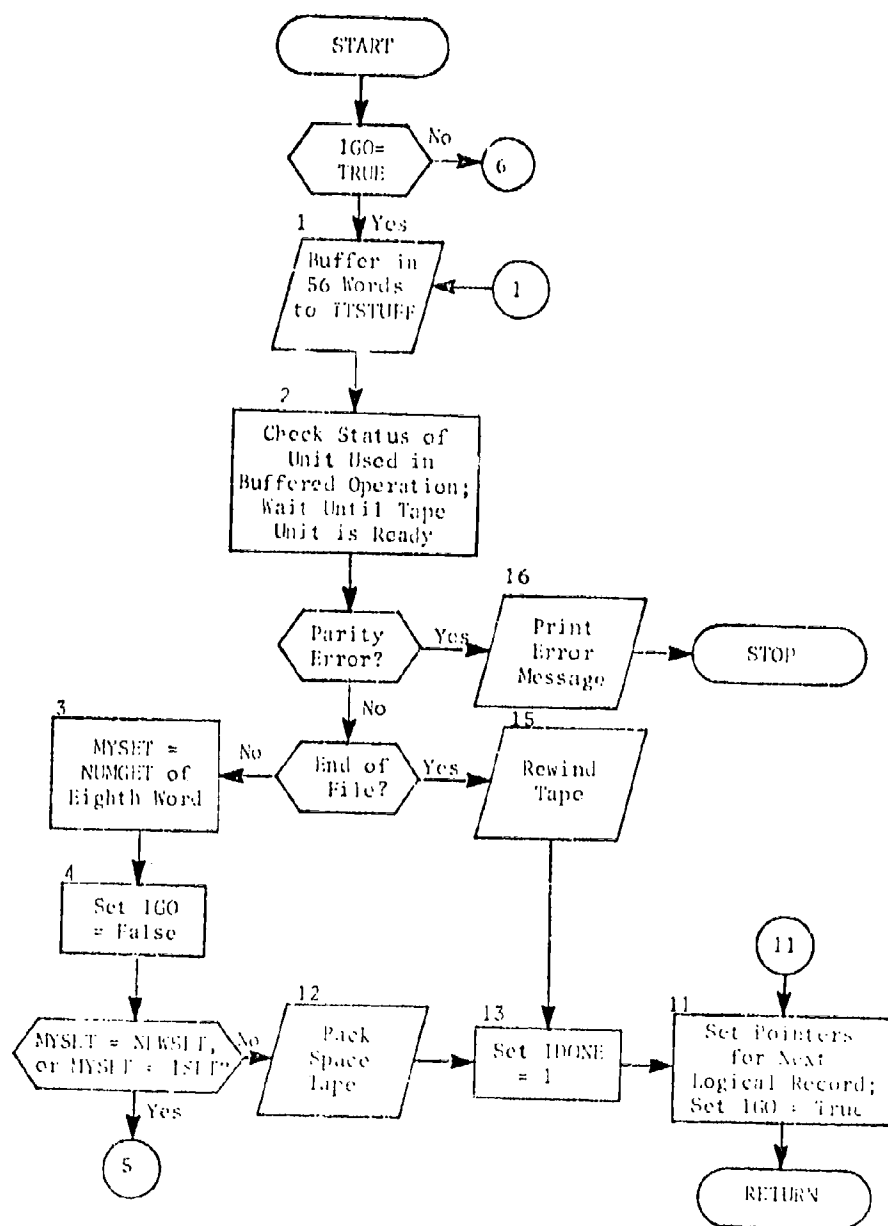


Fig. 79. Subroutine BUFEFF
(Sheet 1 of 2)

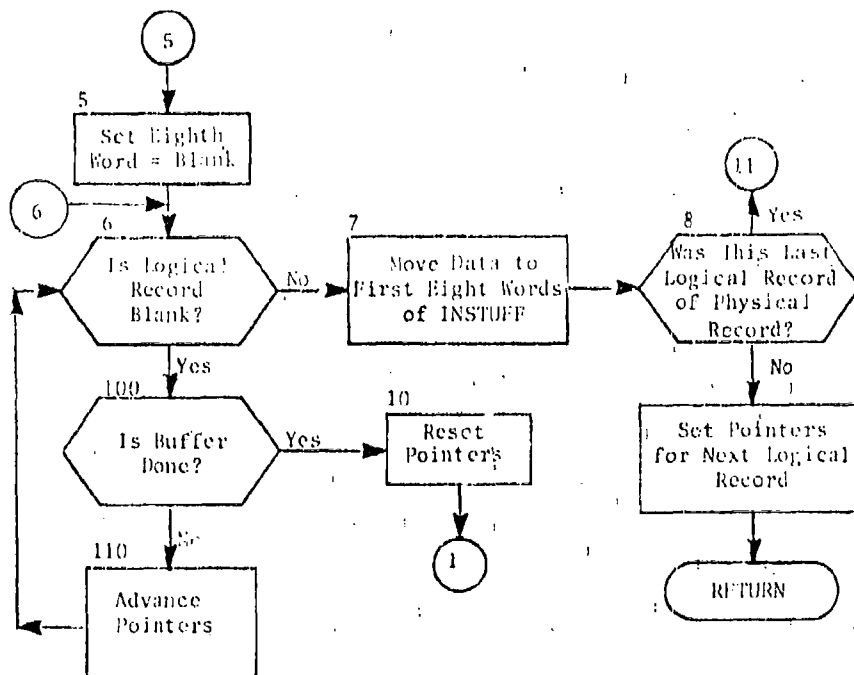


Fig. 79. (cont.)
(Sheet 2 of 2)

SUBROUTINE CARDCK

PURPOSE: To read in and store on a scratch file all of the user-input editing commands and data base updates to be used to modify a DATADB tape.

ENTRY POINTS: CARDCK

FORMAL PARAMETERS: None

COMMON BLOCKS: OPTIONS, MYTAPES, NOERRORS, ITT

SUBROUTINES CALLED: ADDSET, FILEHNR, NUMGET

CALLED BY: FASTSET

Method

Subroutine CARDCK reads in a card, determines if it is an editing command (e.g., ADDAFTER, DELETE, REPLACE), and takes appropriate action. Non-command cards are assumed to be data base items and are written directly to tape; command items are examined for proper sequencing of set and line numbers and for legal commands. All cards are printed. Errors are counted and error messages are printed if illegal cards are encountered. Subroutine CARDCK is illustrated in figure 80.

SUBROUTINE COPYDB

PURPOSE: To make a copy of the updated DATADB tape.
ENTRY POINTS: COPYDB
FORMAL PARAMETERS: None
COMMON BLOCKS: ITP, MYIDENT, OPTIONS
SUBROUTINES CALLED: FILEINR
CALLED BY: FASTSET

Method

Subroutine COPYDB copies the updated DATADB tape generated by a FASTSET update run from NOUT1 to NOUT2. Copying is terminated by the word ENDINPUT. Subroutine COPYDB is illustrated in figure 81.

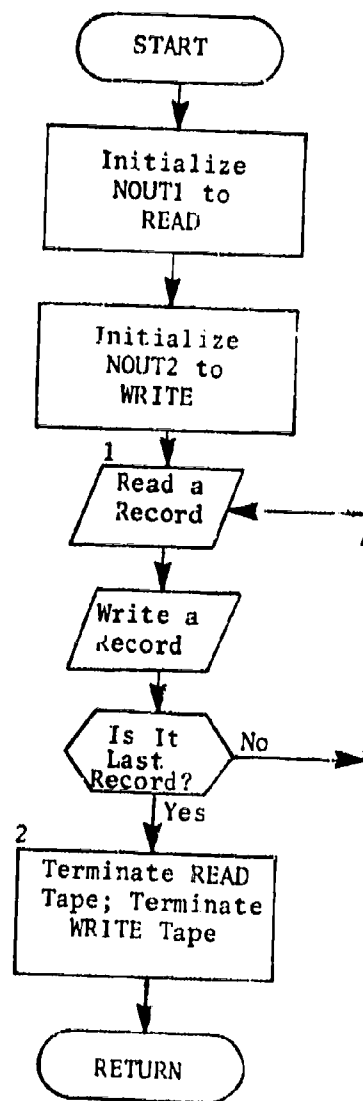


Fig. 81. Subroutine COPYDR

SUBROUTINE COUNTDS

PURPOSE: To tally the number of targets by region and target type.

ENTRY POINTS: COUNTDS

FORMAL PARAMETERS: MYDESIG - The target designator code in A8 format

COMMON BLOCKS: KKSET, IDESIGS, NODESIGS

SUBROUTINES CALLED: None

CALLED BY: NEWBASE

Method

Subroutine COUNTDS decodes the five alphameric characters (two letters and three digits) transmitted to it via MYDESIG into two letters and three numbers. The integer portion of the target designator code MYDESIG (attribute DESIG in the data base) is used to determine the region IREG to which a target class is assigned for summary purposes. Three regions are considered to be defined as 1-499, 500-799, and 800-999.

The target types, indicated by the alpha portion of DESIG, are stored in array IDESIGS by side as they are encountered and tallied by type and region in DESIGN0, a two-dimensional array. Subroutine COUNTDS is illustrated in figure 82.

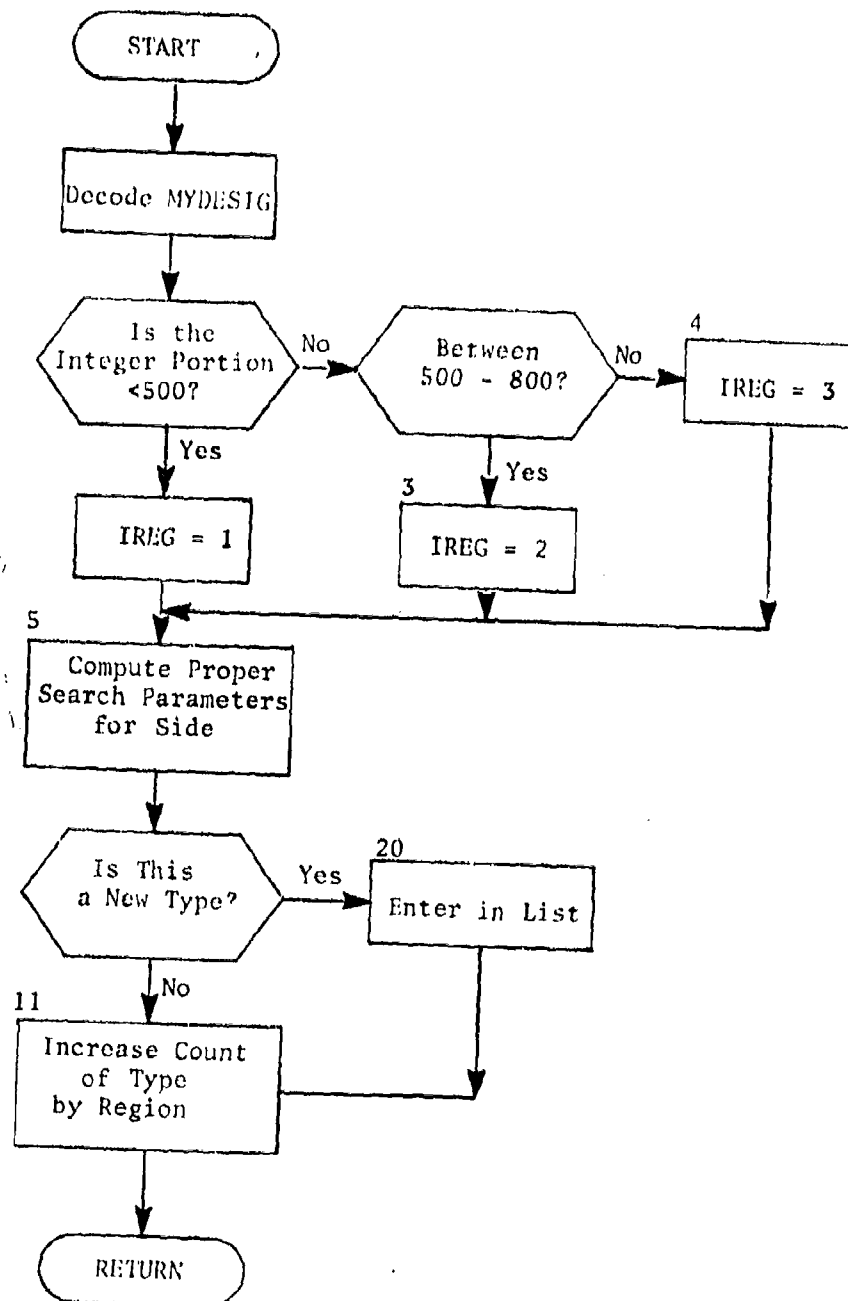


Fig. 82. Subroutine COUNTDS

SUBROUTINE FASTSET

PURPOSE: FASTSET is the main control and monitoring routine for an update run.

ENTRY POINTS: FASTSET

FORMAL PARAMETERS: None

COMMON BLOCKS: ERRORM, ITP, MYIDENT, MYTAPES, NOERRORS, NOPRINT, OPTIONS

SUBROUTINES CALLED: CARDCK, COPYDB, FILEINR, INITFAST, INPRCL, MAKEIT, NEWDATA, PAGESKP, PRTCONT

CALLED BY: QUIKBASE

Method

Subroutine FASTSET calls INITFAST to initialize all arrays. It assigns file names and initializes the filehandler and all pertinent tapes to read and write. If manual print control was requested on the option card, subroutine INPRCL is called. Subroutine CARDCK is called to read in and check all the update data. If that subroutine has found errors in the input deck, FASTSET prints the error messages and aborts the run. If the update data have no discernible errors, subroutine MAKEIT is called to perform the major functions of the program. Control is returned to FASTSET which again looks for and prints, if present, any error messages. The subroutine to print the target-region summary (PRTCONT) is called and, if requested, a second copy of the updated data base is created by subroutine COPYDB. Subroutine FASTSET is illustrated in figure 83.

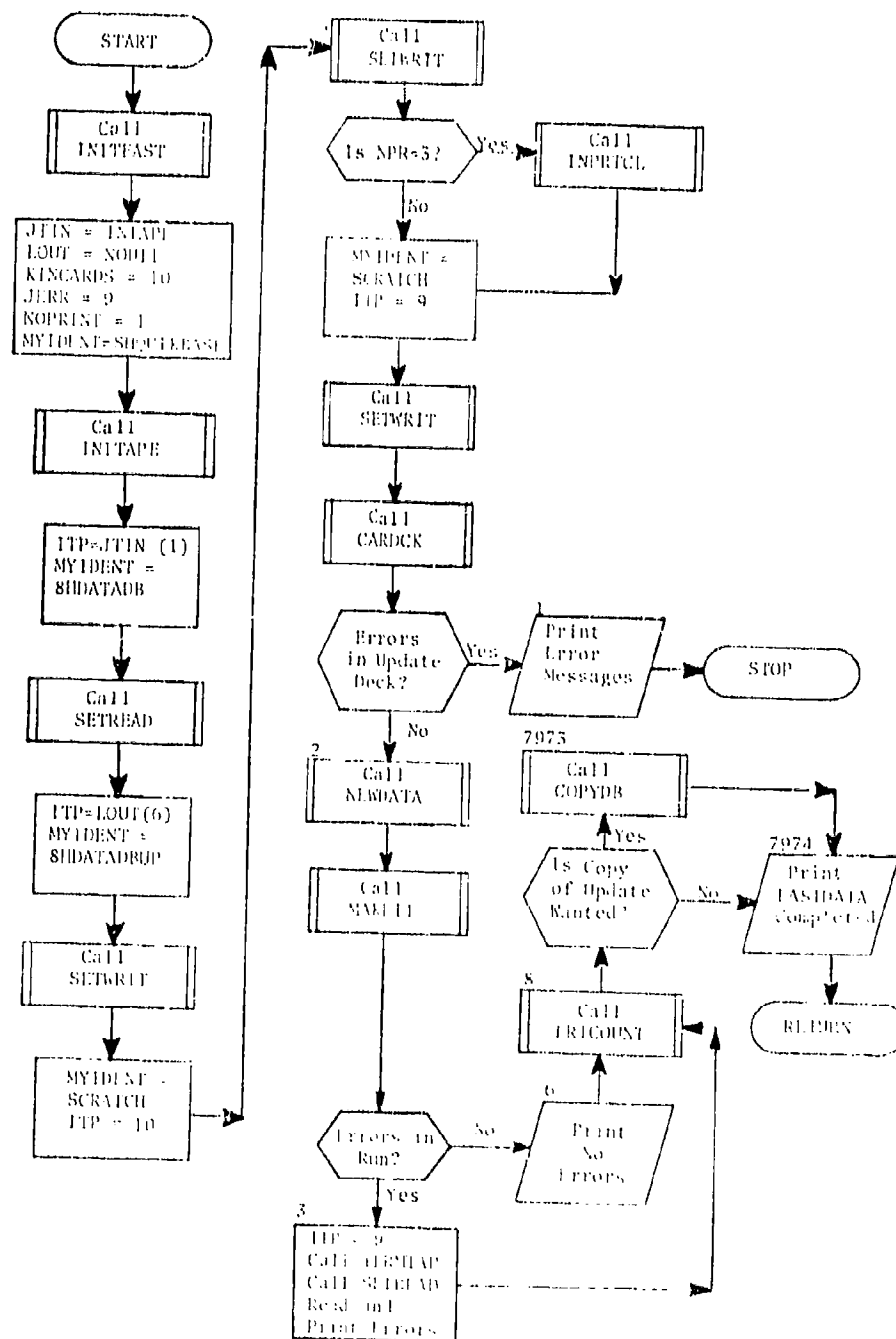


Fig. 85. Subroutine FASTSI 1

FUNCTION ILOOK

PURPOSE: To examine array ID (set numbers to be printed), over IN items, for the presence of I (a set number). The value of the function is 1, if I was found, 0 if I was not found.

ENTRY POINTS: ILOOK

FORMAL PARAMETERS: IN - The number of items to be searched
I - The item to search for

COMMON BLOCKS: SETIDD

SUBROUTINES CALLED: None

CALLED BY: ADDSET, IPRINT

Method

Function ILOOK examines the flag INDEX in common block /SETIDD/ to determine if all sets are to be printed. If INDEX is 9999, a 1 is returned. If the flag is not set, array ID in common block /SETIDD/ (preset by INPRTCL and/or ADDSET) is examined for the occurrence of I. Function ILOOK is illustrated in figure 84.

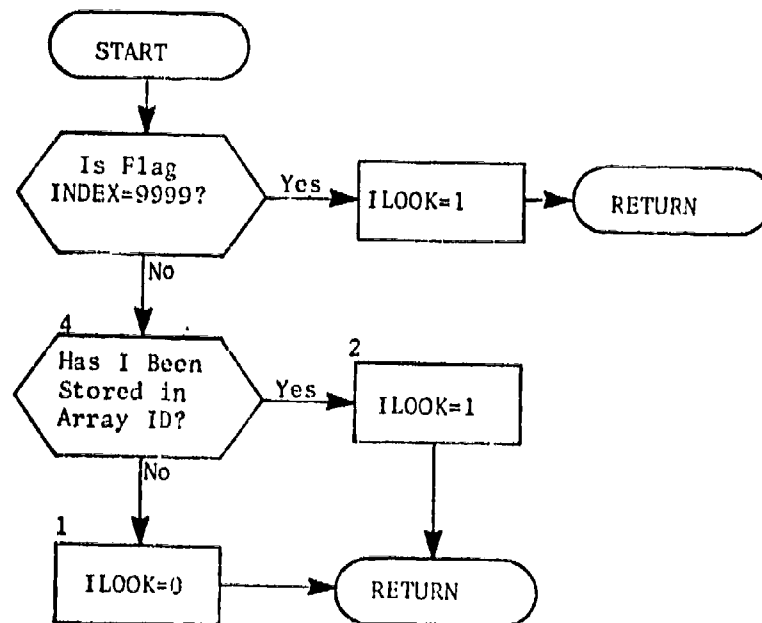


Fig. 84 Function ILOOK

SUBROUTINE INITFAST

PURPOSE: To set constants and clear arrays for subroutines associated with QUIKBASE.

ENTRY POINTS: INITFAST

FORMAL PARAMETERS: None

COMMON BLOCKS: DIRECTRY, ERRORM, ICKTST, ICONTROL, IDESIGS, IENDSET, ITP, ITSTUFF, IWSIDE, JDESTEST, KKSET, MPRTOPT, MYGOODS, MYIDENT, MYINPUT, MYOUT, MYPRINT, MYTAPES, NEWSSET, NODESIGS, NOERRORS, NOPRINT, NOTEST, PRTOPT, SETIDD, TWORD

SUBROUTINES CALLED: None

CALLED BY: FASTSET

Method

INITFAST does no computation. Either through data statements or executable statements, it presets constants and arrays to their appropriate values. Subroutine INITFAST is illustrated in figure 85.

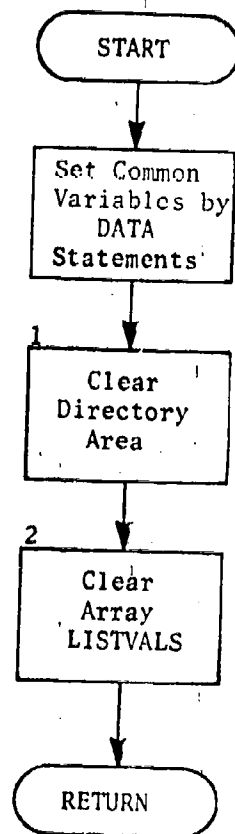


Fig. 85. Subroutine INITFAST

SUBROUTINE INPRTCL

<u>PURPOSE:</u>	To read print control cards for FASTSET (update) if print control option was selected, and to set appropriate flags to monitor printing during the execution of the main program.
<u>ENTRY POINTS:</u>	INPRTCL
<u>FORMAL PARAMETERS:</u>	None
<u>COMMON BLOCKS:</u>	ICKTST, SETIDD
<u>SUBROUTINES CALLED:</u>	NUMGET
<u>CALLED BY:</u>	FASTSET

Method

Subroutine INPRTCL reads a print control card (option UPDATE) and determines if the first word is ALL. If it is, flag INDEX is set to 9999 and the subroutine returns to the calling program. If the card contains the word AUTO, the flag IAUTO is set to true, and another card is read. If the card is CHECKOFF*, the switch to disable the value checking procedures is turned on and a card is read. If the card is neither ALL, AUTO, nor CHEC, it is assumed to be a set number or an end flag. The four characters are translated to digits and, if they are not 9999, they are stored in sequential locations in array ID until the termination flag 9999 is encountered. Counters INDEX and JINDEX are set to the number of set numbers stored.

The command ALL causes all sets to be printed. The command AUTO causes all updated sets to be printed in full. The set numbers listed in array ID will be printed in full whether or not they are updated.

Subroutine INPRTCL is illustrated in figure 86.

*Only first four letters checked.

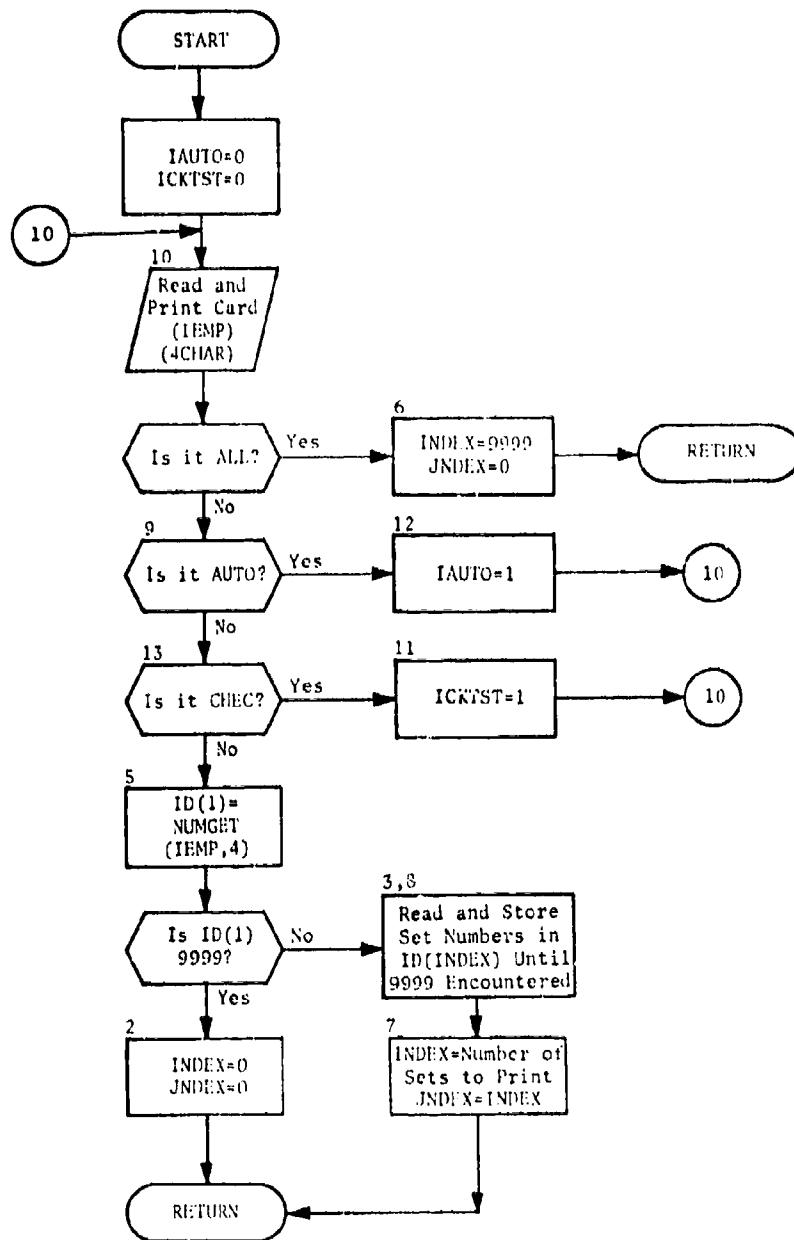


Fig. 86. Subroutine INPRTEL.

FUNCTION IPRINT

<u>PURPOSE:</u>	To determine if a set is to be printed.
<u>ENTRY POINTS:</u>	IPRINT
<u>FORMAL PARAMETERS:</u>	I - The number of the set
<u>COMMON BLOCKS:</u>	SETIDD
<u>SUBROUTINES CALLED:</u>	ILOOK
<u>CALLED BY:</u>	NEWDATA

Method

Function IPRINT examines the flag INDEX to see if either all (INDEX=9999) or none (INDEX=0) of the sets are to be printed. If either condition exists, IPRINT is set to 1 or 0, appropriately. If INDEX is neither value, the flag IAUTO is checked for true (all updated sets are to be printed), or false (specific specified sets are to be printed). If IAUTO is true, the array ID (list of sets to be printed) is searched over the length equal to INDEX; if IAUTO is false the length of the search is INDEX. Function IPRINT is illustrated in figure 87.

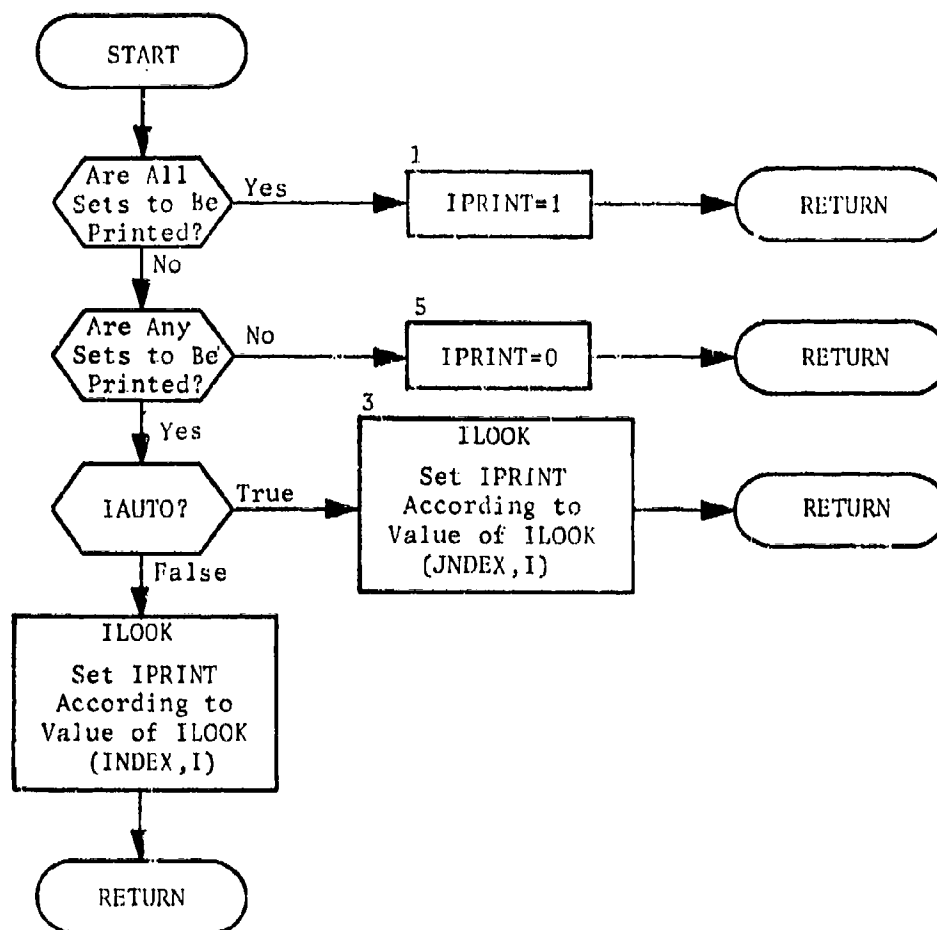


Fig. 87. Function IPRINT

SUBROUTINE MAKEBAS

PURPOSE: To call the subroutines required to prepare a game data base tape QUIKDB from an input data library tape DATADB when updating of the DATADB file is not required.

ENTRY POINTS: MAKEBAS

FORMAL PARAMETERS: None

COMMON BLOCKS: ERRORM, HIST, LOGFLAG, OPTIONS

SUBROUTINES CALLED: ENDDATA, FILEINR, INITFAST, NEWBASE, NEWDATA, NEWDIR, PRNTBASE, PRNTDATA, WRITEDIR

CALLED BY: QUIKBASE

Method

Subroutine MAKEBAS is a driver routine which controls the sequencing of operations required to create the QUIKDB tape when the QUIKDBG option is exercised. As indicated in figure 60, MAKEBAS calls the filchandler (FILEINR) to initialize the read, write, and scratch files. Then, MAKEBAS calls, in order, the other subroutines required to write the QUIKDB tape. In effect, MAKEBAS is essentially the same as a null UPDATE run.

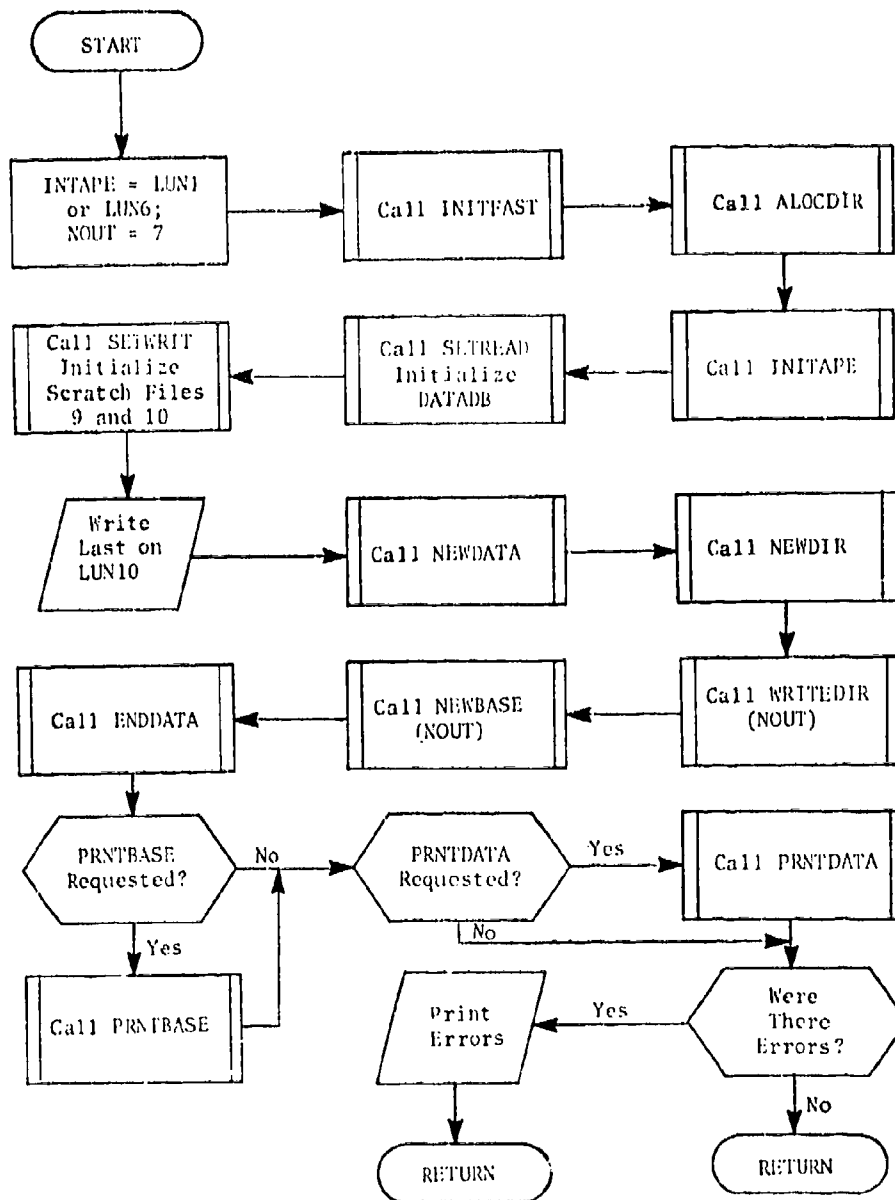


Fig. 88. Subroutine MAKEBAS

SUBROUTINE MAKEIT

PURPOSE: A driver to call the subroutines necessary to make a new data base.

ENTRY POINTS: MAKEIT

FORMAL PARAMETERS: NT1 - The tape where the QUIKDB tape will be written

COMMON BLOCKS: JDESTEST

SUBROUTINES CALLED: ENDDATA, ITLE, NEWBASE, NEWDIR, WRITEDIR

CALLED BY: FASTSET

Method

MAKEIT calls the data base generation subroutines, NEWDIR, WRITEDIR, NEWBASE, and ENDDATA, which create the game base file, QUIKDB. Its only computation function is to look up and store for later use the index number of the attribute DESIG in the data base directory (array ATTNAM). Subroutine MAKEIT is illustrated in figure 89.

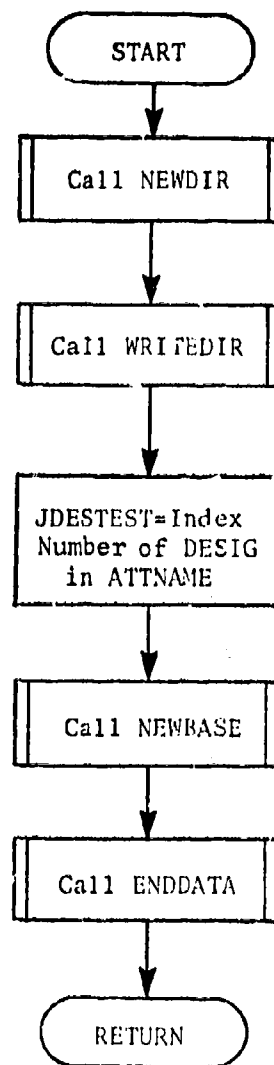


Fig. 89. Subroutine MAKEIT

SUBROUTINE MOVEIT

PURPOSE: To move update data to output buffer; add the desired data or update identification; and to add current set and line number to output record.

ENTRY POINTS: MOVEIT

FORMAL PARAMETERS: IHOWTO - Switch to indicate whether input buffer is to be moved

COMMON BLOCKS: MYOUT, MYINPUT, MYGOODS

SUBROUTINES CALLED: FILEHNR

CALLED BY: NEWDATA

Method

Subroutine MOVEIT increases the current line number within the set by one. It examines the input parameter IHOWTO to determine which of two functions it is to perform. If the argument is equal to a one, MOVEIT transfers a data record from common block /MYINPUT/ to block /MYOUT/. It next sets the tenth word of the block to the update identification. The set and line number are encoded as two four-digit numbers into the ninth word. If the input argument was a two, only the last function is performed by MOVEIT; i.e., the set and line number are encoded into the data record. Subroutine MOVEIT is illustrated in figure 90.

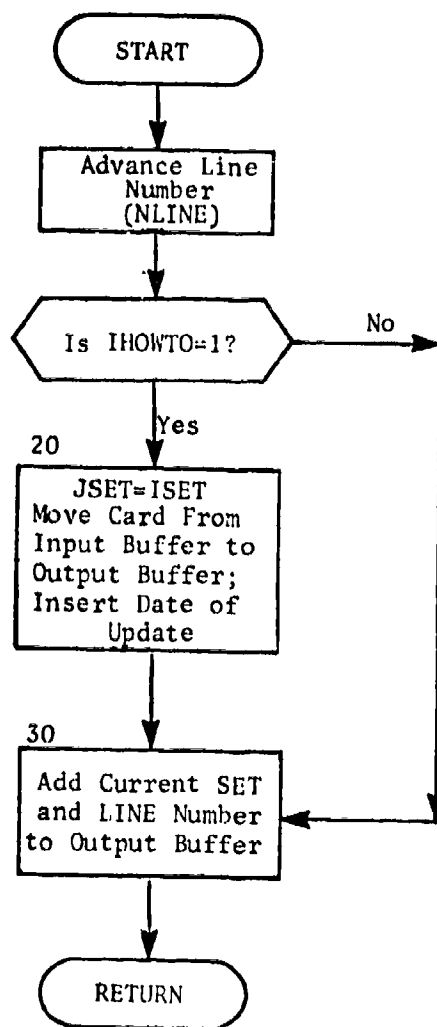


Fig. 90. Subroutine MOVEIT

SUBROUTINE NEWBASE

PURPOSE: To prepare or augment the item portion of a data base tape.

ENTRY POINTS: NEWBASE

FORMAL PARAMETERS: NT1 - the logical tape number where the base is to be written

COMMON BLOCKS: MYGOODS, KKSET, JDESTEST, PRTOPT, DIRECTRY, IENDSET, ITP, TWORD, NOTEST, ERRORM, ICONTROL, MYOUT

SUBROUTINES CALLED: NEWDATA, WRWORD*, WRARRAY*, ITLE, COUNTDS, NUMGET

CALLED BY: MAKEIT

Method

Subroutine NEWBASE employs subroutine NEWDATA to read the item portion of the data base, check for errors, and write each item on the specified output tape, NT1.

Four commands are recognized: DEFINE, UNDEFINE, ITEM, and ENDINPUT. In the case of DEFINE the succeeding fields on the card beginning in columns 11, 21, 31, etc. contain attribute-value pairs which are to be made into global definitions in which the first field of the pair is the BCD name of the attribute, left-justified, followed by the value in the second field. The sequence of attribute-value pairs occurring on a card is terminated by a blank field.

The ITEM card is as described above except that the definition is local and the entire sequence of cards is terminated only upon detection of another command in the first field of a card.

The UNDEFINE card removes global definitions with the names of the attributes to be undefined occurring in succeeding fields on the card, terminated by a blank.

The deck of input cards is terminated by ENDINPUT which also causes NEWBASE to return to the calling program.

*See subroutine FILEHNR.

All cards read by the routine are checked for consistency unless checking has been turned off by an update PRNTCL option card (see subroutine INPRTCL). That is, the attribute specified is checked to determine that it is in fact defined in the data base directory and that the value associated satisfies any range or list check specifications for that attribute. Appropriate error messages are emitted when such inconsistencies are detected. The flowchart (figure 91) consists of three parts. Part I shows the processing sequence used in NEWBASE. Parts II and III show the operations of three local subroutines used by NEWBASE to perform the data checks and, if required, to write error messages. Part II shows the local subroutines used to signal undefined attributes (see statement 110) and to signal an error in the assigned attribute value (see statement 120). Part III shows the procedures used to convert and check the attribute-value pair.

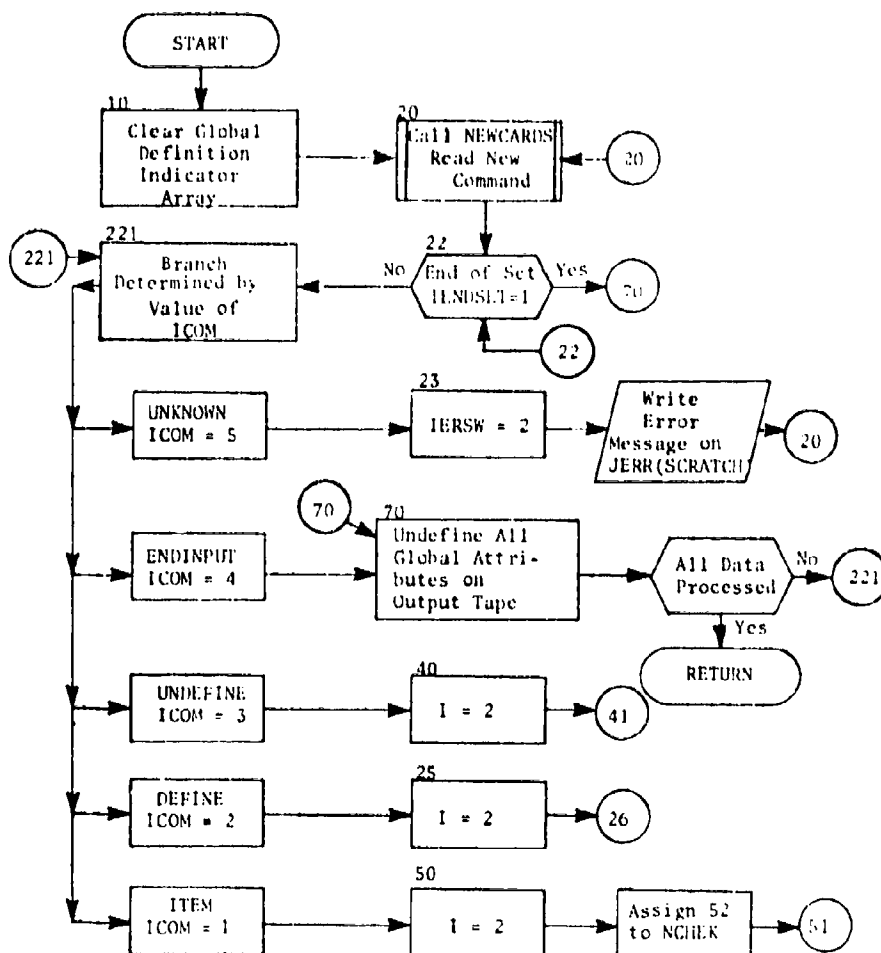


Fig. 91. Subroutine NEWBASE
Part 1: Processing Sequence
(Sheet 1 of 3)

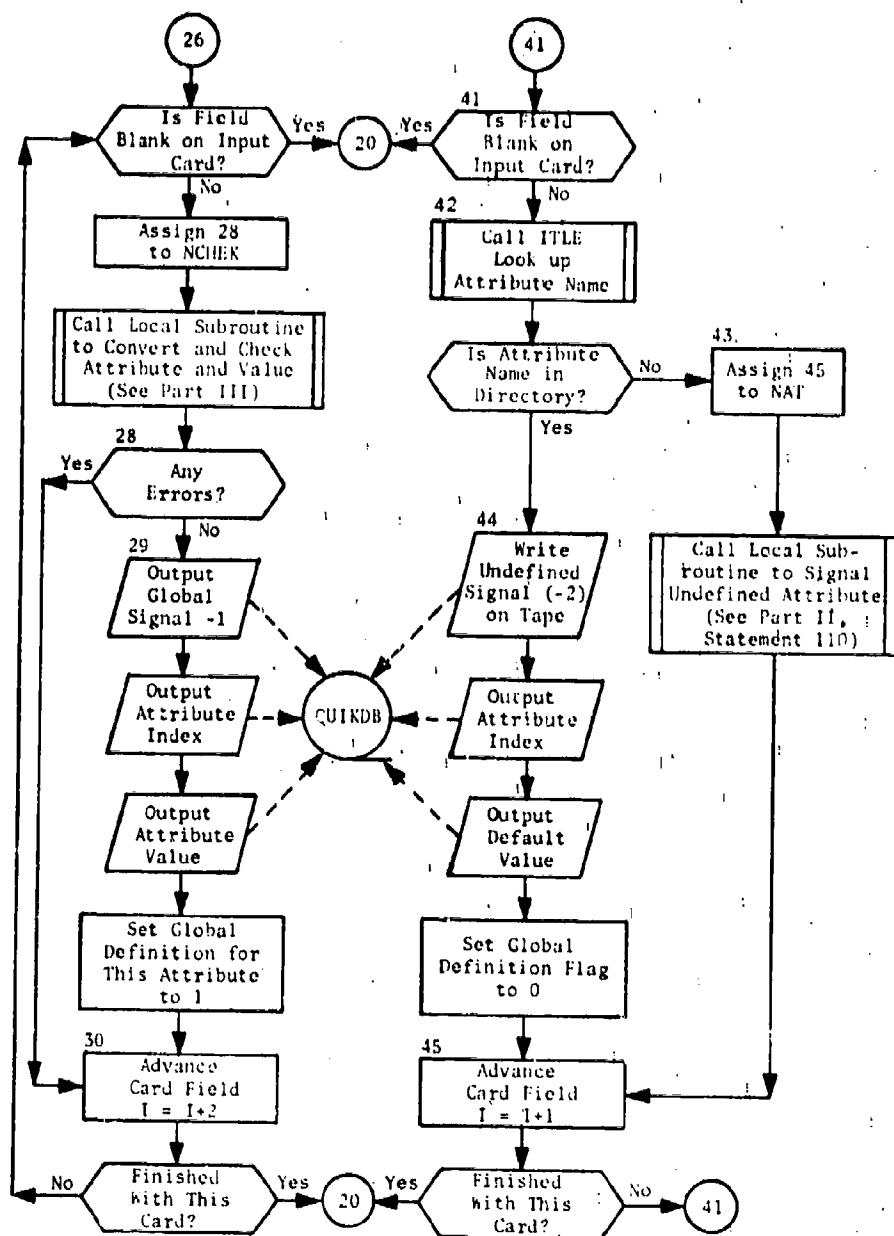


Fig. 91. (cont.)
Part I: (cont.)
(Sheet 2 of 3)

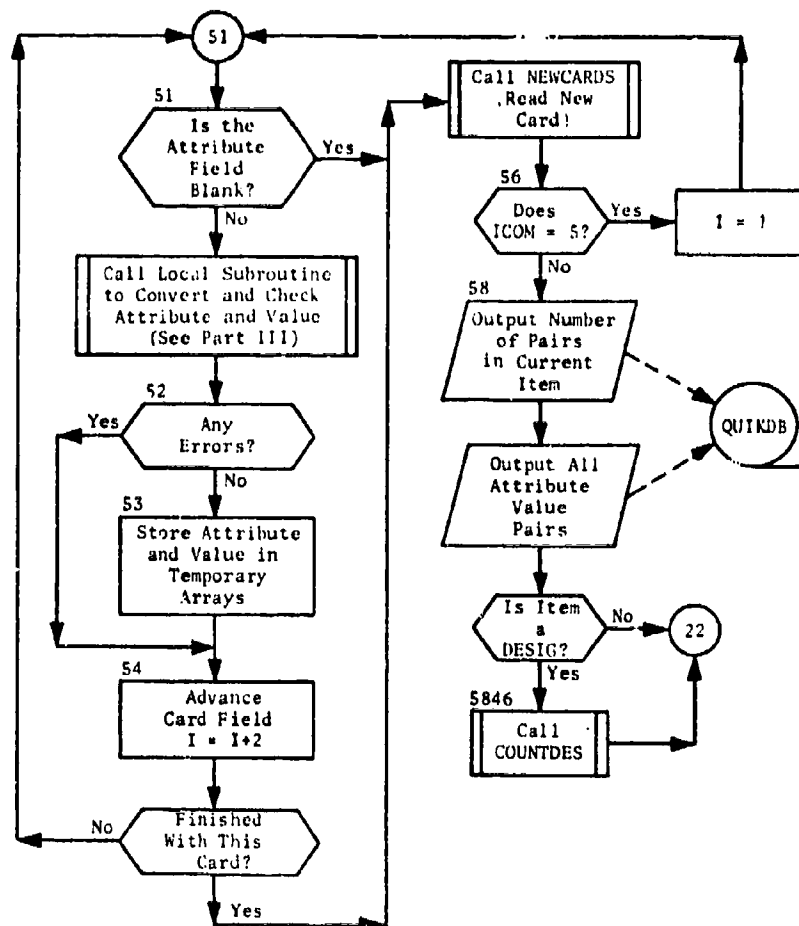


Fig. 91. (cont.)
Part I: (cont.)
(Sheet 3 of 3)

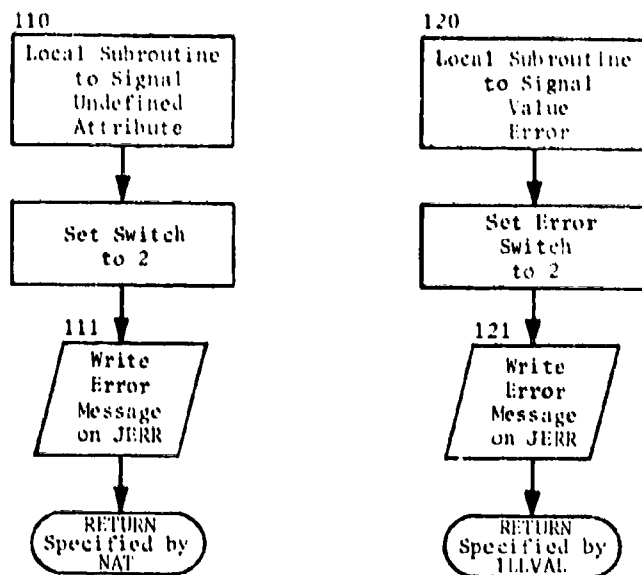


Fig. 91. (cont.)
Part 11: Local Subroutines
Error Processing

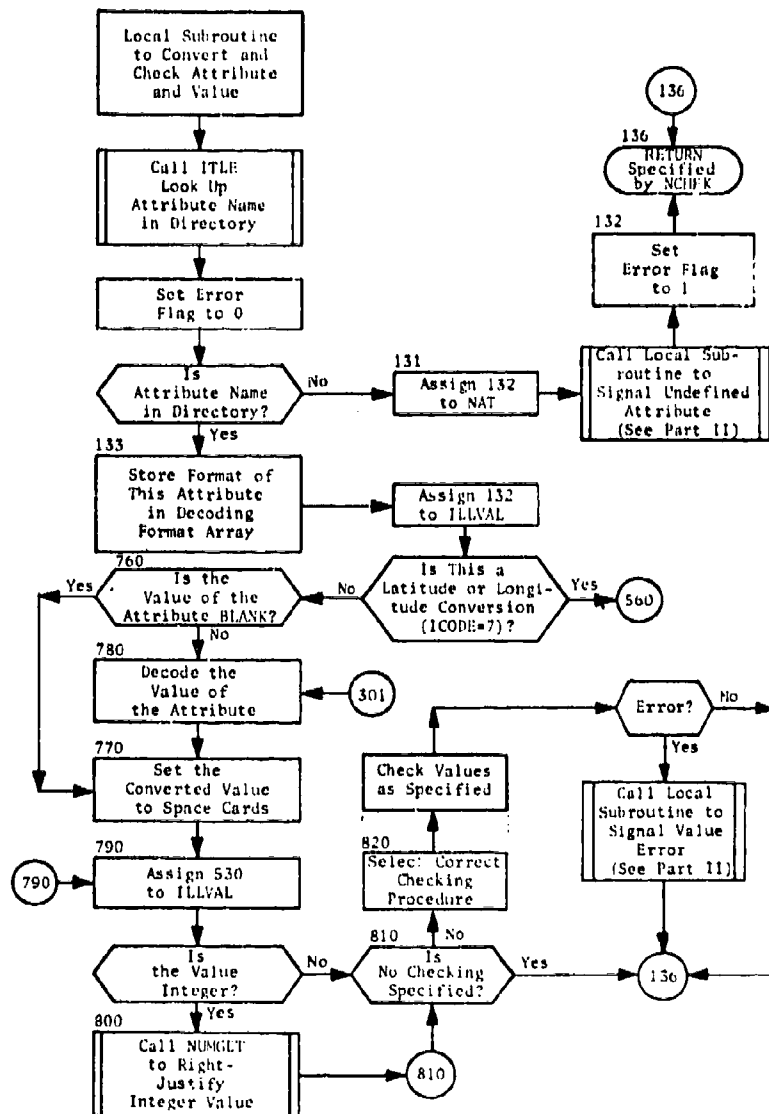


Fig. 91 . (cont.)
 Part III: Local Subroutine for
 Attribute/Value Checking
 (Sheet 1 of 2)

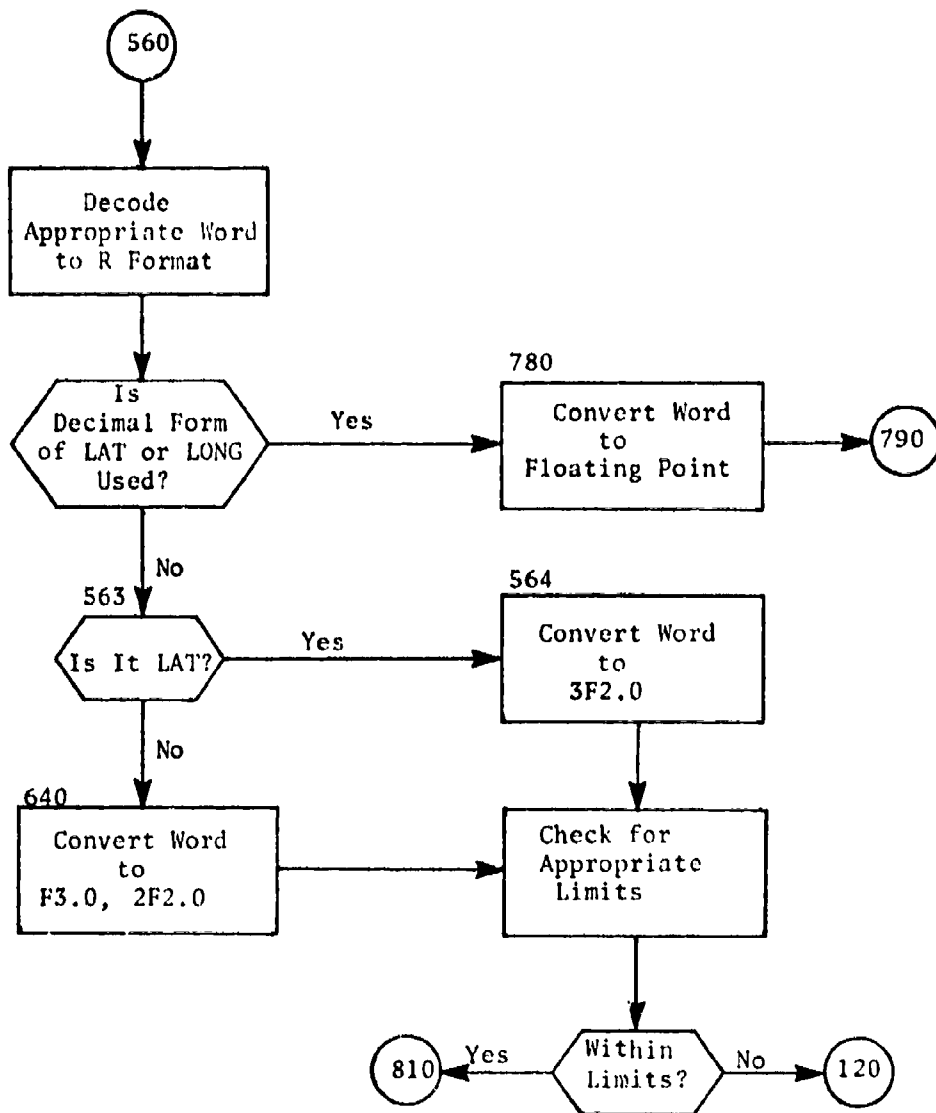


Fig. 91. (cont.)
Part 111: (cont.)
(Sheet 2 of 2)

SUBROUTINE NEWDATA

PURPOSE:

To perform the following functions in conjunction with the FASTSET update of program QUIKBASE:

1. A call on Entry NEWDATA initializes errors and constants for NEWCARDS.
2. A call on NEWCARDS returns the next valid data base card to the calling program in OUTSTUFF.
3. To translate ITEM, DEFINE, UNDEFINE, and ENDINPUT to ICOM=1, 2, 3, and 4, respectively.
4. To act on all editing and deleting commands; to write updated DATADB.

ENTRY POINTS:

NEWDATA, NEWCARDS

FORMAL PARAMETERS:

None

COMMON BLOCKS:

ICKTST, ICONTROL, IENDSET, ITP, MYGOODS, MYINPUT, MYOUT, MYPRINT, MYTAPES, NEWSET, NOTEST

SUBROUTINES CALLED:

BUFFIT, FILEHNR, IPRINT, MOVEIT, NUMGET, OUT, PAGESKP

CALLED BY:

NEWBASE, NEWDIR, FASTSET

Method

Subroutine NEWDATA is an initiating entry which presets constants to zero, true, or false, as appropriate, and prepares the tape written by CARDCK to be read.

Entry NEWCARDS is the primary read and command editing subroutine for the FASTSET update option. A call on NEWCARDS returns a single data base item to the calling program. The subroutine reads a control card record and determines if a record is to be deleted, replaced, or added. It matches update data against a DATADB tape using the set and line numbers to control the position of both tapes. The control cards indicate the source of new data which may be individual BCD images from card or tape, or which may be packed, buffered records on tape. Each data card is examined to see if it is an ITEM, DEFINE, UNDEFINE, or ENDINPUT card. If one of these is encountered, the common variable ICOM is set to 1, 2, 3, or 4, respectively. If the card is none of those listed, ICOM is

returned equal to 5. Set and line numbers are added to each record, as is the run identification. The output tape DATADBUP is written, and the items are printed if the print option for the set being read was selected. Error messages are written to the error summary tape and, in all cases where possible, the run continues noting as many errors as possible. In the event an error cannot be circumvented (e.g., the data base set numbers requested on the update command cards cannot be found), the run is aborted. Subroutine NEWDATA is illustrated in figure 92.

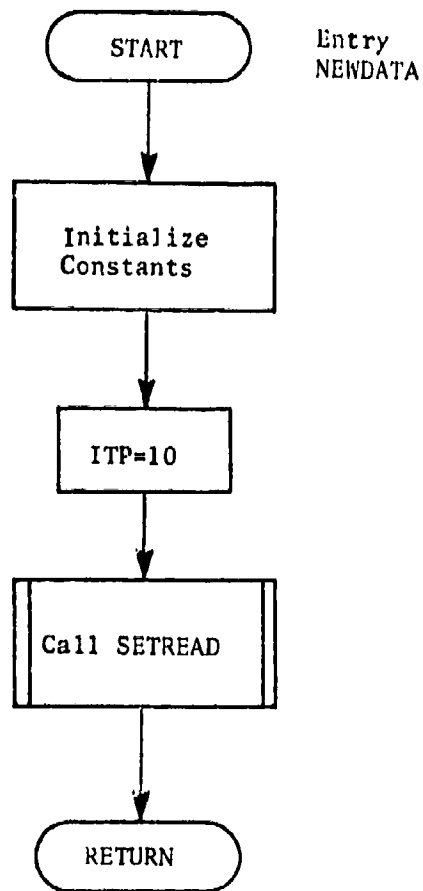


Fig. 92. Subroutine NEWDATA
(Sheet 1 of 7)

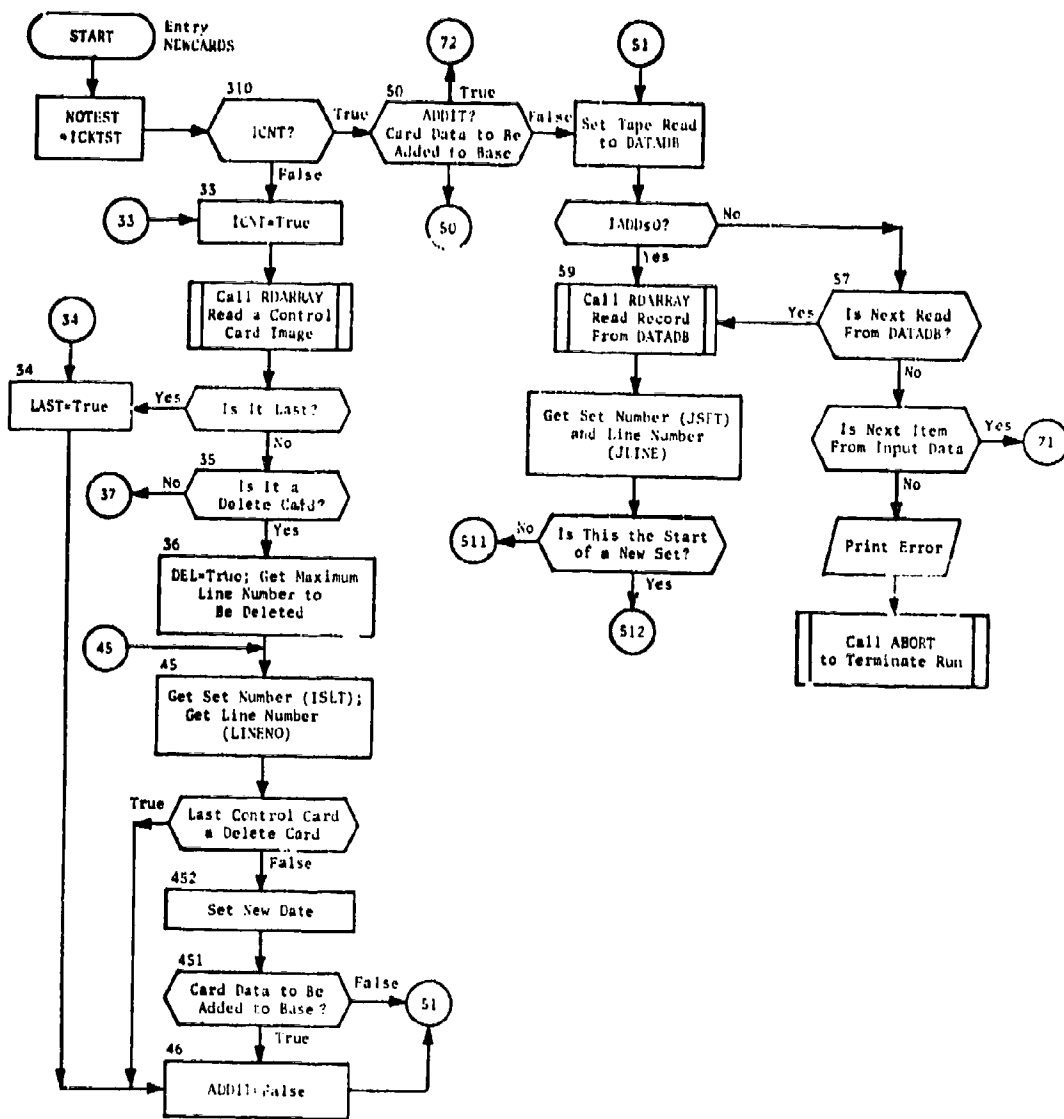


Fig. 92. (cont.)
(Sheet 2 of 7)

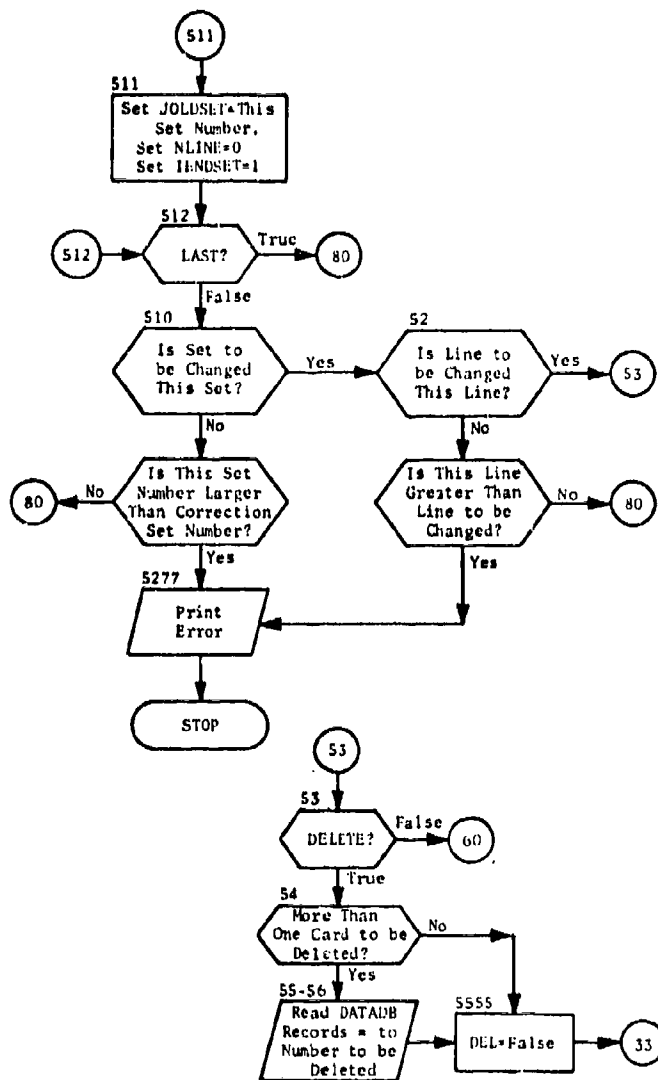


Fig. 92. (cont.)
(Sheet 3 of 7)

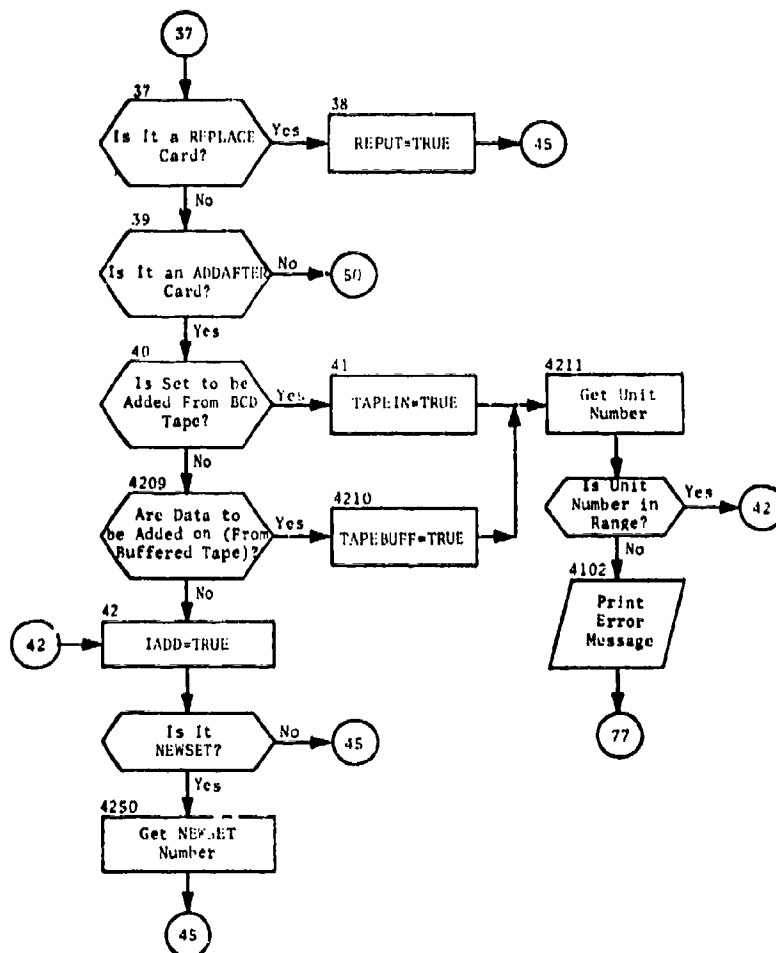


Fig. 92. (cont.)
(Sheet 4 of 7)

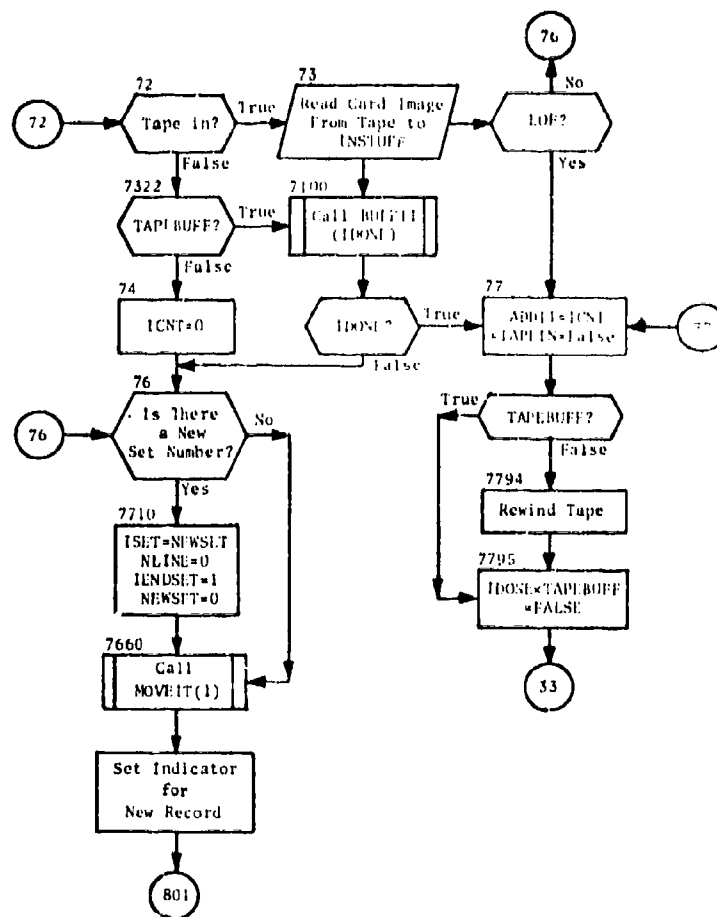


Fig. 92. (cont.)
(Sheet 5 of 7)

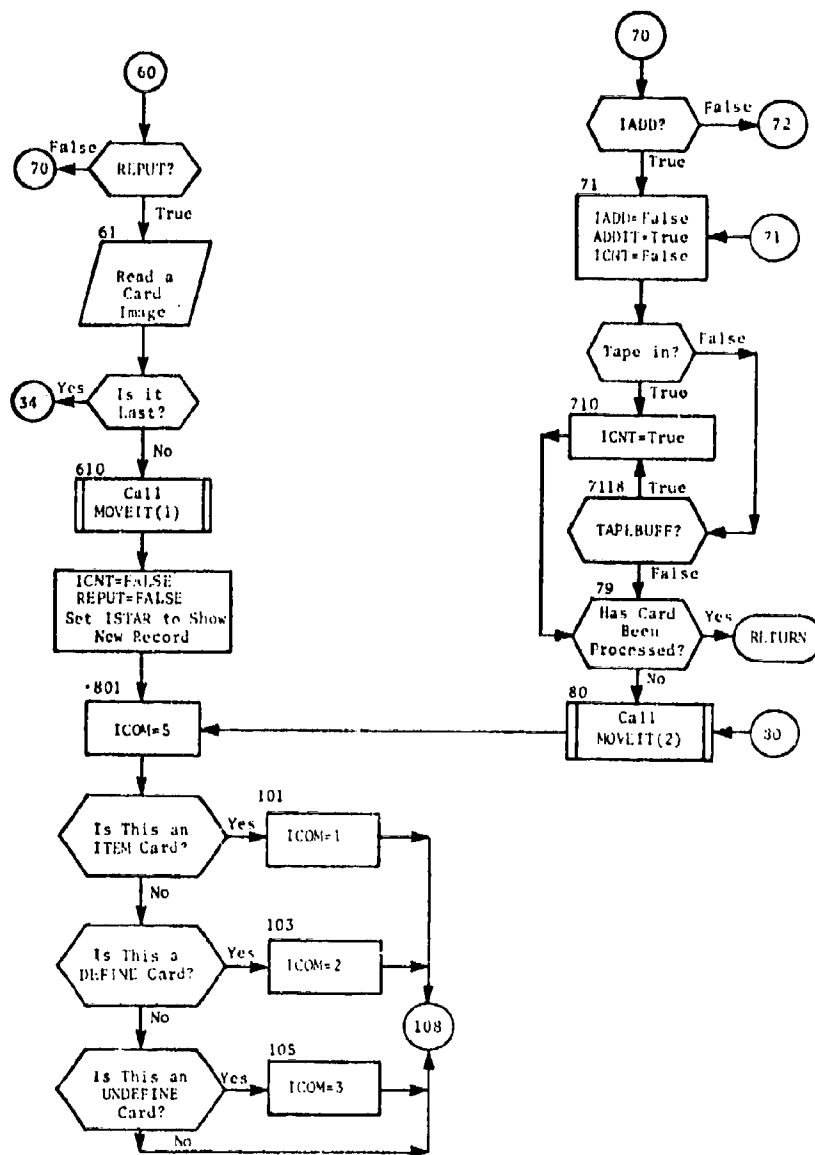


Fig. 92. (cont.)
(Sheet 6 of 7)

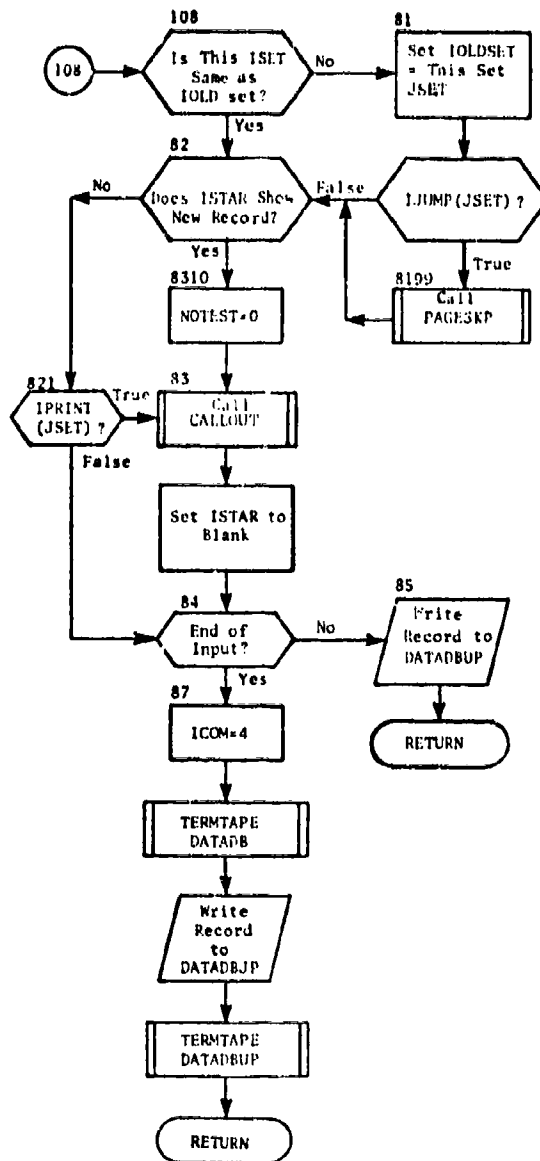


Fig. 92. (cont.)
(Sheet 7 of 7)

SUBROUTINE NEWDIR

PURPOSE: To create a new data base directory.

ENTRY POINTS: NEWDIR

FORMAL PARAMETERS: None

COMMON BLOCKS: DIRECTORY, ERRORM, ITP, MPRTOPT, MYOUT, TWORD

SUBROUTINES CALLED: ITLE, NUMGET, NEWDATA (Entry NEWCARDS)

CALLED BY: MAKEIT

Method

This subroutine employs subroutine NEWDATA (entry NEWCARDS) to read a new directory or modify an existing directory. The card format is eight fields of 10 columns each with all quantities left-justified. Two commands in the first field are recognized: ADD and ENDIRECT. The ENDIRECT card serves to terminate the subroutine and causes a return to the calling program.

The ADD command is used to add a new attribute to the directory, or, in conjunction with a prior delete command, to change an already existing attribute in the directory.

With the ADD command there are, in addition to the first field, six further fields of data on the input card:

1. The name of the attribute in BCD.
2. The input/output conversion format (FORTRAN) associated with the values for that attribute.
3. Code number specifying the type of checking to be conducted for a particular attribute (see below).
4. The default value of the attribute, in the appropriate input/output format for that attribute as specified by item 2. This is the value that will be associated with the attribute when it is in an undefined state.

5. Checking specifications. This field may contain the word LIST, which specifies list checking with the list of allowable attribute values to follow on subsequent cards; the word NOCHECK, which specifies no checking of the attribute values; or the lower value of the allowable range of values for this attribute in the case of range checking.
6. This field is unused in the event of list checking, or no checking, and contains the upper value for the range of allowable values of the attribute in the case of range checking.

If list checking is specified on the ADD card, this card is followed by any number of cards containing the list of allowable values for that attribute, eight per card, in the format specified for the particular attribute. The fields for these values are the first eight columns of each 10-column field. The series of allowable values in these cards is terminated by the first blank field. A blank field can be specified as an allowable attribute value by including the value BLANK in the list.

Appropriate error messages are written on an error message tape to point out inconsistent operations such as attempting to issue a command other than ENDIRECT or ADD, or attempting to add attributes which already exist in the directory. The error-checking codes permissible for the third extra field are:

<u>CODE</u>	<u>TYPE OF DATA TO BE INPUT</u>	<u>CHECKING SPECIFIED</u>
1	Floating point numeric	Range (Min-Max)
2	Floating point numeric	List
3	Fixed point numeric	Range (Min-Max)
4	Fixed point numeric	List
5	Alphameric	List
6	Alphameric	No checking
7	Special conversion for latitude, longitude	Range

Subroutine NEWDIR is illustrated in figure 93.

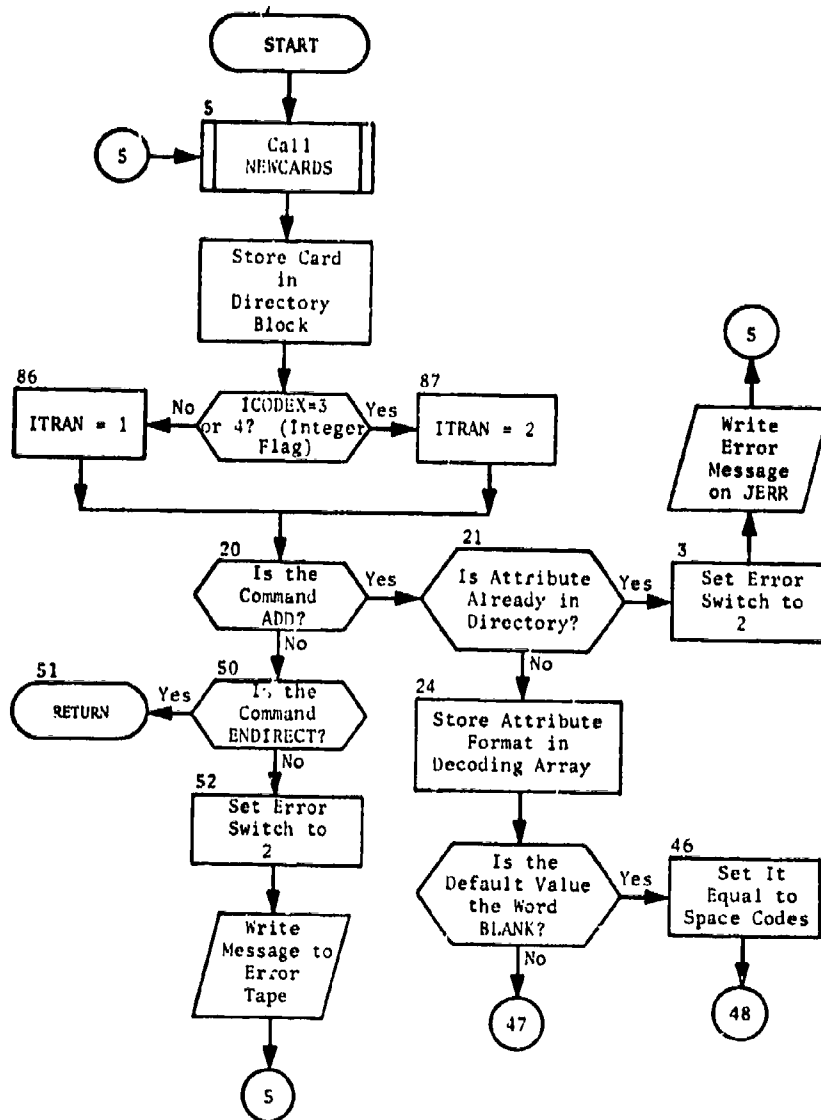


Fig. 93. Subroutine NEWDIR
(Sheet 1 of 3)

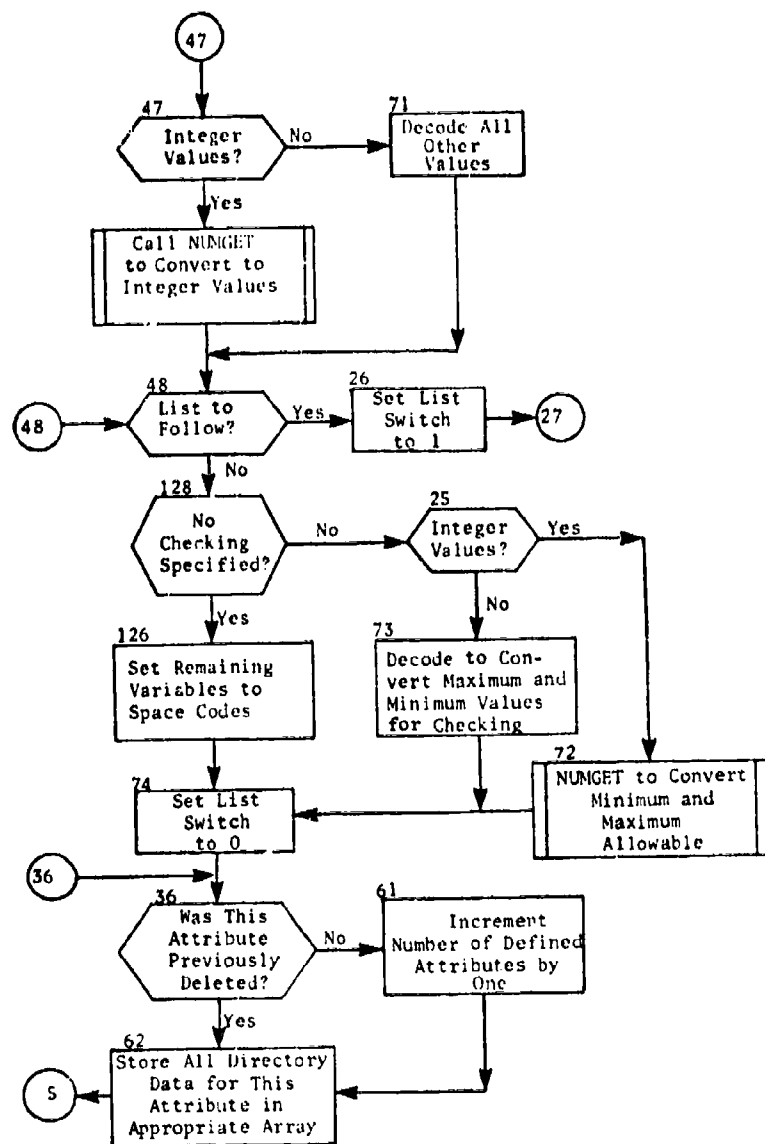


Fig. 93. (cont.)
(Sheet 2 of 3)

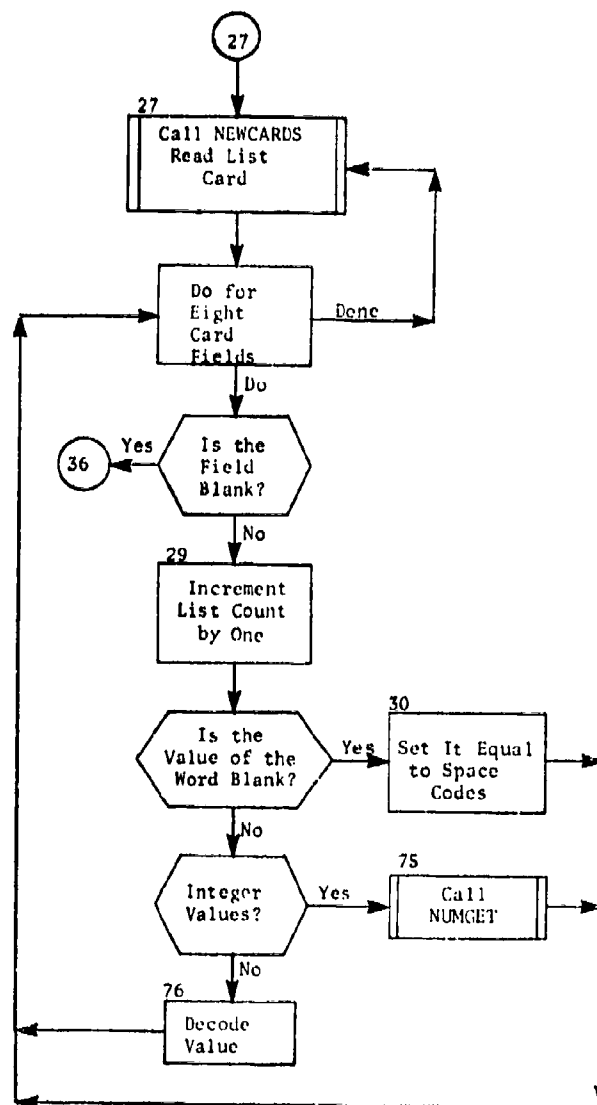


Fig. 93. (cont.)
(Sheet 3 of 3)

SUBROUTINE OUT

PURPOSE: To print an item from the data base tape DATADB.
ENTRY POINTS: OUT
FORMAL PARAMETERS: None
COMMON BLOCKS: ICONTROL, MYOUT, MYPRINT
SUBROUTINES CALLED: None
CALLER BY: NEWDATA

Method

Subroutine OUT examines the "type" switch (ICOM) to determine the print format. ITEM cards are printed with a preceding blank; DEFINE cards are printed with a preceding *; UNDEFINE cards are printed with a preceding **. Subroutine OUT is illustrated in figure 94.

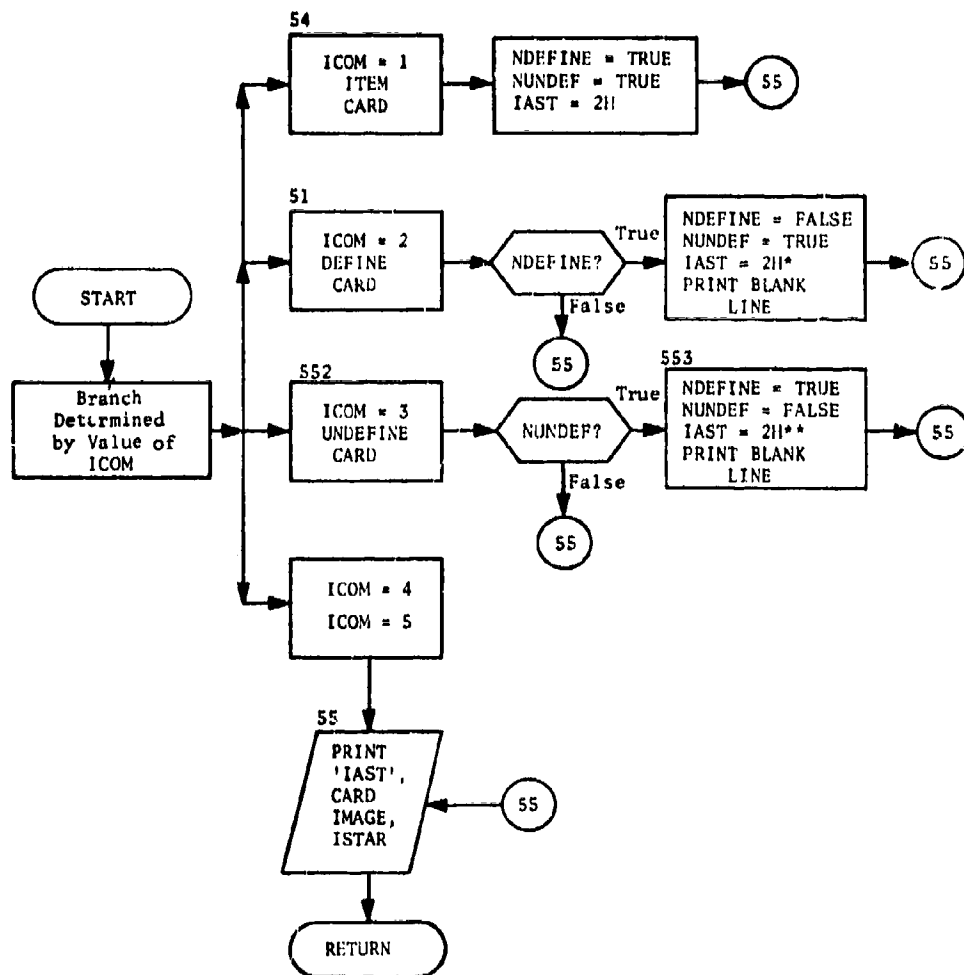


Fig. 94. Subroutine OUT

SUBROUTINE PRONLY

PURPOSE: To permit the user to print the contents of the data base tape QUIKDB, as prepared by program QUIKBASE, in either of two standard output formats.

ENTRY POINTS: PRONLY

FORMAL PARAMETERS: None

COMMON BLOCKS: HIST, MYIDENT, OPTIONS

SUBROUTINES CALLED: INITAP, PRNTBASE, PRNTDATA

CALLED BY: QUIKBASE

METHOD

The PRINTDB option of program QUIKBASE provides the capability of printing the contents of the data base tape QUIKDB. Inclusion of the PRINTDB option card in the program run deck serves to select the option and also identifies the desired output print format.

When this option is selected, subroutine PRONLY is called by QUIKBASE to determine the desired print format and call the appropriate utility routine to accomplish the printing.

As shown in figure 95, PRONLY first initializes the filehandler by calling entry INITAP. The third and fourth data fields of the PRINTDB option card are then examined to determine the data base print format which is to be used. Either or both of the standard data base print formats (i.e., PRNTDATA or PRNTBASE) may be obtained.

The order in which the prints are requested is immaterial. PRONLY determines the first print request and calls the associated subroutine: PRNTDATA or PRNTBASE (entry PRNTBAS is used). The second request is processed in the same manner and control is returned to the calling program.

Subroutine PRONLY is illustrated in figure 95.

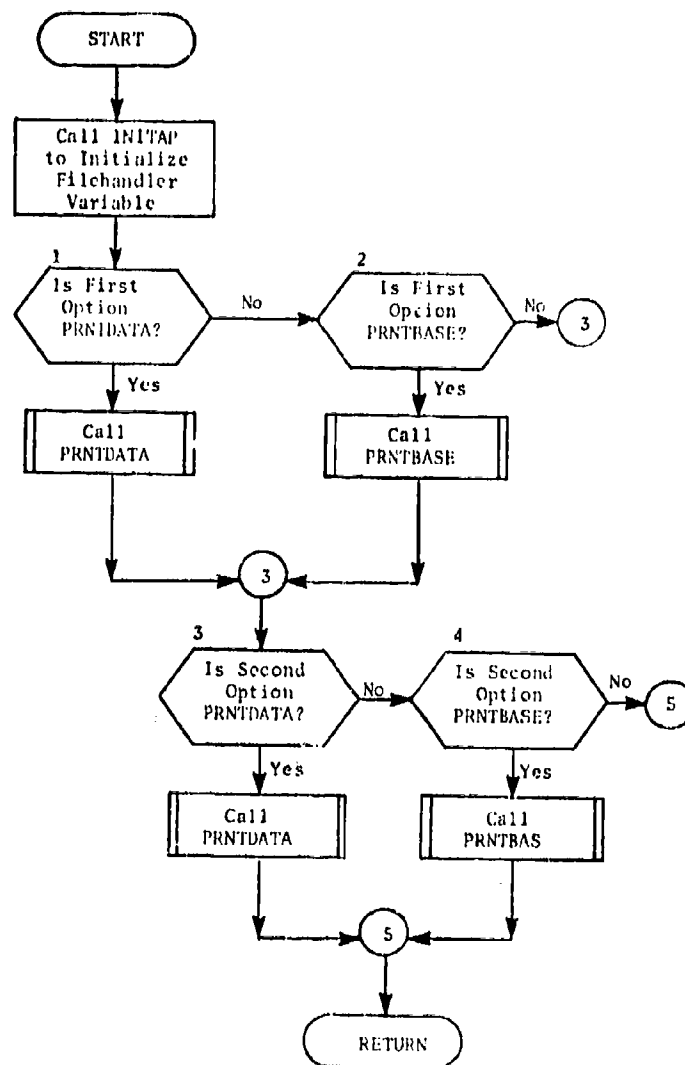


Fig. 95. Subroutine PRONLY

SUBROUTINE PRTCONT

PURPOSE: To summarize (print) the number of targets by side, region, and target type.

ENTRY POINTS: PRTCONT

FORMAL PARAMETERS: None

COMMON BLOCKS: IDESIGS, NODESIGS, IWSIDE

SUBROUTINES CALLED: PAGESKP

CALLED BY: FASTSET

Method

Subroutine PRTCONT prints information tallied by subroutine COUNTDS in common blocks /IDESIGS/ and /NODESIGS/. Subroutine PRTCONT is illustrated in figure 96.

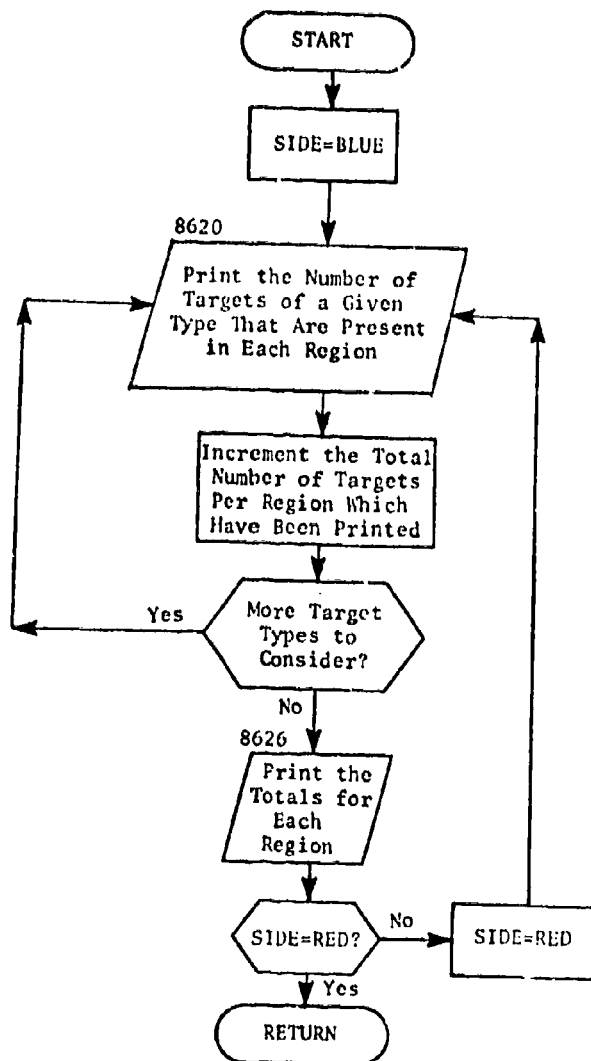


Fig. 96. Subroutine PRTCONT

SUBROUTINE SETID

PURPOSE: To begin a data base from cards or card images and to write the DATADB tape.

ENTRY POINTS: SETID

FORMAL PARAMETERS: None

COMMON BLOCKS: 1, FILABEL, IFTPRNT, ITP, MYIDENT, MYLABEL, NOPRINT, OPTIONS, TODAY, TWORD

SUBROUTINES CALLED: FILEHNR, NUMGET

CALLED BY: QUIKBASE

Method

Subroutine SETID is used to create a data library file (DATADB) from a card deck or a tape containing BCD card images. The data library tape (DATADB) is formatted for ease in updating the information contained thereon. The tape consists of a series of card images with three identifiers for each card. The identifiers consist of a set number, a line number, and an update identifier; e.g., the date. The card images are divided into sets, each with a unique number. Within each set, each card is assigned a consecutive line number. Thus, each card is uniquely defined by its set and line number. The update identifier is an eight-character word which aids the user in determining the run which introduced the card into the data library file.

The user may specify that one of three methods be used to divide the data library into sets (see SETID Option, program QUIKBASE, chapter 2, User's Manual, Volume II). The default option (SIDECLAS) causes sets to be established considering the attributes SIDE and CLASS. Under this method, the data base directory is defined as the first set. The set number is then increased by one each time a DEFINE card is processed which includes the attribute SIDE and/or CLASS. The MAXSET option causes the data to be divided into the largest feasible sets. Under this option, the directory is defined as the first set, and each succeeding 5,000 cards are defined as a new set. Selection of the MAXSET option also causes the existing global attribute definitions to be carried over to the next set. The third option (MANUAL) enables the user to specify the exact points in the data base where a new set is to be started. The user may also establish the set numbers; however, they must be assigned in ascending order and the directory must be the first set.

As shown in figure 97, SETID initializes the filehandler and prepares to write one or two copies of the DATADB tape, as requested. The subroutine reads the directory cards; if manual division of sets has been selected, it looks for and uses (if present), a user-assigned set number. The directory is read and written to one or two tapes, as directed, until it reads the ENDDIRECT card. Each attribute name is stored in array NS as it is encountered. On option, each record is printed with its assigned set, line, and date identification. After the directory has been completed, the set number is advanced by one if manual control of sets has been selected. For all other options, the set number is made two. The item cards are read and processed according to the set option chosen.

If the card is neither a DEFINE nor an UNDEFINE card, the print option is checked and exercised according to its value; the set and line number and the update identifiers are inserted into the data record; and the record is written to one or two tapes. If a card is a DEFINE card, a NEWSET or BEGINSET card, or an UNDEFINE card, processing is as described below:

1. If the occurrence of either SIDE or CLASS is to indicate the end of one set and the beginning of the next, each DEFINE card is searched for the key word; i.e., SIDE or CLASS. If the key word is found, the set number is advanced, the line number is reset to one, and the processing continues as described above.
2. If manual control has been selected, the word BEGINSET or NEWSET is sought and, if found, the optional set number is sought. If it is present, it is used; if not present, the set number is advanced by one and the line number reset to one.
3. If the MAXSET option has been selected, the list of attributes on each DEFINE card is searched, and the name of the attribute and its value are saved. The occurrence of an UNDEFINE card causes the attribute-value pair to be erased. When the line number reaches ISETSLZ, the preset maximum* for a set (now set to 5,000), the set number is advanced; the line number is reset to one; and all defined attribute-value pairs are written to tape as the beginning items of the new set.

When the card ENDDINPUT or LAST is encountered, the tape writing is terminated and, if two tapes have been generated, the tapes are compared. If discrepancies are found, they are printed. If no errors are found, the record count, together with the message NO ERRORS, are printed. Control is returned to the calling program QUIKBASE.

*Approximate number. All ITEM card attributes for the current data base item are processed before the set number is changed.

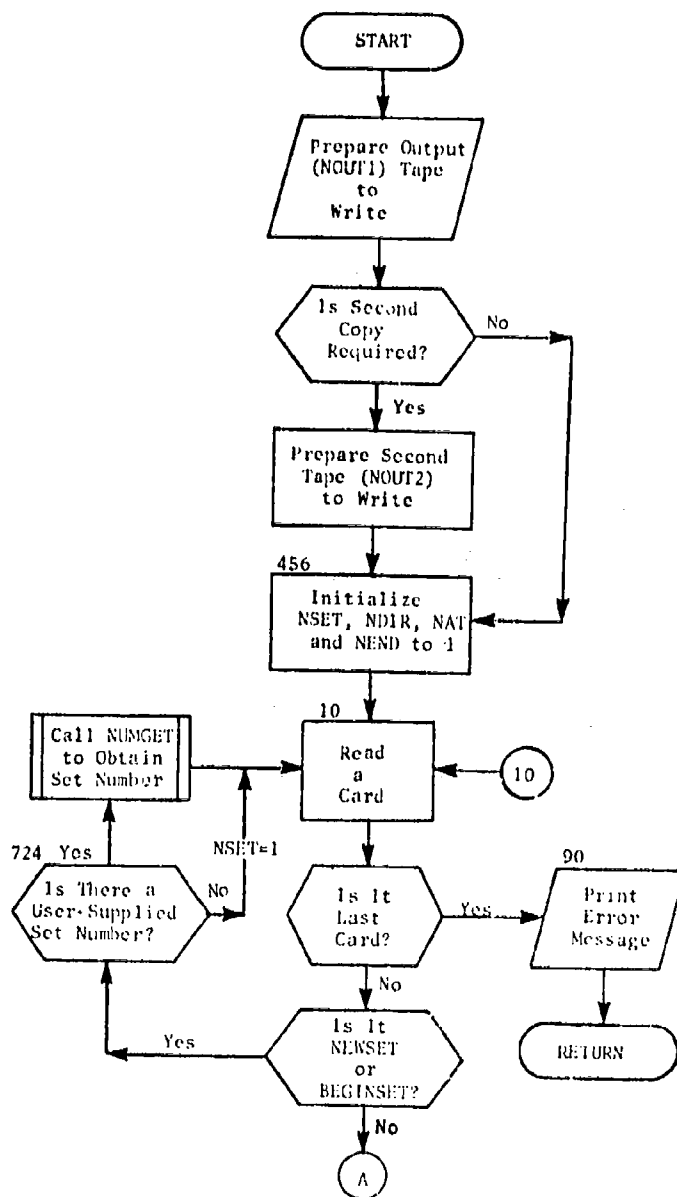


Fig. 97. Subroutine SETID
(Sheet 1 of 7)

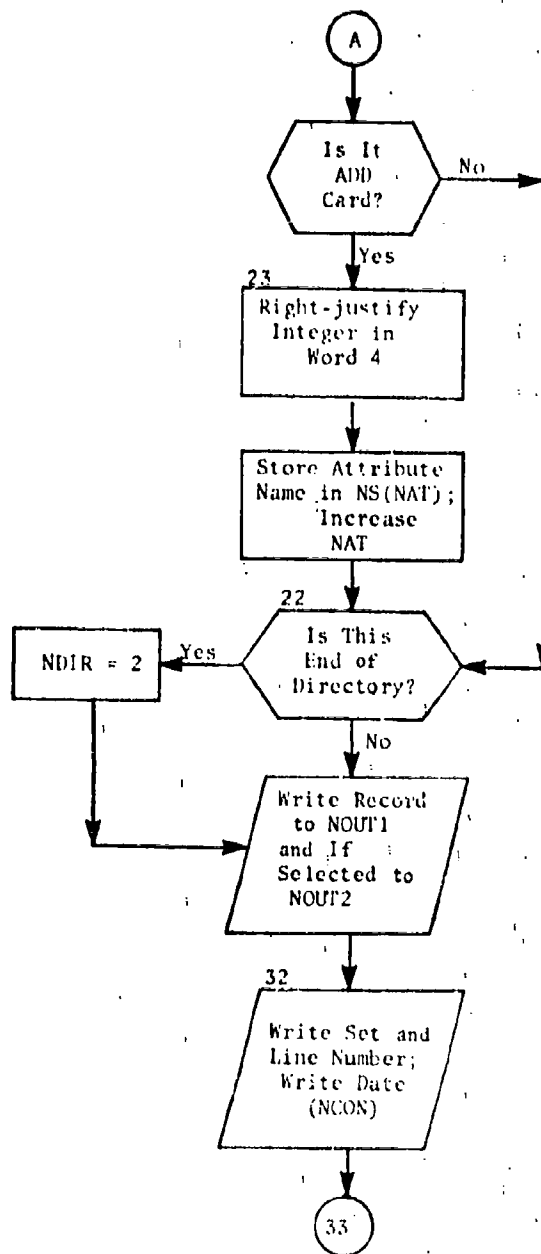


Fig. 97. (cont.)
(Sheet 2 of 7)

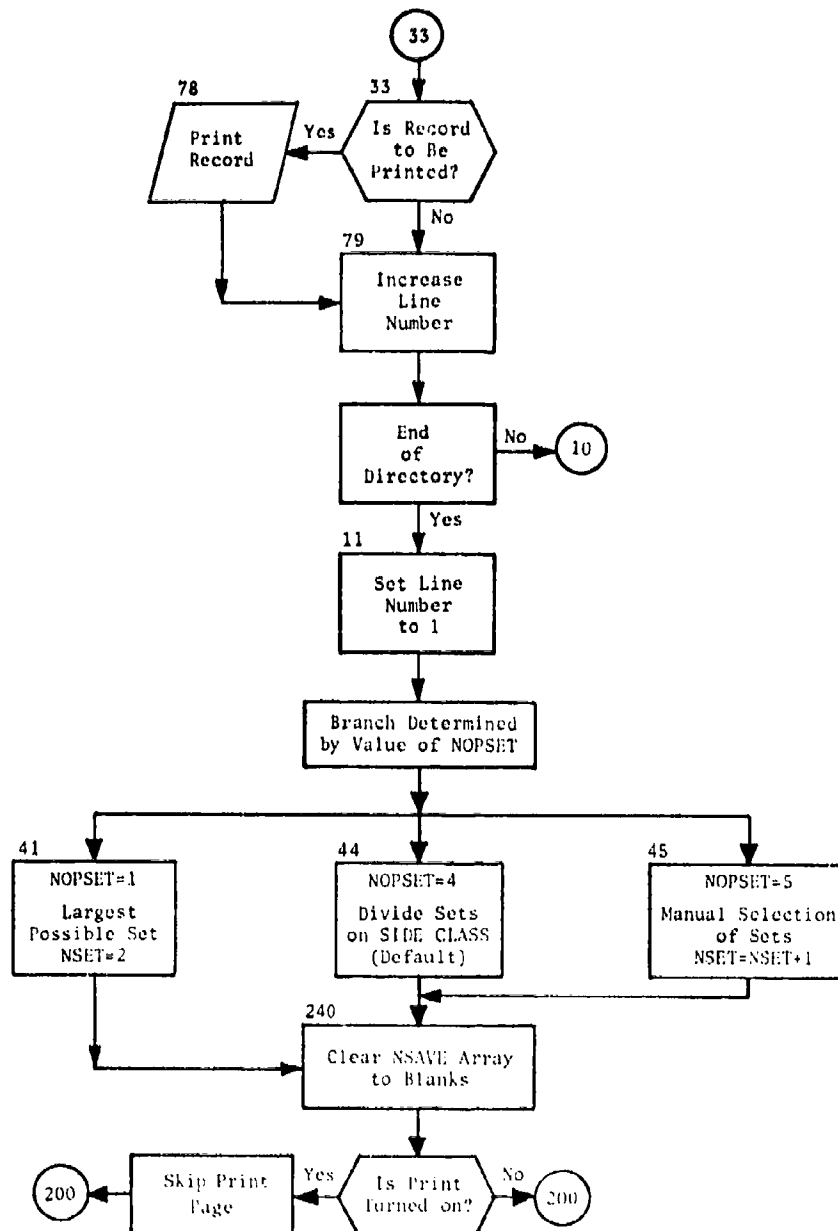
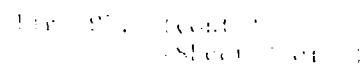
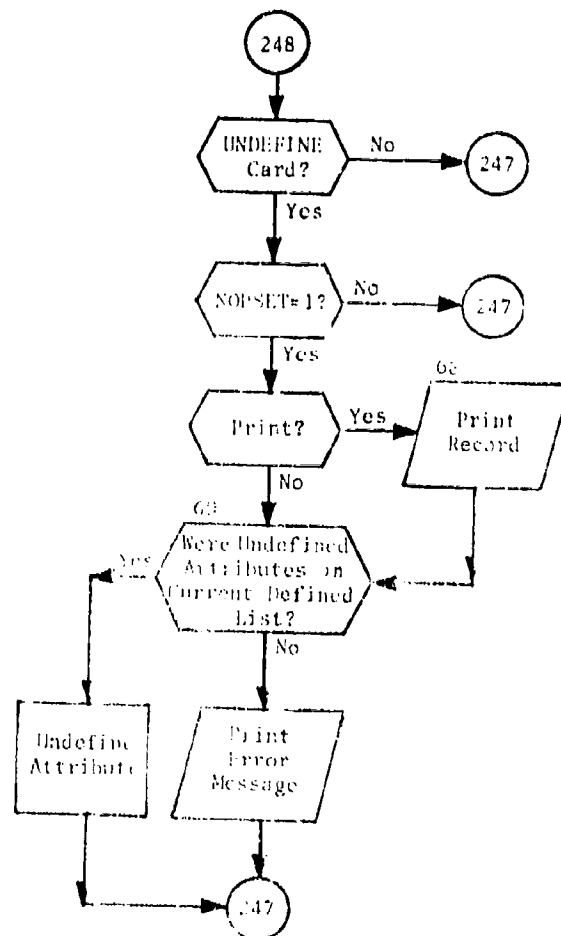


Fig. 97. (cont.)
(Sheet 3 of 7)





(cont.)
Sheet 5 of 4

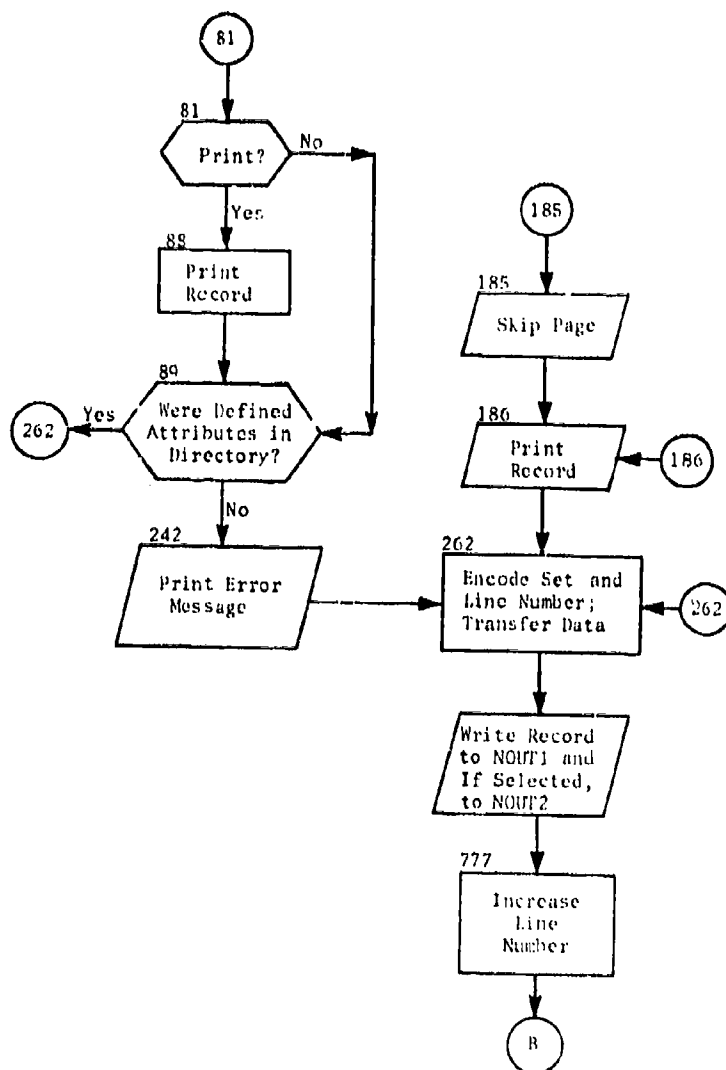


Fig. 97. (cont.)
(Sheet 6 of 7)

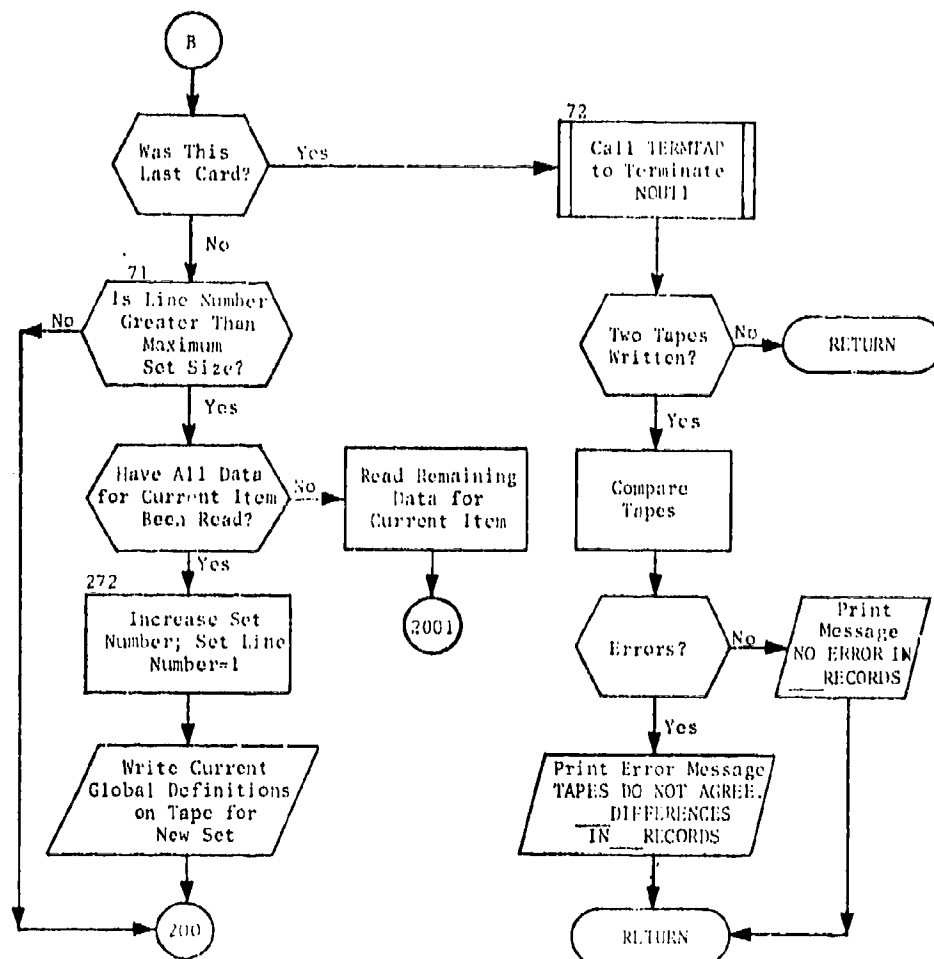


Fig. 97. (cont.)
(Sheet 7 of 7)

CHAPTER 6 PROGRAM BASEMOD

PURPOSE

The purpose of program BASEMOD is to alter the content or characteristics of a data base in order to adapt the data base to the specific scenario for which the plan is being developed. Because of its highly specialized nature, program BASEMOD should be examined for possible revision each time a new war plan is to be generated.

INPUT FILES

There are two distinct sets of input tapes which may be required by program BASEMOD -- one for post-QUIKBASE operation, and one for post-INDEXER operation.

When the program is executed in the former instance, the only required input tape is QUIKDB, the game base tape produced by program QUIKBASE. If, however, the user desires to exercise either the TARDEF or the ZONE option, a second tape is required. This tape is produced by program STACKER, a program of the NMCSSC QUICK Data Base Generator System (QDBGCS).

The input from the QUIKDB tape consists of the entire data base; the input from program STACKER is contained in two common blocks. Common /XLAT/ contains the necessary data for the introduction of TARDEFs, while common /MYZONES/ contains the data required for the determination of zones.

When program BASEMOD is executed post-INDEXER, the only required input tape is INDEXDB, the indexed data base produced by program INDEXER. Again, input from this tape consists of the entire data base.

OUTPUT FILES

In both post-QUIKBASE and post-INDEXER operation, there is one output tape produced by program BASEMOD. This tape contains the modified version of the game data base. In the former instance, this tape is QKMODDB, while in the latter instance, the tape is INMODDB.

CONCEPT OF OPERATION

The exact functions of program BASEMOD are directly related to the particular war plan being constructed; the program described herein is one which currently performs the modifications desired by the NMCSSC. Since program BASEMOD may be run either after program QUIKBASE or after program INDEXER, there are two distinct sets of program capabilities and user options. When BASEMOD is run after program QUIKBASE, the program performs such tasks as: removal of inappropriate targets (attribute RESERVE=0) from the game data base; establishing the number of aircraft per squadron NOPERSQN, number in commission NOINCOM, and number on alert NOALERT for each bomber and tanker unit; selection of the appropriate type name TYPE, value VAL, and relative effectiveness EFFECTNES for each fighter-interceptor squadron. Further, the user has available the options to specify, for urban/industrial targets, the attribute VAL as a function of either index of general industrial worth (IGIW) or population (POP); to calculate bomber local defense parameters (TARDEFs); and to calculate defensive zones for either or both sides.

When run after program INDEXER, BASEMOD provides the capability of selecting or deleting targets on the basis of geographic location (i.e., country location, CNTRYLOC).

IDENTIFICATION OF SUBROUTINE FUNCTIONS

Program BASEMOD is the main control routine. This subroutine reads the option card which specifies whether the run will be post-QUIKBASE or post-INDEXER and calls the appropriate subroutines.

Post-QUIKBASE Operation

Subroutine DBMOD is the controlling subroutine for this mode of operation. It calls subroutine RDTYPES to read in the values of the scaling factors to be used for the calculation of NOINCOM and NOALERT, and it reads in the remainder of the input parameters itself. If either the ZONE or TARDEF option is to be exercised, it calls subroutine STKRIN to read in the required data for the calculations. Subroutine DBMOD performs all of the required data base modifications itself, except for the determination of TARDEFs (if desired) which is done in subroutine TARDEFs, and the determination of ZONES (if desired) which is done in subroutine MYZONE. The record by region, type, and side of the targets deleted from the game base is kept by subroutine NUMDEL. The record by region, type, and side of the targets kept is maintained in subroutine COUNTDES.

Subroutine ADDVAL maintains a record of target value by class, type, and side and prints this summary after the processing is completed. Subroutine PRINTIT determines, on the basis of the user-specified parameter, whether a given item from the data base should be printed and, if so, prints it.

Post-INDEXER Operation

Subroutine INDMOD is the controlling subroutine for this mode of operation. It reads in all of the user parameters, and it performs the necessary data base modifications. It calls subroutine COUNTDES to maintain a record by region, type, and side of the targets kept, and it calls subroutine NUMDEL to maintain a summary by region, type, and side of the targets deleted. Subroutine PRTCOUNT is called to print the summaries kept in subroutine COUNTDES.

COMMON BLOCK DEFINITION

External Common Blocks

Program BASEMOD references the following utility routine common blocks, which are described in appendix A of this manual: /EDITAPE/, /EDITERM/, /IIP/, /MYIDENT/, /NOPRINT/, and /PROCESS/.

Internal Common Blocks

Table 7 lists the common blocks used within BASEMOD and identifies the arrays contained in them.

Table 7. Program BASEMOD Common Blocks
(Sheet 1 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
CUTIGIW	1	Dummy constant
	JCOUNTRY	List of countries to be assigned values of MINIGIW
	MINIGIW	List of minimum allowable values of the attri- bute IGIW
	NOIGIWS	Number of countries in the list for each side
IDESIGS	IDESIGS	First two letters of target designator code
	DESIGNO	Array containing summaries by region and type of items kept
JCARD	JCARD(1)	PRINT, if a print of items in the data base is desired; blank, otherwise
	JCARD(2)	Frequency of above print
	JCARD(3)	SELECT, if the items in the country list are to be kept; DELETE, if they are to be deleted
	JCARD(4)	Number of countries in the country list
JSIDE	JSIDE	Hollerith side name
LDESIGS	LDESIGS	First two letters of target designator code
	LDESIGNO	Array containing summaries by region and type of items omitted

Table 7. (cont.)
(Sheet 2 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
LODESIGS	LODESIGS	Number of different types of weapons deleted (for each side)
	LLMIN	Internal index parameter; =1 for BLUE, =251 for RED
MYSIDE	MYSIDE	Current side
MYZONES		Zone data from program STACKER
	BLAT	Latitude associated with the point of origin of a leg
	BLONG	Longitude associated with the point of origin of a leg
	1ZIT	The BLEGNO associated with the last IPOINT which describes a zone
	ILINK	Value of LINK associated with BLEGNO
	MINBLUE	Minimum BLUE zone index number
	MAXBLUE	Maximum BLUE zone index number
	MINRED	Minimum RED zone index number
	MAXRED	Maximum RED zone index number
	MINTEST	Internal BLEGNO index parameter
	JLINK	Internal LINK index parameter
	NTEST	Internal counter
	MIN	Internal BLEGNO index parameter

Table 7. (cont.)
(Sheet 3 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
NODESIGS	NODESIGS	Number of different types of weapons (for each side)
	KKMIN	Internal index parameter; =1 for BLUE, =251 for RED
NRYPES	NRYPES	Number of weapon types for current side for which NOALERT and NOINCOM are to be scaled
	MTYPES	Same as NRYPES, but in alphameric format
	NNTYPES	Type name of each weapon type for which NOALERT and NOINCOM are to be computed
	ALERTNO	NOALERT scaling factor for each weapon type
	COMINNO	NOINCOM scaling factor for each weapon type
PRINTS	IFREQ	Desired frequency of prints
	IPRT	Index used to count number of items processed between successive prints
	IPRINT	=1 if prints are desired; =0 if not
TYPENAME	INDBEG	Smallest index number for each type
	TYPENAME	Type names in order of increasing index number
	CUMNO	Cumulative number of types in each class
	BTYPES	Number of BLUE side types in each class
	INDCLAS	Smallest index number in each class

Table 7. (cont.)
(Sheet 4 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
XLAT		SAM data from program STACKER
	XLAT	Array containing the latitude of the centroids of each SAM site complex (locations 1 through 500 store BLUE data; 501 through 1,000 store RED data)
	XLONG	Array containing the longitude of the centroids of each SAM site complex (same storage as above)
	RADIUS	The defensive radius of each SAM site complex
	NUMBATTs	Number of SAMs located within the radius of the complex
	JINDEX	Index of the first SAM site in each major geographical area
	LINDEX	Index of the last SAM site in each major geographical area
	ATEST	Values of longitude which subdivide the complexes into reasonably well defined sites
	NBAREAS	Not used
	NRAREAS	Not used
	NTARSHI	TARDEF number which is associated with a high-altitude defense of a given strength
	NTARSLO	TARDEF number which is associated with a low-altitude defense of a given strength
	NTARTEST	Contains values which divide the total number of SAMs into several distinct ranges

Table 7. (cont.)
(Sheet 5 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
XLAT (cont.)		
	XTEST	Factor used to determine if a SAM complex can provide high-altitude defense to a target
	MAXHI	The maximum TARDEF which can be assigned for high-altitude defenses
	MAXLO	The maximum TARDEF which can be assigned for low-altitude defenses
	FACLOW	Factor used to determine whether a value of TARDEFLO should be assigned to a given target
	LAREAS	Number of geographical areas into which SAM sites are divided
	LNLOW	The beginning indices of the areas for BLUE and RED (1 and 101, respectively)
	LOW	Lower index indicating where storage of data begins for BLUE and RED, respectively, in NTARSHI, NTARSLO, and NTARTEST
	THIGH	Index indicating where the storage of data ends for BLUE and RED (7 and 17, respectively) in the abovementioned arrays
	UTARTAPE	Not used
	JJLOW	Beginning indices of SAM complexes for BLUE and RED (1 and 501, respectively)
	JAREAS	Number of areas in which SAM complexes are divided
	JLOCS	Number of SAM complexes for each side

PROGRAM BASEMOD

PURPOSE: To determine whether the program is being run post-QUIKBASE or post-INDEXER, and to transfer control to the appropriate subroutine.

ENTRY POINTS: BASEMOD

FORMAL PARAMETERS: None

COMMON BLOCKS: None

SUBROUTINES CALLED: DBMOD, INDMOD

CALLED BY: None

Method

The variable IALT is read from an input data card. If IALT is zero or negative, the run is post-QUIKBASE and subroutine DBMOD is called. If IALT is equal to one or more, the run is post-INDEXER and subroutine INDMOD is called. After the subroutine returns the program stops.

Program BASEMOD is illustrated in figure 98.

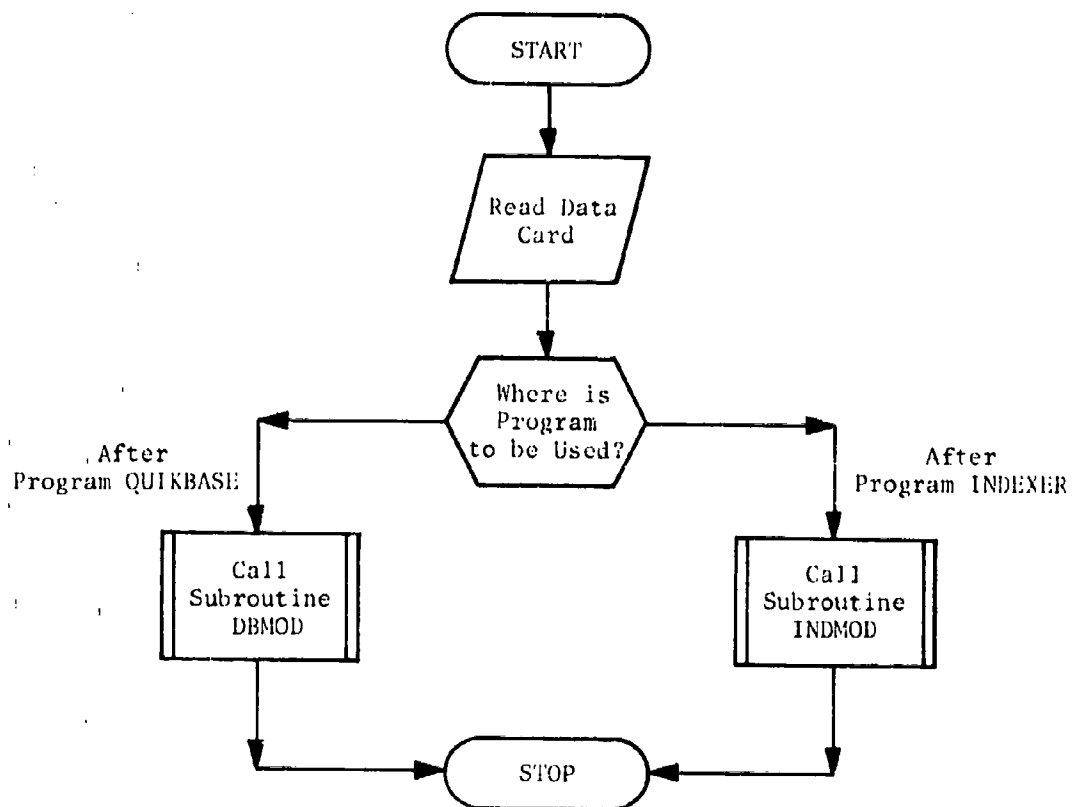


Fig. 98. Program BASEMOD

SUBROUTINE ADDVAL

PURPOSE: To accumulate target value by class, type, and side, and to print the information in tabulated form.

ENTRY POINTS: ADEVAL, PRNTVAL

FORMAL PARAMETERS: IS - Value of attribute SIDE
IC - Value of attribute ICLASS
IT - Value of attribute TYPE
V - Value of attribute VAL
JC - Value of attribute CLASS

COMMON BLOCKS: None

SUBROUTINES CALLED: None

CALLED BY: DBMOD

Method

Each time the subroutine is entered through the entry point ADDVAL, the value of the item being considered is added to the total value of the other items of the same class, type, and side which have already been considered. In addition, a count of the total number of items kept in each of the categories (various combinations of class, type, and side) is maintained. If the item under consideration is the first with its class, type, and side characteristics, a new category is created for it.

When the subroutine is entered through the entry point PRNTVAL, the summaries which have been maintained are printed out.

Subroutine ADDVAL is illustrated in figure 99.

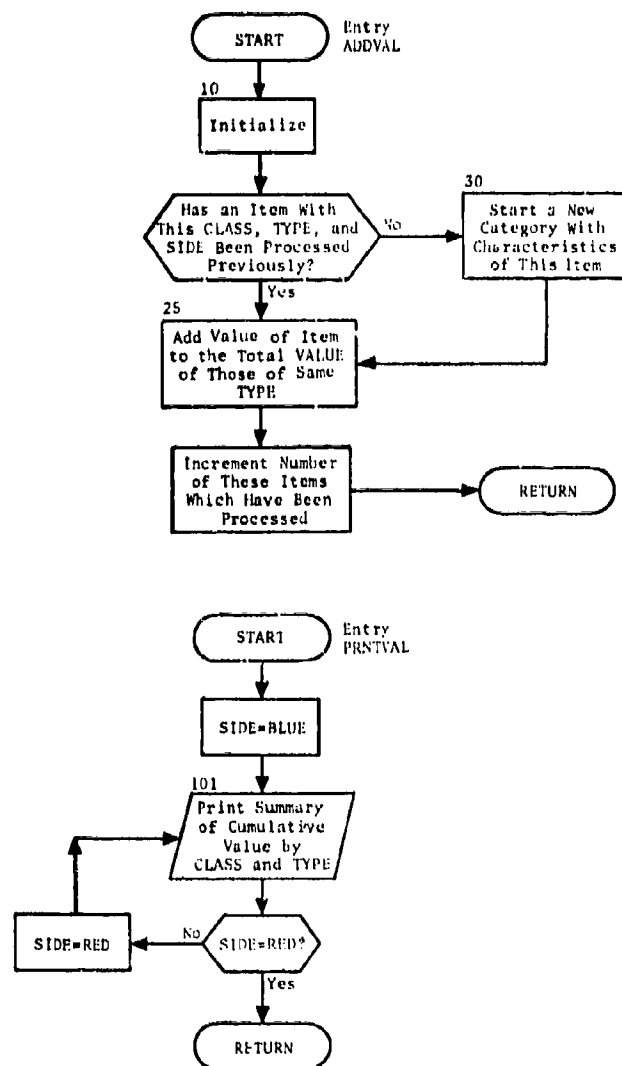


Fig. 99. Subroutine ADDVAL

SUBROUTINE COUNTDES

PURPOSE: To keep a tally by region and type of the targets kept for each side after processing by subroutine DBMOD.

ENTRY POINTS: COUNTDES

FORMAL PARAMETERS: II - Equals 1 if side is BLUE; 2 if side is RED
MYDESIG - Designator code of item
IREG - Region in which item is located

COMMON BLOCKS: IDESIGS, NODESIGS

SUBROUTINES CALLED: None

CALLED BY: DBMOD, INDMOD

Method

Each time the subroutine is entered, the total number of items retained with the same type, region, and side is incremented by one. The region is determined from the target designation code. Subroutine COUNTDES is illustrated in figure 100.

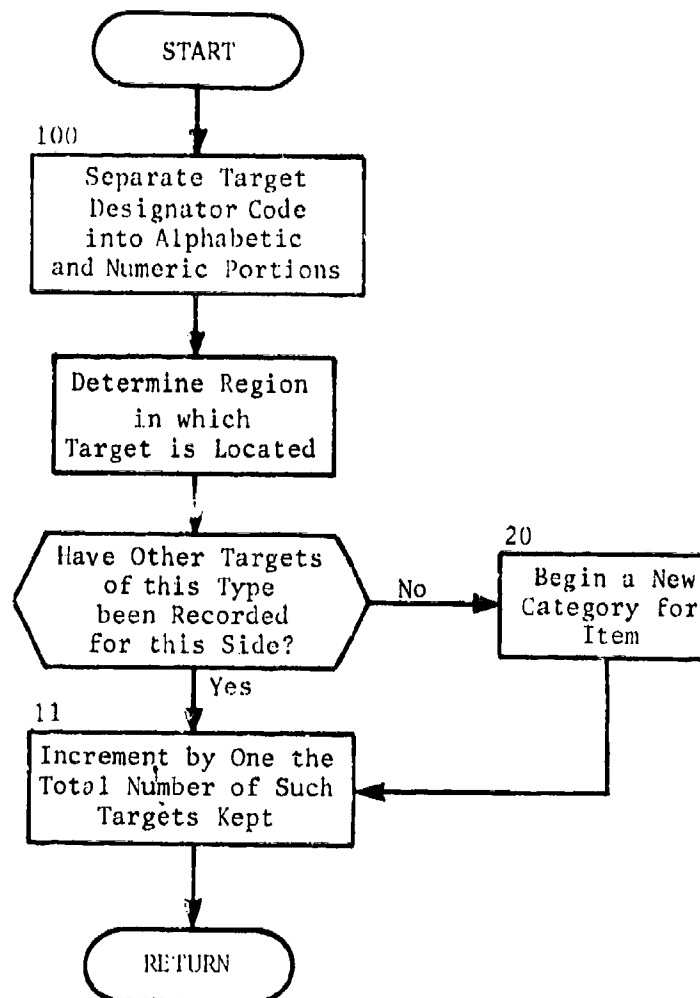


Fig. 100. Subroutine COUNTDES

SUBROUTINE DBMOD

PURPOSE: To control the information processing when program BASEMOD is run post-QUIKBASE.

ENTRY POINTS: DBMOD

FORMAL PARAMETERS: None

COMMON BLOCKS: CUTTGIW, EDITAPE, EDITERM, IDESIGS, ITP, LDESIGS, LODESIGS, MYIDENT, MYSIDE, NODESIGS, NOPRINT, NRYPES, PRINTS, PROCESS

SUBROUTINES CALLED: ADDVAL, COUNTDES, INITAPE*, INITEDIT, INPITEM, MYZONE, NEXTITEM, NUMDEL, NUMGET, OUTITEM, PAGESKP, PRINTIT, PRNTVAL, RDYPES, STKRIN, TARDEFS

CALLED BY: BASEMOD

Method

Subroutine DBMOD effects a sequential examination of each item in the game data base (contained on the QUIKDB tape). Each item is read in, filtered through a series of tests, and assigned appropriate values for certain of its attributes. The item is then either retained or deleted from the game data base. The 12 tasks accomplished by this processing are:

1. Targets which are inappropriate for the plan under consideration; i.e., those targets assigned the attribute RESERVE=0, are excluded from further consideration.
2. The appropriate number of bombers or tankers for each bomber or tanker squadron (NOPERSQN) is selected, depending upon the particular plan being developed (Initiative, Surprise, or Retaliatory).
3. The number of bombers or tankers in commission (NOINCOM) for each bomber or tanker squadron is calculated by specifying that NOINCOM is equal to a user-specified fraction of NOPERSQN.

*See subroutine FILEINR.

4. The number of bombers or tankers which are on alert (NOALERT) for each squadron is calculated by specifying that NOALERT is equal to a user-specified fraction of NOINCOM.
5. The appropriate value of the attributes TYPE, VAL, and EFFECTNES is established for each fighter interceptor unit based on the user-input parameter POSTURE. If POSTURE=1, these attributes are assigned the values of the attributes TYPE1, VAL1, and EFFECTNES1, respectively. If POSTURE=2, the values of the attributes TYPE 2, VAL2, and EFFECTNES2 are assigned.
6. The relative value (VAL) of urban/industrial targets is calculated as a function of either general industrial worth (IGIW) or population (POP).
7. If the TARDEF option is exercised, each target (opposing side) is processed and the level of local bomber defense available at the target is calculated.
8. If the ZONE option is exercised, items in ICLASS 4 and 5 (defensive command and control sites and interceptor bases, respectively) are processed to determine the air defense zone in which the item is located.
9. The value of the attribute IREG is determined based on the target designator code DESIG assigned to the item.
10. Each Blue (SIDE=BLUE) installation is assigned a value for the attribute FLAG. The assigned value (numeric code 1 through 9 established based on ICLASS) is subsequently used in program ALOC to impose user-restrictions on the allocation of weapons (see Program ALOC, User-input Parameters, FLAGREST Function -- Restriction of Weapons Using FLAG Attribute in chapter 3 of User's Manual, Volume 11).
11. Targets may be deleted from the game base on the basis of TASK or MINIGIW (user-specified parameter which establishes the minimum index of general industrial worth to be considered).
12. The appropriate value of the attribute DBL (probability of destruction before launch) is chosen.

Subroutine DBMOD is illustrated in figure 101.

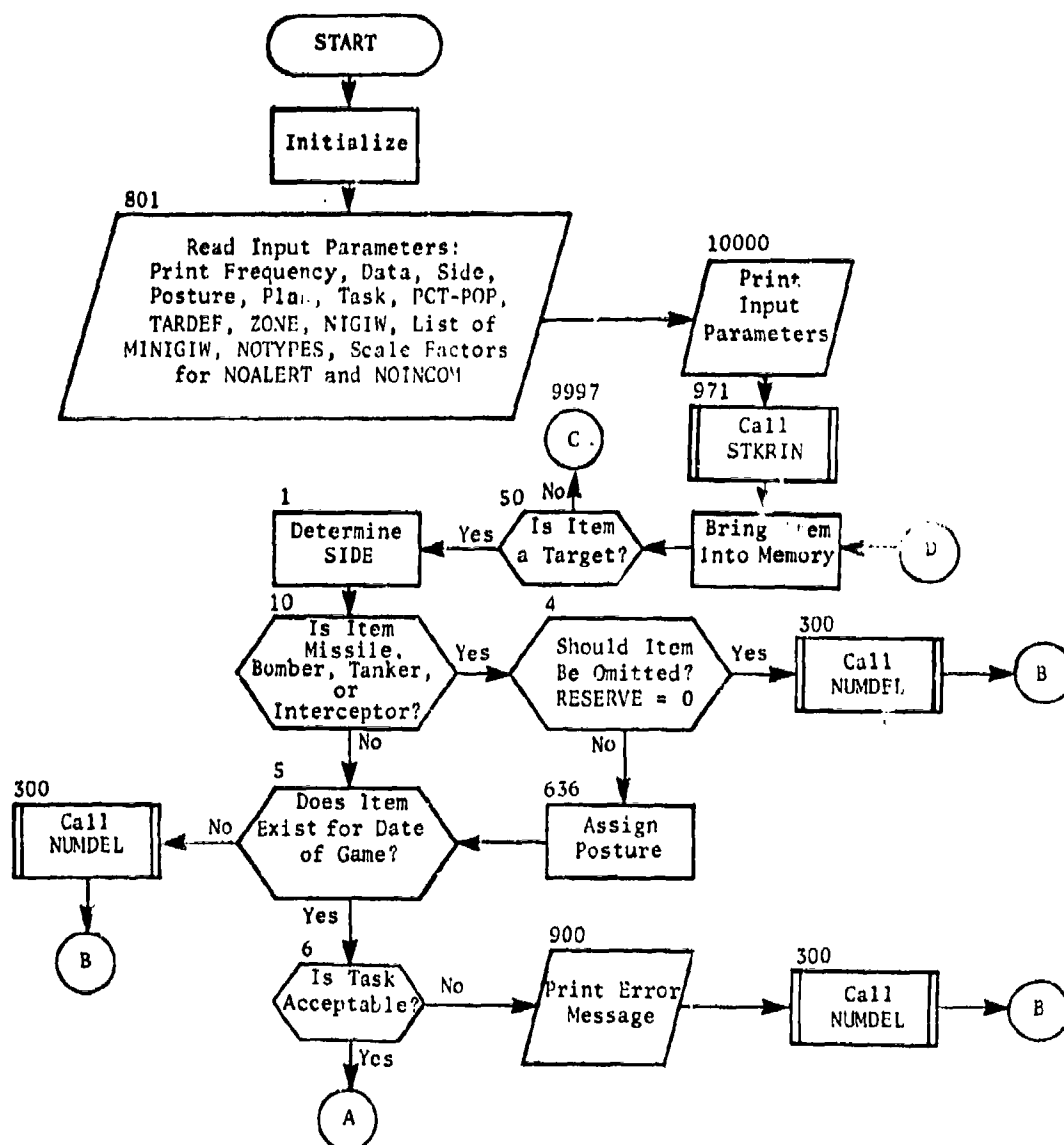


Fig. 101. Subroutine DBMOD
(Sheet 1 of 2)

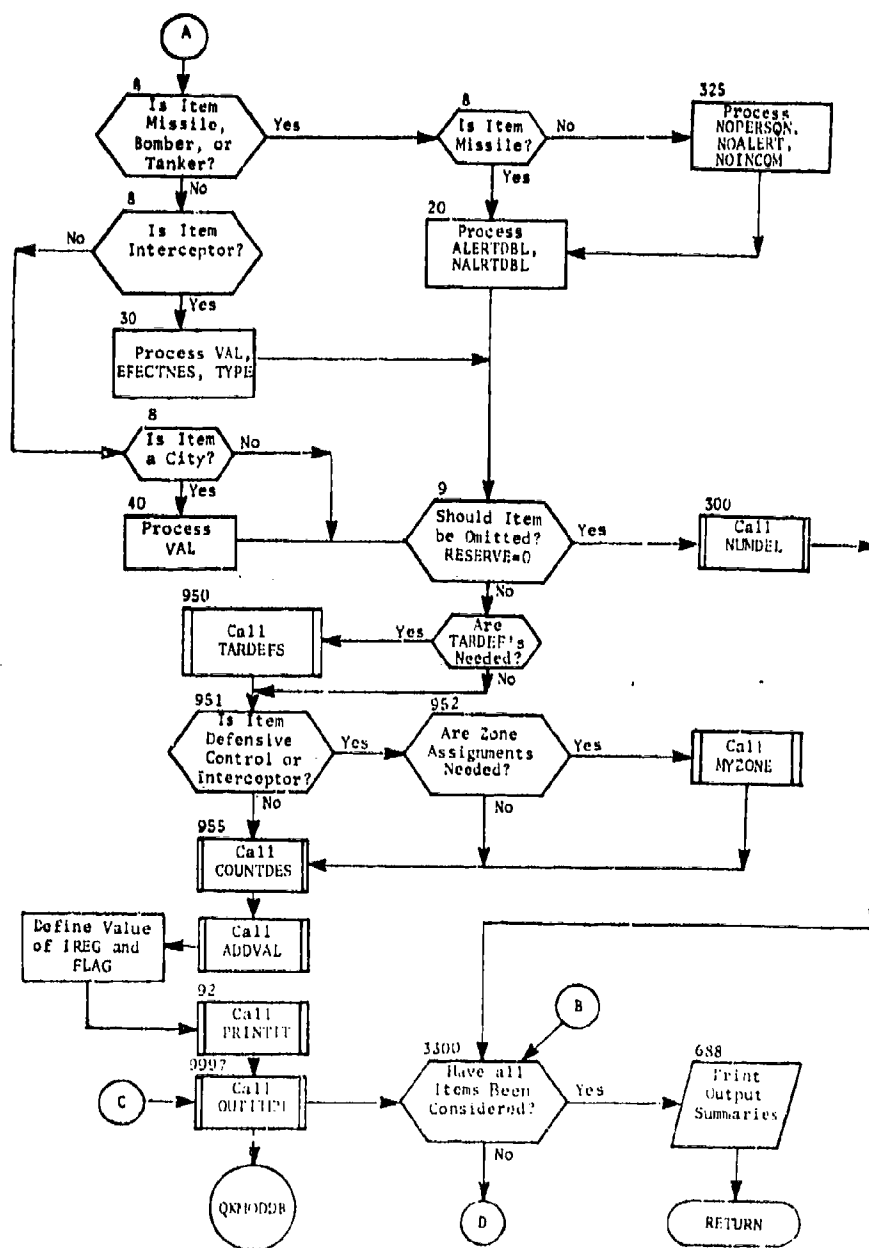


Fig. 101. (cont.)
(Sheet 2 of 2)

FUNCTION INDEXTYP

PURPOSE: To compare the value of NUMBATTs with the value of NTARTEST until the region in which the value of NUMBATTs lies is ascertained.

ENTRY POINTS: INDEXTYP

FORMAL PARAMETERS: NUMBATTs - Number of SAMs located within the radius of the complex being considered
ISIDE - Hollerith value of side

COMMON BLOCKS: JSIDE, XLAT

SUBROUTINES CALLED: None

CALLED BY: TARDEFs

Method

This function subroutine is entered with a given value for the parameter NUMBATTs. The range of allowable values of NUMBATTs has been divided into (IHIGH-ILOW+1) equal segments, and the lower and upper limits of each of the segments have been stored in the array NTARTEST (IHIGH and ILOW are externally specified index parameters). A search is then implemented to determine the segment in which the value of NUMBATTs under consideration lies. This segment is characterized by the index number of the upper limit of the segment (e.g., if NUMBATTs lies between NTARTEST(3) and NTARTEST(4), the value of INDEXTYP for NUMBATTs will be 4).

Function INDEXTYP is illustrated in figure 102.

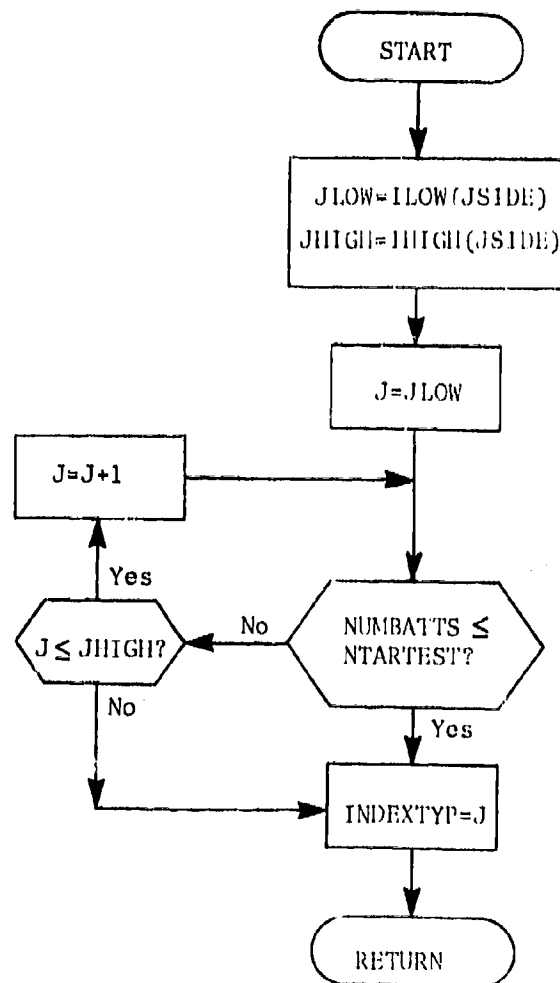


Fig. 102. Function INDEXTYP

SUBROUTINE INDMOD

PURPOSE: To control the information processing when program BASEMOD is run post-INDEXER.

ENTRY POINTS: INDMOD

FORMAL PARAMETERS: None

COMMON BLOCKS: EDITAPE, EDITERM, IDESIGS, ITP, JCARD, LEVSIGS, LODESIGS, MYTEXT, NOBSIGS, NOPRINT, PROCL, TYPENAME

SUBROUTINES CALLED: ABORT, COUNTPLS, INFLATE*, INFLDIT, INPUTEM, NEXTITEM, NUMDEL, NUMGET, OUTITEM, PAGESEP, PRITEM, PRPCOUNT, STORAGE, TERM*AP*

CALLED BY: BASEMOD

Method

INDMOD begins by reading the input parameters which include a list of country codes to be used in selecting or deleting targets on the basis of geographical location. The program then performs an item-by-item examination of the game data base as contained on the INDEADB tape and retains or deletes items in accordance with the input criteria. Items selected for retention are output to the INMODEL tape.

Subroutine INDMOD is illustrated in figure 102.

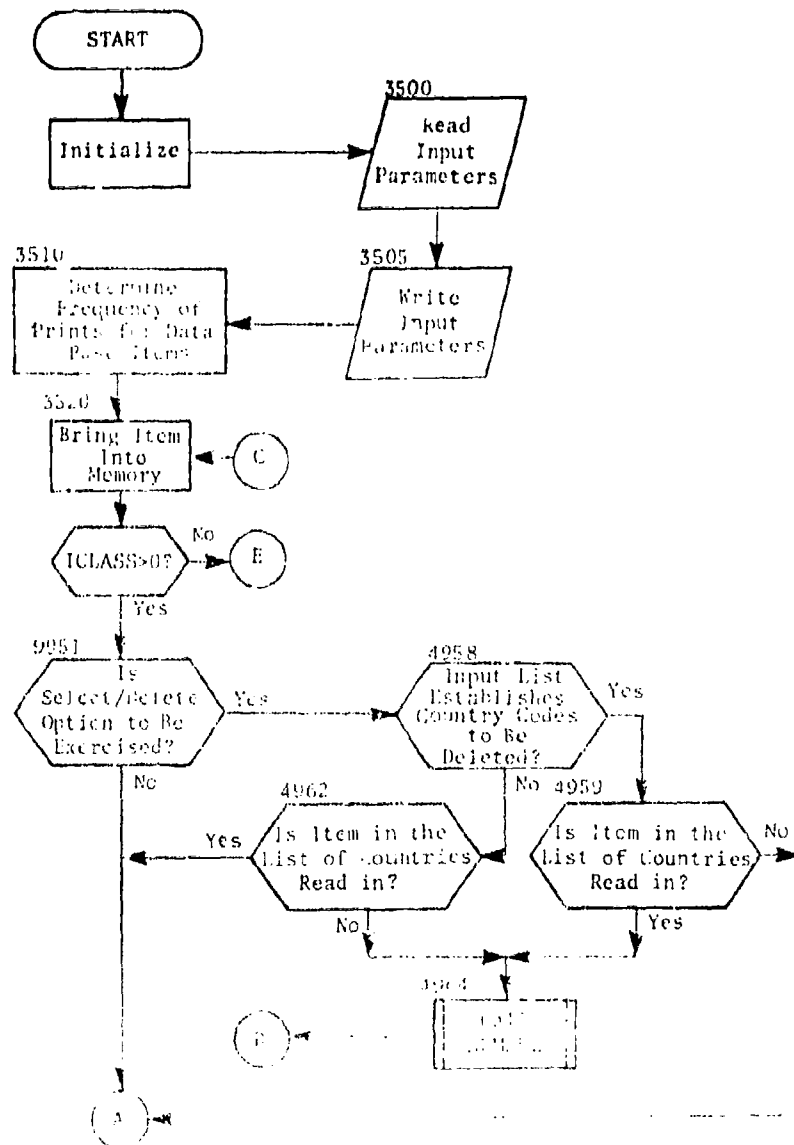


FIG. 10. Flowchart of the Data Base

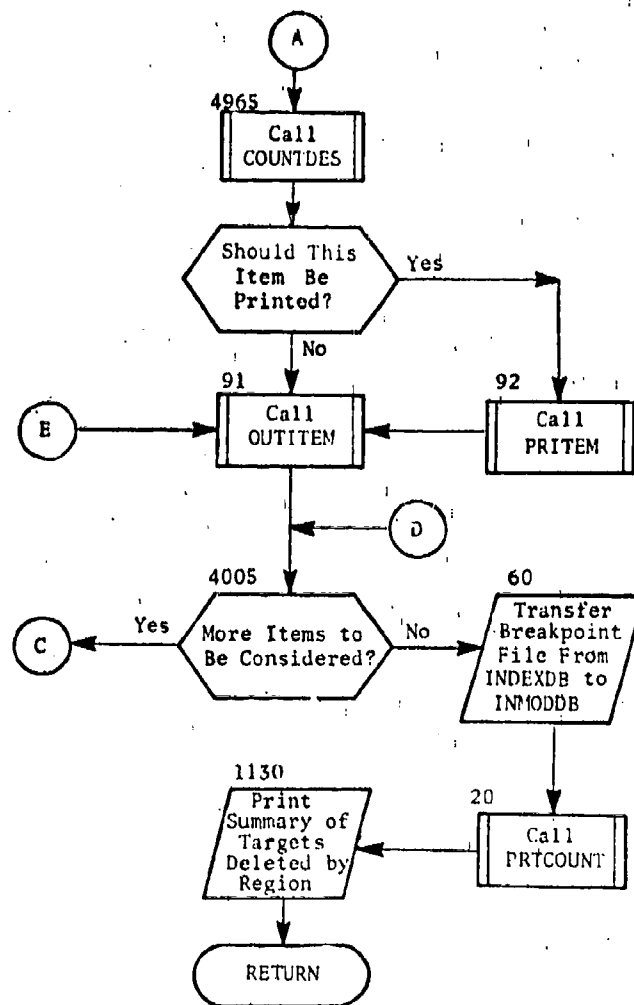


Fig. 103. (cont.)
(Sheet 2 of 2)

FUNCTION MYZONE

PURPOSE: To determine in which defensive zone a target lie.

ENTRY POINTS: MYZONE

FORMAL PARAMETERS: ZLAT - Latitude of item
ZLONG - Longitude of item

COMMON BLOCKS: MYSTDE, MYZONES

SUBROUTINES CALLED: DIFFLONG

CALLED BY: DBMOD

Method

The subroutine considers each defensive zone for the side corresponding to that of the target, and determines the sum of all angles formed by the lines connecting the target point to two adjacent boundary points of the zone. If the target is in the zone, or on a boundary, the sum will be 360° , and if the target is outside the zone, the sum will be 0° . As soon as a target is found to be within a zone, it is identified as belonging to that zone by assigning it the zone index. Thus if the target is on the boundary between two zones it will be assigned the smaller zone index. If the target is not in a zone of its side it is assigned a zone index at zero.

Function MYZONE is illustrated in figure 404.

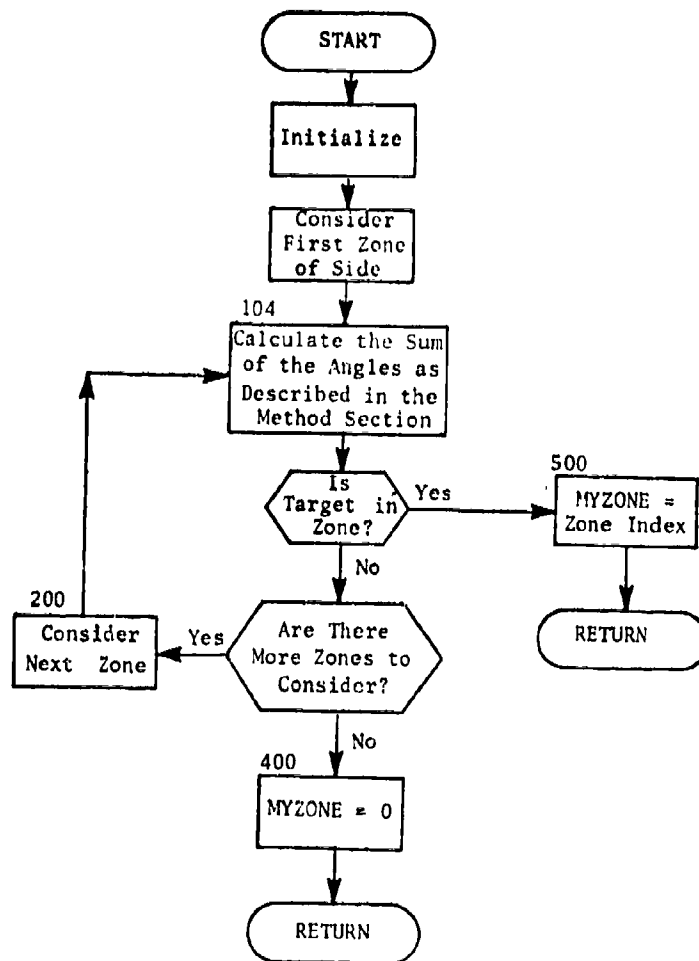


Fig. 104. Function MYZONE

SUBROUTINE NUMDEL

PURPOSE: To keep a tally by region and type of the targets which have been deleted for each side after processing by subroutine DBMOD.

ENTRY POINTS: NUMDEL

FORMAL PARAMETERS: II - Equals 1 if side is BLUE; 2 if side is RED
MYDESIG - Target designator code of item
IREG - Region in which item is located

COMMON BLOCKS: LDESIGS, LODESIGS

SUBROUTINES CALLED: None

CALLED BY: DBMOD, INDMOD

Method

Each time the subroutine is entered, the total of the number of items deleted with the same type, region, and side as the one under consideration is incremented by one. If the item under consideration is the first one with its particular characteristics, a new category is created for it.

Subroutine NUMDEL is illustrated in figure 105.

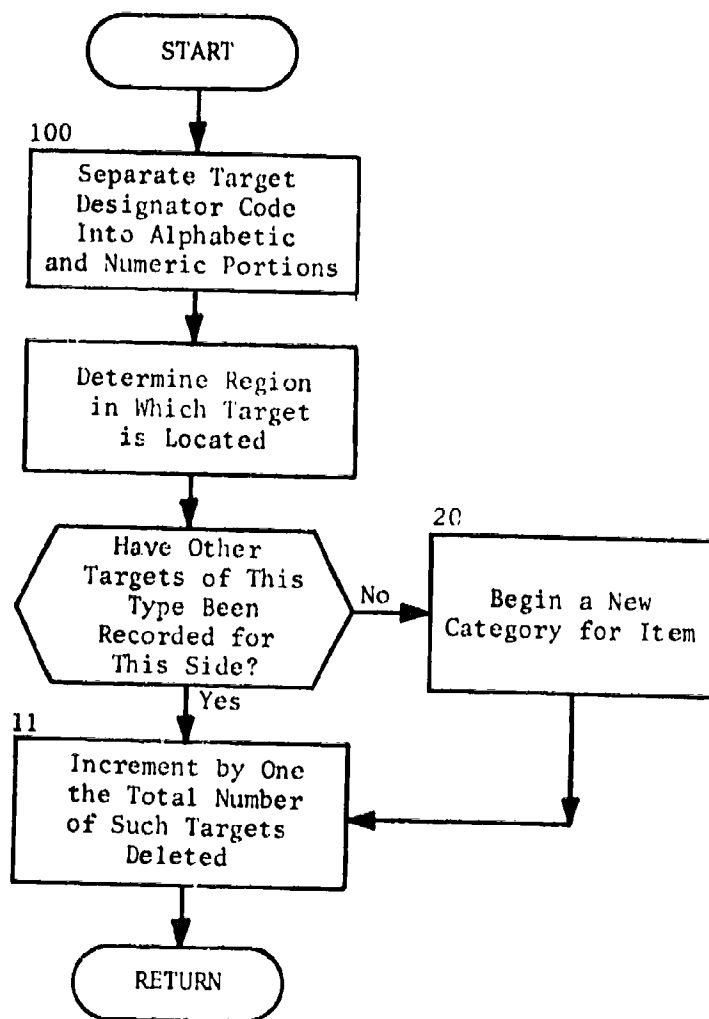


Fig. 105. Subroutine NUMDEL

SUBROUTINE PRINTIT

<u>PURPOSE:</u>	To determine whether the item being processed by subroutine DBMOD should be printed and, if so, prints it.
<u>ENTRY POINTS:</u>	PRINTIT
<u>FORMAL PARAMETERS:</u>	None
<u>COMMON BLOCKS:</u>	PRINTS
<u>SUBROUTINES CALLED:</u>	PRITEM
<u>CALLED BY:</u>	DBMOD

Method

This subroutine maintains a record of the number of items processed and kept by subroutine DBMOD. Each time it is entered, a check is made to determine whether the current item being processed should be printed. If it is to be printed, subroutine PRITEM is called to print the attribute-value pairs for the item.

Subroutine PRINTIT is illustrated in figure 106.

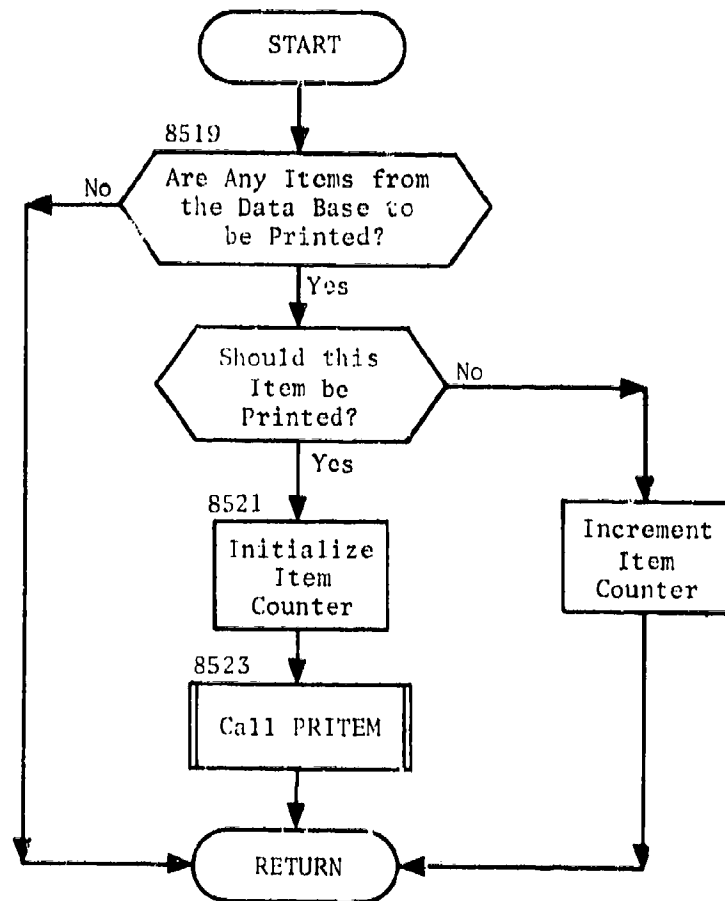


Fig. 106. Subroutine PRINTIT

SUBROUTINE PRTCOUNT

PURPOSE: To effect the printing of the records of target count by region which were kept by subroutine COUNTDES for the targets processed and kept by subroutine INDMOD.

ENTRY POINTS: PRTCOUNT

FORMAL PARAMETERS: None

COMMON BLOCKS: IDESIGS, NODESIGS

SUBROUTINES CALLED: PAGESKP

CALLED BY: INDMOD

Method

A print for each side is made of the targets kept in the data base, by type and region. Also, the total number of targets in each region is presented.

Subroutine PRTCOUNT is illustrated in figure 107.

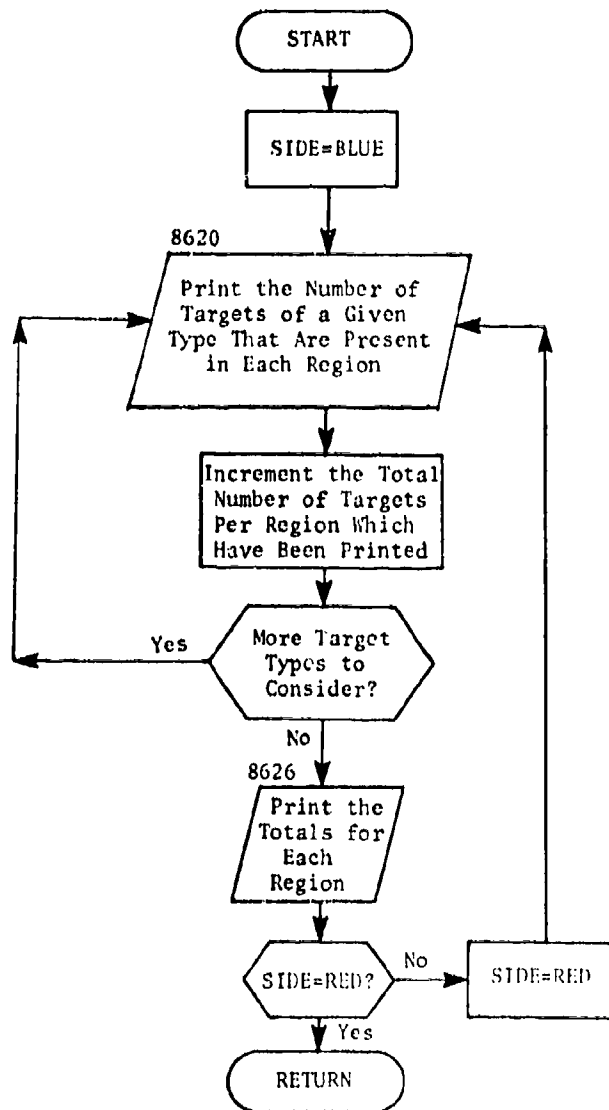


Fig. 107. Subroutine PRTCOUNT

SUBROUTINE RDTYPES

PURPOSE: To read in the values of the scaling factors to be used for the calculation of NOINCOM and NOALERT.

ENTRY POINTS: RDTYPES

FORMAL PARAMETERS: LSIDE - Hollerith value of side

COMMON BLOCKS: NRTYPES

SUBROUTINES CALLED: NUMGET

CALLED BY: DBMOD

Method

This subroutine is entered once to read the data for the BLUE side and a second time to read the data for the RED side. When it is entered, a card is read which indicates how many cards with scaling factors are to be read, and these cards are then read. The number of weapon types to be scaled, the scaling factor for NOALERT, and the scaling factor for NOINCOM for the BLUE side are stored in locations 1 through 50 of the arrays NNTYPES, ALERTNO, and COMINNO, while the data for the RED side are stored in locations 51 through 100.

Subroutine RDTYPES is illustrated in figure 108.

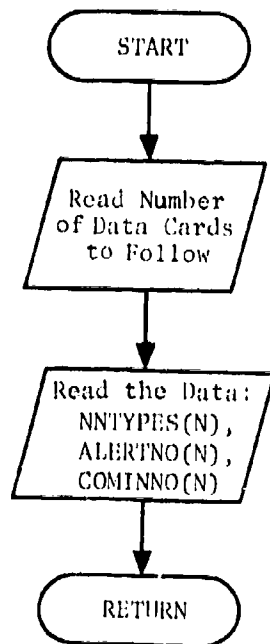


Fig. 108. Subroutine RDTYPES

SUBROUTINE STKRIN

PURPOSE: To read in the arrays in common blocks /XLAT/ and /MYZONES/ from the output tape created by program STACKER and to assign initial values to certain variables.

ENTRY POINTS: STKRIN

FORMAL PARAMETERS: None

COMMON BLOCKS: ITP, JSIDE, MYIDENT, MYZONES, NOPRINT, XLAT

SUBROUTINES CALLED: RDARRAY*, SETREAD, TERMTAP*

CALLED BY: DBMOD

Method

This subroutine reads in the necessary data from the tape created by program STACKER to enable the calculation of TARDEFs and the determination of ZONES. The data used for the former task are contained in common block /XLAT/, while that required for the latter task are contained in common block /MYZONES/.

Subroutine STKRIN is illustrated in figure 109.

* See subroutine FILEINR.

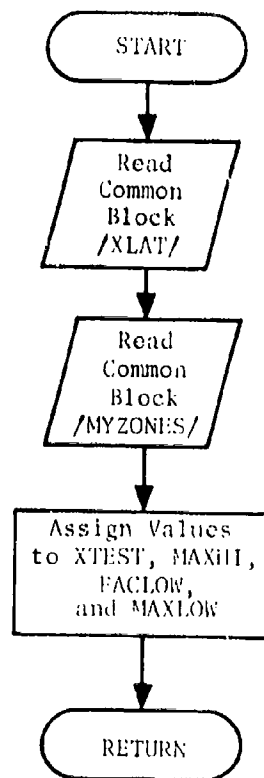


Fig. 109. Subroutine STKRIN

SUBROUTINE TARDEFS

PURPOSE: To assign values of JTARHI and JTARLO to characterize a given target. These parameters describe the amount of high-altitude and low-altitude terminal SAM (surface-to-air missile) defenses that are available to defend the target.

ENTRY POINTS: TARDEFS

FORMAL PARAMETERS: JTARHI - Parameter characterizing the level of high-altitude local bomber defenses
JTARLO - Parameter characterizing the level of low-altitude local bomber defenses
YLAT - Latitude of target being considered
YLONG - Longitude of target being considered
ISIDE - Hollerith value of side

COMMON BLOCKS: JSIDE, XLAT

SUBROUTINES CALLED: DIFFLONG, DSTF, INDEXTYP

CALLED BY: DBMOD

Method

Within QUICK, the attributes TARDEHI and TARDELO are defined for each potential target. The value assigned these attributes represents the level of local bomber defenses available to defend the target against attacks expected at high altitude and at low altitude, respectively. Normally, these attribute-value pairs are defined by NMCSSC with the use of programs which are external to the QUICK system; however, program BASEMOD provides an alternate method for assigning these values. When this option is exercised, subroutine TARDEFS is called for each target to compute and to return to the calling subroutine the values of JTARHI and JTARLO, the values that will be assigned to TARDEHI and TARDELO, respectively. To perform this task, the location of the target is considered, and the number of SAM batteries (NBATTS) capable of providing high and low altitude defense for the target are determined. This information is then used for the calculation of the values of JTARHI and JTARLO. The data describing the SAM defenses which are stored in common block /XLAT/ are developed in program KRUNCH, a program external to the QUICK system. If the TARDEFS option is to be exercised, these data must be provided as input to program BASEMOD.

Program KRUNCH considers all SAM sites within the target area and groups these sites in circular complexes on the basis of the effective range of the SAM sites. Each SAM complex is characterized by the latitude and longitude of its centroid, its radius, and the number of SAM batteries (NBATTS) associated with the complex. Each SAM site is included in one, and only one, complex. KRUNCH orders and indexes the SAM complexes according to increasing longitude, and then assigns to each complex a sector index. These sectors are established to facilitate subsequent processing and are formed through a longitudinal division of the land area under consideration. When called, subroutine TARDEFS utilizes these data and determines first the area and then the complexes in which the given target lies. On the basis of this information, values of JTARII and JTARLO, parameters which reflect the amounts of available local bomber defense, are assigned.

Subroutine TARDEFS is illustrated in figure 110.

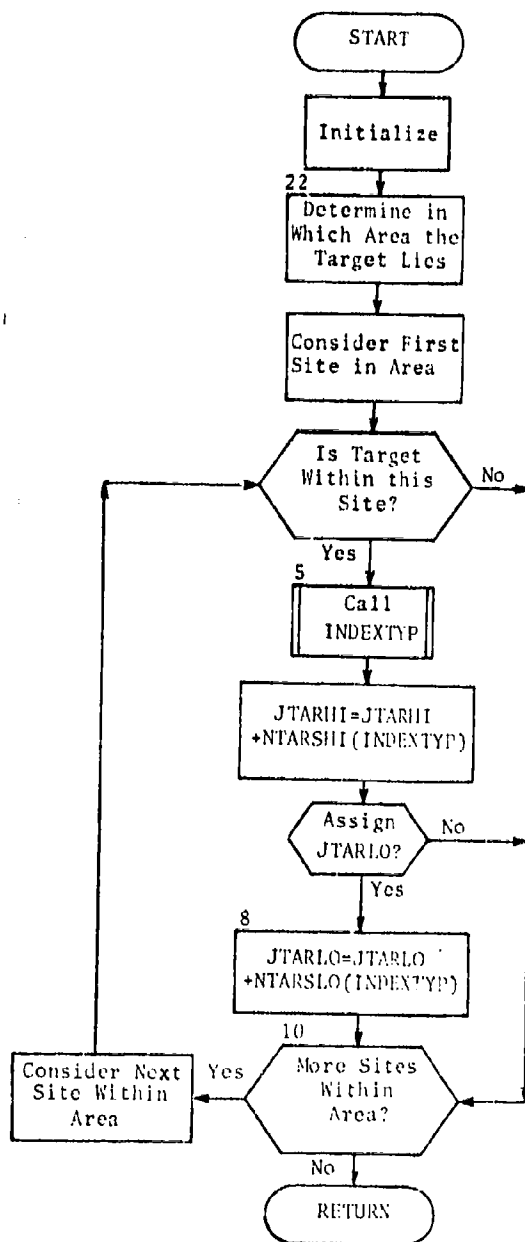


Fig. 110. Subroutine TARDEF5

CHAPTER 7 PROGRAM INDEXER

PURPOSE

To provide for economical handling of data and to facilitate communications between QUICK programs, it is necessary to assign indices to various data contained in the data base. Program INDEXER is designed to perform this task. In addition, INDEXER processes all potential targets and, where appropriate, forms them into collocation islands and complex targets. Having completed the required processing, INDEXER prepares the indexed data base tape INDEXDB and the simulation data tape SIMTAPE.

INPUT

The input to program INDEXER consists of the user-input parameters and a data base file. The user-input parameters identify the print options to be exercised, and provide parameters used in blast damage calculations. The required user-input parameters and a description of the output prints provided by INDEXER are presented in chapter 3 of the User's Manual, Volume I, and in chapter 2 of the User's Manual, Volume II.

The data base is input to program INDEXER via magnetic tape. This source file may be either the QUICKDB tape created by program QUICKBASE or the QKMODDB tape prepared by program BASEMOD. This is not to imply that either source file is always acceptable. If the data base tape QUICKDB was processed by program BASEMOD to adapt it to the game scenario, the BASEMOD output tape QKMODDB is input to INDEXER.

OUTPUT

Program INDEXER prepares two output files: an indexed data base tape INDEXDB, and a simulation data tape SIMTAPE. The INDEXDB tape is used as input to program BASEMOD (if used in the post-INDEXER mode) or as input to the Plan Generation and Data Output subsystems. The INDEXDB tape is

prepared using the same logical record format as the input data base tape (QUICKDB or QKMODDB) except that an index breakpoint table is added to the file*. Table 8 shows the logical record format of this file. The SIMTAPE includes selected weapon and target data and is prepared for use in program SIMULATE. The SIMTAPE format is shown in table 9. (The SIMTAPE is produced by the QUICK system filehandler. The physical format of the tape is described in Chapter 2, QUICK System Filehandler.)

CONCEPT OF OPERATION

General

The data base which is input to INDEXER contains those items which are to be considered in a specific game scenario. The information included in the data base is categorized by CLASS; e.g., bomber, and by TYPE within class; e.g., B-52. Fifteen classes are used to describe the targetable-type installations included in the data base (see Data Base Organization, chapter 2, Analytical Manual, Volume I). To facilitate subsequent processing, program INDEXER assigns various indices to these data items.

When the data base is prepared, each of the target classes is assigned a value, from 1 to 15, for the attribute ICLASS**. During INDEXER processing, all target types which belong to these indexed classes are assigned distinct values of the attribute ITYPE, and all types within each class are assigned distinct values of the attribute JTYPE, maintaining the order established by the ITYPE assignment. In addition, each of these data items is assigned a unique value of the attribute INDEXNO (index number). The order of indexing is as follows: first, all items of the same class are numbered consecutively. Within a single class, items are grouped according to the attribute SIDE (value RED or BLUE). Items are further grouped according to type (attribute TYPE). Within a type, items are assigned index numbers according to the order in which they appear in the data base.

After these indices have been assigned, collocation islands and complex targets are made up from the collection of all potential targets (items for which INDEXNO is defined). Collocation islands are defined by the

* The data base terminator block is followed by a padding record and an end-of-file mark placed on the tape by subroutine TERMTEAP (see Chapter 2, QUICK System Filehandler). The single breakpoint table record as displayed in table 8 follows the terminating end of file written by TERMTEAP.

** See appendix B for attribute definitions.

Table 8. Indexed Data Base File (Logical Record Format)
(Sheet 1 of 2)

<u>BLOCK TYPE</u>	<u>ARRAY</u>	<u>LENGTH</u>	<u>DESCRIPTION</u>
Directory	IDEF	1	Number of attributes
	LASTLIST	1	Number of entries in value list
	ATTNAME	IDEF	Attribute names
	IFORMAT	IDEF	Hollerith format codes
	ICODE	IDEF	Error checking code
	DEFAULT	IDEF	Attribute default value
	N1	IDEF	Minimum allowable attribute value
	N2	IDEF	Maximum allowable attribute value
	LISTCHK	IDEF	Logical array to specify list checking
	LGLOB	IDEF	Logical array to specify global definitions
	LISTVALS	LASTLIST	List of values to be checked
Item	NI	1	Number of attributes defined locally for this item
	INITEM	2*NI	Array containing attribute index (from directory) in odd elements and attribute value in even elements
Define	NI	1	= -1 as DEFINE block indicator
	L	1	Attribute index from directory
	VALUE	1	Attribute value
Undefine	NI	1	= -2 as UNDEFINE block indicator
	L	1	Attribute index from directory
	VALUE	1	New attribute value
Terminator	NI	1	= ENDDATA as terminator block indicator

Table 8. (cont.)
(Sheet 2 of 2)

<u>BLOCK TYPE</u>	<u>ARRAY</u>	<u>LENGTH</u>	<u>DESCRIPTION</u>
Breakpoint Tables	CUMNO(I)	15	Total number of types within each class (I = 1 to 15)
	BTYPES(I)	15	Number of BLUE types in class I
	INDCLAS(I)	15	Beginning index number INDEXNO for class I
	INDBEG(J)	250	Beginning index number INDEXNO for type J
	TYPENAME(J)	250	Type name for each type J

Table 9. SIMTAPE Format
(Sheet 1 of 4)

<u>VARIABLE NAME</u>	<u>MAXIMUM LENGTH/ ACTUAL LENGTH</u>	<u>DESCRIPTION</u>
NTYPE	1	Number of target types in classes 1 through 15
INDBEG	250/NTYPE	Smallest INDEXNO assigned to each target type
TYPENAME	250/NTYPE	Contains the name of each target type in the order referred to by INDBEG
CUMNO	15	CUMNO(I) is the total number of target types in each QUICK class, 1 through I
BTYPES	15	BTYPES(I) is the number of BLUE target types in class I
INDCLAS	15	Smallest INDEXNO assigned within each class
NAMCLAS	15	Hollerith name of target class
NVULN	1	Number of distinct values of vulnerability which occur in the data base
CVULN	63/NVULN	List of all distinct values of VULN; indexed by IVULN
NCOL	1	Number of collocated targets
COLAR	4000/NCOL	Packed data related to collocated targets
NTDEF	1	Number of targets with terminal ballistic missile defenses (BMD)
NTINTX	500/NTDEF	Number of terminal BMD interceptors; indexed by ITERM
NUMAINT	1	Number of zones with area BMD interceptors

Table 9. (cont.)
(Sheet 2 of 4)

<u>VARIABLE NAME</u>	<u>MAXIMUM LENGTH/ ACTUAL LENGTH</u>	<u>DESCRIPTION</u>
AINI	60	Number of area interceptors assigned each antiballistic missile (ABM) base
NUMNLRR	1	Number of zones with long-range BMD radars
NLRR	20	Number of long-range BMD radars covering each zone
NUMIOVR	1	Number of entries in the IOVERLAP array
IOVERLAP	20	Packed radar data (radar index, index of area defense zone)
MAXIND	1	The maximum (largest) index number (INDEXNO) assigned
STATUS	12000/MAXIND	Packed target data; indexed by INDEXNO
NMIS	1	Number of missile types
MIS	11*80/11*NMIS	Missile type data: MIS consists of 11 arrays, each indexed by JTYPE, containing PINC, PLABT, PDES, PEPE, TVUL, TRETARG, IREP, CEP, PKMIS, DELTA, and FUNCTION, respectively
NBOM	1	Number of bomber types
BOM	7*80/7*NBOM	Bomber type data: BOM consists of seven arrays, each indexed by JTYPE, containing PLABT, TMDEL, ABRATE, PRABT, CEP, DELTA, and FUNCTION, respectively
NTANK	1	Number of tanker types
TANK	5*40/5*NTANK	Tanker type data: TANK consists of five arrays, each indexed by JTYPE, containing PLABT, TMDEL, ABRATE, DELTA, and FUNCTION, respectively

Table 9. (cont.)
(Sheet 3 of 4)

<u>VARIABLE NAME</u>	<u>MAXIMUM LENGTH/ ACTUAL LENGTH</u>	<u>DESCRIPTION</u>
NASMT	1	Number of ASM types
ASMT	20/20•NASMT	ASM type data: two arrays, each indexed by ASMTYPE, containing PLABT and CEP, respectively
NWIID	1	Number of warhead types defined in the data base
WHD	30/30•NWIID	Warhead type data: three arrays, each indexed by WHIDTYPE, containing PDUD, YIELD, and CEP, respectively
NZONES	1	Number of air defense (bomber) zones
ZONES	30/30•NZONES	Three arrays, each indexed by ZONE: the first contains AREA; the second and third contain the accumulated effectiveness (EFFECTNES) for classes Defensive Control (DEFCONTR) and Interceptor (INTCPTOR), respectively
NDEF	1	Number of types of defensive command and control defined in the data base
DEFPOT	20/NDEF	EFFECTNES of each defensive command and control type, indexed by JTYPE
NINT	1	Number of interceptor aircraft types
INTPOT	20/NINT	EFFECTNES of each interceptor type, indexed by JTYPE
NBLUPLD	1	Number of BLUE payloads
IPAYLD	50/50•NBLUPLD	BLUE payload type data: five arrays, each indexed by PAYLOAD, containing NOBOMB1, WHIDTYPE, NWIDS, NDECOYS, and NAREADDEC, respectively

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

<u>VARIABLE NAME</u>	<u>MAXIMUM LENGTH/ ACTUAL LENGTH</u>	<u>DESCRIPTION</u>
NREDPLD	1	Number of RED payloads
IPAYLD	$5 \cdot 40/5 \cdot \text{NREDPLD}$	RED payload type data: same form as described above for side BLUE
NWHDS	1	Number of words in DBLDATA tables
PSASW	$100/\text{NWDS}$	Destruction before launch probability assigned a weapon for a specified time period
TSASW	$100/\text{NWDS}$	Time at which a time period ends for DBL data table; there may be up to 10 time periods for each table

following criteria: if the distance between two targets is less than the sum of the lethal radii of (for convenience) a one-megaton weapon for the hardnesses of the two targets, then they belong to the same collocation island; a collocation island consists of all targets which are linked by this distance criterion; and a single target does not form a collocation island. Thus, there is no theoretical upper limit to the size of an island; but, in practice, islands are usually rather small clusters. Two targets are said to be collocated if they belong to the same collocation island; a collocated target is one which belongs to some collocation island. The definition of a complex target is identical to the definition of a collocation island except that the distance criterion is one-half the distance used for collocation. Thus, every complex target is a subset of some collocation island. Collocation islands are used in the Simulator in determining the status of targets following warhead bursts; complex targets are used in the Plan Generator.

To perform the required processing and prepare the output files, INDEXER makes three passes through the data base. In the first, a temporary index JTYPE is assigned to selected items, and breakpoint tables reflecting the assignment of index numbers (INDEXNO) by target type and class are formed. In addition, the various vulnerabilities; i.e., values of the attribute VULN, assigned to data base items are retrieved. In pass two, an index number (INDEXNO) is assigned to each potential target. Then, subroutine COLOCATE is called to form collocation islands and complex targets. In pass three, the indexed data base tape INDEXDB is prepared, and the bulk of the data base input for the Simulator is compiled and written on the output tape SIMTAPE.

With regard to processing the data base, it should be noted that data items in classes WARHEAD, ASM, PAYLOAD, and DBLDATA must precede all other items on the data base tape. In addition, it is assumed that the individual launch sites assigned to each missile squadron are grouped together; i.e., entered sequentially, in the input data base, and that the value of the attribute ISITE (site number) is set to 1 for the first site appearing on the tape in each squadron. The use of this technique facilitates processing and eases the task of forming offensive weapon groups for use in the Plan Generator. When input in this manner, the missile squadrons are viewed as one launch base during plan generation, but the individual launcher locations are used in computing kill probabilities during simulation.

Pass 1 Processing (Sheets 1 to 6, Figure 111)

Figure 111 reflects the logical flow within program INDEXER. The program begins by calling subroutine INITIND to initialize counters and arrays. Then, the user-input parameters are read by calling subroutine READIN.

After keys* for packing data in the COLAR, IOVERLAP, and COMPLEX arrays have been formed and the filehandler initialized, pass 1 begins. The data base directory is read in and written onto a temporary data base file (LUN2) established on the disk using subroutines READER and WRITER. Then each data item contained on the input data base is read and processed. As each item is read in, the value of the attribute ICLASS is checked. Items for which ICLASS equals zero; i.e., the auxiliary classes, are copied onto the scratch data base file created on the disk, and the next item is read. The items for which ICLASS is greater than zero are checked to determine if the item is a nonlead element of a missile squadron; i.e., an element of a missile squadron for which ISITE is not set to 1. If so, the site number (ISITE) is incremented by one, and the item is added to the data base. The remaining items; i.e., missile sites for which ISITE equals 1 and all items in classes 2 through 15 (ICLASS = 2,3,...15) are processed and assigned a temporary index JTYPE.

The assigning of JTYPE takes place in the following manner. Items are separated according to side, and the array TYPENAME(J,ICLASS) is searched for a type name match ($1 \leq J \leq 40$ for side Blue, and $41 \leq J \leq 80$ for side Red). If no match is found, the first blank word is filled with the new type name. The attribute JTYPE then is assigned the corresponding value of the index J. Up to 40 types can be stored for each side and class. If this number is exceeded, the message TYPENAME TABLE TOO SMALL is printed, and the run halts.

The number of items of each type is stored in the corresponding word of the array TYPEFBL(J,ICLASS). Each time the type is encountered, the number is incremented by JADD, where JADD equals the number of sites per missile squadron or 1 for all other items.

After each item has been processed by type, the array CVULN is searched for the vulnerability attribute VULN defined for that item. If no match is found, the first blank word is filled with the value of VULN. Should more than 100 different values be encountered in processing the data base, an error message is printed to indicate an array overflow, the item is written on the scratch tape, and the next item is read from the data

* Key words returned by the utility function KEYMAKE which contain instructions for packing and unpacking of data according to a specified format.

base. Otherwise, the attribute IVULN* is defined in the data base to be the index of the word in the CVULN array which matches the item attribute VULN. The item is then written onto the scratch file, and the next item is read.

After all items on the input data base tape have been processed, INDEXER prepares the breakpoint tables. These tables contain the beginning index numbers (INDEXNOs) of each class and type and the number of RED and BLUE types in each class. The breakpoint tables are used by subsequent programs to obtain information as to the general content of the data base and to facilitate cross-referencing operations; e.g., the class, type, and side of an item can be determined based on its assigned index number (INDEXNO). These tables are formed by assigning a beginning index (INDBEG) to each type, starting with the first BLUE type in class 1 (see sheet 4). Each new index is found by incrementing the previous index by the number of items for the type under consideration (stored in TYPETBL). An index L to the array INDBEG then is stored in TYPETBL, replacing the number of items of that type. The number of BLUE types in each class (BTYPES) and the total number of types in all classes processed thus far (CUMNO) also are accumulated. After all beginning indices have been found, the array TYPENAME is collapsed so that types are listed consecutively in the order established by the INDBEG assignment, with no blank words remaining. The beginning index for each class (INDCLAS) then is calculated, and the breakpoint tables and vulnerability (CVULN) array are written on the simulation data tape SIMTAPE (see sheet 6).

Pass 2 Processing: (Sheets 6 to 11, Figure 111)

Before pass 2 begins, a critical distance for collocation (CRDIST) is calculated for each value in the vulnerability array CVULN. This distance, which is approximately the lethal radius of a one-megaton weapon corresponding to the value VULN, is stored in degrees of latitude to minimize computation in forming collocation islands. The type beginning indices (INDBEG) then are stored in a current index array (INDCUR), and the first item is read from the data base file written during pass 1 (see statement 200, sheet 6). If the item is not a potential target and will not be

* The upper limit of 100 vulnerabilities was established to aid NMCSSC in processing large data bases containing an unknown number of vulnerability codes. Up to 100 different codes will be reflected in the print of the CVULN array. Actually, the system is limited by the structure of the STATUS array to an upper limit of 63 vulnerabilities. If more than 63 different vulnerability codes are encountered, the message SIMTAPE UNUSABLE---TOO MANY VULNERABILITIES is printed. This condition does not preclude the use of the INDEXDB tape in plan generation but does prohibit the simulation of such plans.

assigned an index number INDEXNO (i.e., if ICLASS=0), it is copied onto a scratch disk file (LUN3, see statement 215, sheet 8) and the next item is read. Otherwise, the item is assigned a value for the attribute INDEXNO as follows.

The type index (L) to the array INDCUR is retrieved from TYPETBL, and the corresponding value in INDCUR is assigned to the attribute INDEXNO (sheet 7). Index L then is assigned to the attribute ITYPE. In addition, the index JTYPE is reassigned so that all types within each class are indexed consecutively. The value in INDCUR then is incremented by NADD, where NADD equals the number of sites per squadron for missiles and 1 otherwise. It is assumed that the first missile site in each squadron is processed first; subsequent sites in the squadron are assigned consecutive values of INDEXNO by incrementing the index assigned to the first site by a modified site index (ISITE-1).

After INDEXNO has been assigned in the data base, it is stored, along with the latitude (LAT), longitude (LONG), and critical distance (CRDIST) for use in forming collocation islands. Due to the restriction of available storage in the computer memory, most of the collocation data must be kept on the disk. This is accomplished by splitting the earth into 10 longitudinal sectors and writing the information for all sectors but the first on the scratch disk file (see sheet 8). Information for items in the first sector is stored in the arrays IND(INDEXNO), X(LONG), Y(LAT), and Z(CRDIST) for immediate use in subroutine COLOCATE (statement 267, sheet 8). At this point, the item is written on the scratch tape and the pass continues.

Once all the data for collocation have been collected, a check is made to insure that no sector contains more than 4,000* items (sheet 9). If this number is exceeded, the message TOO MANY ITEMS FOR COLLOCATION, together with the sector concerned and the number of targets in the segment, are printed, and the number of targets to be considered is limited to 4,000. When all sectors have been checked, subroutine COLOCATE is called to process the data stored in the arrays IND, X, Y, and Z for the first earth sector.

When control is returned from COLOCATE, the scratch file is read and the arrays IND, X, Y, and Z are filled with the data for the next sector. The data for the remaining sectors are copied onto a second disk scratch file, and subroutine COLOCATE is called to process the data for the next sector. This process is repeated, with the input/output roles of the two scratch files LUN6 and 7 alternating, until all earth sectors have been processed. (The input and output scratch files used in this process are designated LIN and LOUF, respectively.)

* Program coding reflects the variable MTARSEC (maximum targets per sector) which is set at 4,000.

When control is returned from subroutine COLOCATE after processing the last sector, the scratch file LUN2 containing the COLAR array (written by COLOCATE) is terminated. A check is then made to ensure that no more than 4,000 target elements were found to be collocated. If the number of collocated targets exceeds 4,000, counters are set to cause the message ARRAY OVERFLOW COL TGETS (number of collocated targets) to be printed in the final phase of processing.

Pass 3 Processing (Sheets 11 to 18, Figure 111)

Before the pass begins, relevant arrays and counters are initialized, keys are formed for use in packing data into the array STATUS, and the maximum value of INDEXNO is retrieved (sheet 11). The directory of the data base written during pass 2 then is read from LUN3 and copied onto the final data base tape INDEXDB.

As each item is read from the data base, it again is tested for class assignment. Items which are not potential targets (i.e., ICLASS=0) are separated into appropriate classes (sheet 12). Relevant data for items in classes WARHEAD, ASM, ZONE, and DBLDATA are stored in the corresponding array (see table 10), and the number of items in each class is accumulated. Items in class PAYLOAD are separated by side and assigned a new payload index (NBLUPLD or NREDPLD), beginning with "1" on each side. The index is stored under the old index PAYLOAD in array KNARRAY, and the attribute PAYLOAD is changed in the data base to represent the new index. Relevant data then are retrieved and stored in corresponding arrays (see table 10). The item is written onto INDEXDB, and the next item is read. As each of the items (data base records) associated with these auxiliary classes is processed, a check is made to ensure that the maximum limit for the class is not exceeded (for maximum limits, see description of common block /MAX/ in table 13). If the upper limit is exceeded, flags are set to cause an array overflow error message to be printed. The item is then written onto the INDEXDB tape without storing the associated data (sheet 12).

With one exception, data base items from the auxiliary classes POINT, BOUNDARY, CORRIDOR, and LEGS are merely written onto the INDEXDB tape and the next item is read. The exception pertains to a check of class CORRIDOR items to ensure that the corridor types DUMMY and NAVALAIR are defined in the data base (required if tactical bombers or bombers with the attribute PKNV>0 are to be considered in plan generation). If these corridor types are not encountered in processing the data base, a message is printed to alert the user to a possible error. As appropriate, one or both of the following messages is printed: DUMMY CORRIDOR FOR TACTICAL AIR NOT DEFINED; and/or NAVAL AIR CORRIDOR NOT DEFINED.

Table 10. Warhead, ASM, Payload, and DBL Data

<u>ARRAY</u>	<u>POSITION</u>	<u>ATTRIBUTE</u>	<u>DESCRIPTION</u>
WHD	WHDTYPE,1	PDUD	Probability of a dud
WHD	WHDTYPE,2	YIELD	Yield (megatons)
WHD	WHDTYPE,3	CEP	CEP (nautical miles)
ASMT	ASMTYPE,1	PLABT	Probability of launch abort
ASMT	ASMTYPE,2	CEP	CEP (nautical miles)
ZONES	ZONE,1	AREA	Area of a defense zone (millions of nautical miles ²)
MIRV	PAYLOAD,ISIDE	NOBOMB1	Number of bombs carried by vehicle
INWHDS	PAYLOAD,ISIDE	NWIDS	Number of warheads (missiles)
IWHDTYPE	PAYLOAD,ISIDE	WHDTYPE	Warhead type index
INDECOYS	PAYLOAD,ISIDE	NDECOYS	Number of decoys carried by vehicle
INARDEC	PAYLOAD,ISIDE	NAREDEC	Number of area decoys
TMASW		TPASW	Time for DBL data table
OBLASW		PSASW	DBL probability for DBL data table

If the item under consideration is a potential target, it is checked for collocation (logical array COLO). Collocated targets are so indicated by packing a "1" under TCOL in the STATUS array (indexed by INDEXNO (see table 11)). The logical array COMP then is checked to see if the collocated target is also a member of a complex. If it is, function ICPL is called to unpack the index ICOMPLEX from array COMPLEX. ICOMPLEX then is assigned in the data base.

For all potential targets, the values of the attributes TARDEFHI, TARDEFLO, IVULN, IATTACK, IAREA, ADEFZON, and ADEFCMP are packed into the STATUS array (table 11).

Dynamic targets (items assigned the attribute-value pair TGTSTAT=1 in the data base) are so indicated by packing a "1" under the alive-dead indicator TSTAT and status flag IKEEP in the STATUS array. In addition, for all items in ICLASS 4 and 5 (currently classes defensive control DEFCONTR, and interceptor INTCPTR), the value of the attribute ZONE is packed into the STATUS array; the defense potential of the target (attribute EFFECTNES) is accumulated in the zone array (table 11); and the value of EFFECTNES, which must be the same for all items of a given type, is stored by ITYPE in the array CPACTY if the item is the first of its type. For items which are antiballistic missile (ABM) bases (ADEFCMP=1, 2, or 3), the value of NAIN is stored in the AINT array, doubly indexed by ABM defense zone (ADEFZON) and ABM defense component (ADEFCMP). For items which are radars (ADEFCMP=4), NLRR is incremented by one for each of the area defense zones AZON1, AZON2, and AZON3 supported by the radar. In addition, the values of the attributes INDEXNO, AZON1, AZON2, and AZON3 are packed in a word of the array LOVERLAP (table 12) indexed by ADEFZON.

Items belonging to ICLASS 1, 2, and 3 (currently classes MISSILE, BOMBER, and TANKER) now are tested for payload. If the payload index (attribute PAYLOAD) is defined, the new payload index (NBLUPLD or NREDPLD) is retrieved from array KNARRAY and assigned to the attribute PAYLOAD in the data base. Certain data which do not change within a given type are stored in the corresponding MISSILE, BOMBER, or TANKER array for the first item of each type (indicated in array ICHK). The item then is written onto INDEXDB and the next item is read.

After all items have been processed, the array COLAR, which was written during subroutine COLOCATE, is read into memory and copied onto the Simulator tape SIMTAPE (sheet 17). Subroutine TDEFSTT now is called to make the final assignment of terminal defense indices (NTDEF). When control is returned to INDEXER, data for the Simulator are written on SIMTAPE in the format shown in table 9, and the tape is terminated. Subroutine SKIPFILE then is called, so that the index breakpoint tables may be written at the end of the INDEXDB tape. Depending upon the print

Table 11. Structure of a Word in the Array STATUS

<u>BITS</u>	<u>VARIABLE</u>	<u>VALUES</u>	<u>DESCRIPTION</u>
0-1	TSTAT	0,1	Value equals 1 for (alive) dynamic targets, equals 0 for nondynamic targets
2	IKEEP	0,1	Value equals 1 for dynamic targets, equals 0 for nondynamic targets
3-5	TCOL	0,1	Collocation indicator; 1=collocated
6-8	TARDEFLO	0-7	Level of local bomber defenses at low altitude
9-11	TARDEPHI	0-7	Level of local bomber defenses at high altitude
12-14	IATTACK	0,1	Selection index for preferential BMD; value equals 1 if selection is user-directed
15-20	IVULN	1-63	Index to vulnerability number table
21-23	IAREA	0,1	Value equals 1 if item assigned to an area BMD zone; i. e., item attribute ADEFZON > 0, and otherwise set to 0
24-32	ITERM	0-500	Terminal BMD defense index
33-38	ADEFZON	0-63	Area BMD zone number
39-41	ADEFCMP	0-4	Area BMD component code (ABM base or radar)
42-47	ZONE	1-63	Bomber area defense zone number (ICLASS 4 and 5 only)

Table 12. Structure of a Word in the Array 1OVERLAP

<u>BITS</u>	<u>VARIABLE</u>	<u>VALUES</u>	<u>DESCRIPTION</u>
0-14	ITAR	1-12000	Index number INDEXNO of the radar
15-20	AZON1	0-20	First area BMD defense zone covered by radar
21-26	AZON2	0-20	Second area BMD defense zone covered by radar
27-32	AZON3	0-20	Third area BMD defense zone covered by radar

controls, the data on SIMTAPE and in the STATUS array may be printed at this point. Otherwise, the processing of INDEXER is complete.

Common Block Definition

Program INDEXER references the following utility routine common blocks, which are described in appendix A of this manual: /DIRECTRY/, /EDITERM/, /EDITAPE/, /ERRORM/, /IFTPRINT/, /ITP/, /MYIDENT/, /NOPRINT/, /PROCESS/, and /TWORD/.

Table 13 lists the common blocks which are local to this program and identifies the arrays contained in them.

Table 13. Program INDEXER Common Blocks
(Sheet 1 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
1	NISL	Number of collocation islands
	NN	Index to COLAR array
	NCOL	Number of collocated targets
	NITEM	Number of items in segment being processed
	X	Array containing longitude
	Y	Array containing latitude
	Z	Array containing critical distance
	IND	Array containing index number INDEXNO
	STATUS*	Array containing packed data for targets
2	COL	Array indicating targets belonging to a collocation island
	CL	Array indicating targets belonging to current collocation island
	CLT	Array indicating targets in current island which have not been checked for further collocation
	CP	Array indicating targets belonging to a complex
3	ICHR	Complex index
	ISTORE	Index to array COMPLEX

* Equivalenced to X to save storage space.

Table 13. (cont.)
(Sheet 2 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
3 (cont.)		
	COLAR	Array containing packed data for collocated targets
	COMPLEX	Array containing packed data for complex targets
4	NULL	Beginning INDEXNO reference point in CUMNO
	CUMNO(I)	Total number of type
	BTYPES(I)	Number of BLUE types in class I
	INDCLAS(I)	Beginning index number for class I
	INDBEG(J)	Beginning index number for type J
	TYPENAM	Array containing type names
	TYPETBL	Array containing number of items of each type
	INDCUR(J)	Current index number for type J
	MIS(FMIS)	Array containing missile type data
	BOM	Array containing bomber type data
	TANK	Array containing tanker type data
	ASMT	Array containing ASM data
	WHD	Array containing warhead data
	ZONES	Array containing zone data
	CAPACTY	Array containing effectiveness (EFFECTNES)
	ICHK	Array used to process information pertaining to all items of the same type

Table 13. (cont.)
(Sheet 3 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
4 (cont.)	MIRV	Array containing number of multiple independent re-entry vehicles
	IWHDTYP	Array containing warhead type index
	INWHDS	Array containing number of warheads carried by vehicle
	INDECYS	Array containing number of bomber decoys
	INARDEC	Array containing number of area decoys
	NAMCLAS	Array containing Hollerith name of target classes
5	NTDEF	Number of terminal ABM defenses
	ITERM	Array containing index number of terminally defended targets
	NTINTX	Array containing number of terminal interceptors
7	COLO	Logical array which flags collocated targets
	COMP	Logical array which flags complex targets
	LTERM*	Logical array which flags terminally defended targets

* Equivalenced to COMP to save storage space.

Table 13. (cont.)
(Sheet 4 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
9	ICHKFLG	Array containing names of classes which have exceeded maximum number of target types per class
	ICHKNUM	Array containing number of target types greater than maximum allowed per class
	NCHKFLG	Array containing Hollerith identifier for variables and arrays which exceed upper limits
	NCHKNUM	Array containing number of items associated with each NCHKFLG entry
10	KNARRAY	Array containing payload indices for each side
AREADAT	AJNT	Array containing number of area interceptors
	NLRR	Array containing number of long-range radars covering a zone
	IOVERLAP	Array containing packed data for radars
COMP	LCOMP	Array containing index numbers of targets in current collocation island
KEY	KEY	Array containing keys for packing data into COLAR

Table 13. (cont.)
(Sheet 5 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
KEYC	KEYC1	Key for packing index number into COMPLEX
	KEYC2	Key for packing complex index into COMPLEX
	MASK1	Mask corresponding to KEYC1
	MASK2	Mask corresponding to KEYC2
KEYS	TSTAT (KEYS(1))	Key for packing status flag
	TCOL (KEYS(2))	Key for packing collocation flag
	TARDLO (KEYS(3))	Key for packing TARDEFLO
	TARDHI (KEYS(4))	Key for packing TARDEFHI
	KATTACK (KEYS(5))	Key for packing IATTACK
	TVULN (KEYS(6))	Key for packing IVULN
	KARDEF (KEYS(7))	Key for packing IAREA
	KTERM (KEYS(8))	Key for packing ITERM
	ZON (KEYS(9))	Key for packing ZONE
	KDEFZON (KEYS(10))	Key for packing ADEFZON
	KDEFCMP (KEYS(11))	Key for packing ADEFCMP

Table 13. (cont.)
(Sheet 6 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
MAX		Array containing maximum limits
	MALERT	Alert conditions (2)
	MASMTYP	ASM types (20)
	MBNDRY	Boundary legs (200)
	MCCREGN	Command/control (20)
	MCLASS	Weapon classes (2)
	MCNTRY	Country codes (250)
	MCORR	Penetration corridors (30)
	MCORTYP	Corridor types (5)
	MDPEN	Depenetration corridor route points (50)
	MDEPNLG	Depenetration legs (50)
	MGROUP	Weapon groups (200)
	MPAYLOD	Payload types per side (40)
	MRECOVR	Recovery bases (200)
	MRECVLG	Recovery legs (60)
	MREF	Directed refuel points (20)
	MRTLEG	Route legs (200)
	MRTPT	Route points (200)
	MSPERMT	Missile sites per multiple target (5)

Table 13. (cont.)
(Sheet 7 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
MAX (cont.)		
	MTANKBS	Tanker bases (60)
	MTARCLS	Target classes (15)
	MTARCOL	Collocated targets (4,000)
	MTARCPX	Target complexes (4,000)
	MTARERS	Targets per collocation island (100)
	MTARGET	Targets (program ALOC limit) (5,000)
	MTARIND	Target index numbers (12,000)
	MTARSEC	Targets per earth sector (4,000)
	MTARTEL	Targets with terminal BMD interceptors (500)
	MTARTYP	Total target types (250)
	MTARVAL	Target complexes with value greater than zero (2,500 per side)
	MTELMCM	Target elements per complex (40)
	MTOTBAS	Weapon bases per group (150)
	MTYPE	Weapon types (missiles plus bombers per side) (80)
	MVULN	Vulnerabilities (100)
	MWEAPGP	Weapons per group (1,000)
	MWHDTYPE	Warhead types (50)
	MZONEPT	Zone points (200)
	MZONES	Zones (63)
	MTARPCL	Target types/class (missile and bomber 40; all other classes 20)

Table 13. (cont.)
(Sheet 8 of 9)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
MAX (cont.)	MABMDFZ	Area ballistic missile defense (BMD) zones (20)
	MABMSIT	Missile sites per ballistic missile defense zone (3)
NAVALTB	TMASW	Array containing times for time-dependent DBL data tables
	DBLASW	Array containing cumulative DBL probabilities for DBL data tables
	ITMAX	Maximum number of times per DBL data table
	IDBLMAX	Number of DBL data tables currently defined
PRNT	IPRNT	Array containing print controls
RADATA	PA	Arrays containing vulnerability information for subroutine VLRAD1
	PG	
	QA	
	QG	
TRANS	NASMT	Number of ASM (air-to-surface missile) types in data base
	NVULN	Number of distinct vulnerabilities assigned to items in data base

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

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
TRANS (cont.)		
	NBLUPLD	Number of BLUE payloads in base
	MAXIND	Number of index numbers stored in STATUS array
	NREDPLD	Number of RED payloads in the data base
	LMAX	Total number of types assigned index numbers in data base
WRIT		
	WR(IWR)	Array used for collocation data
	NR	Array containing number of items in an earth sector
	XLONG	Designates sector boundaries

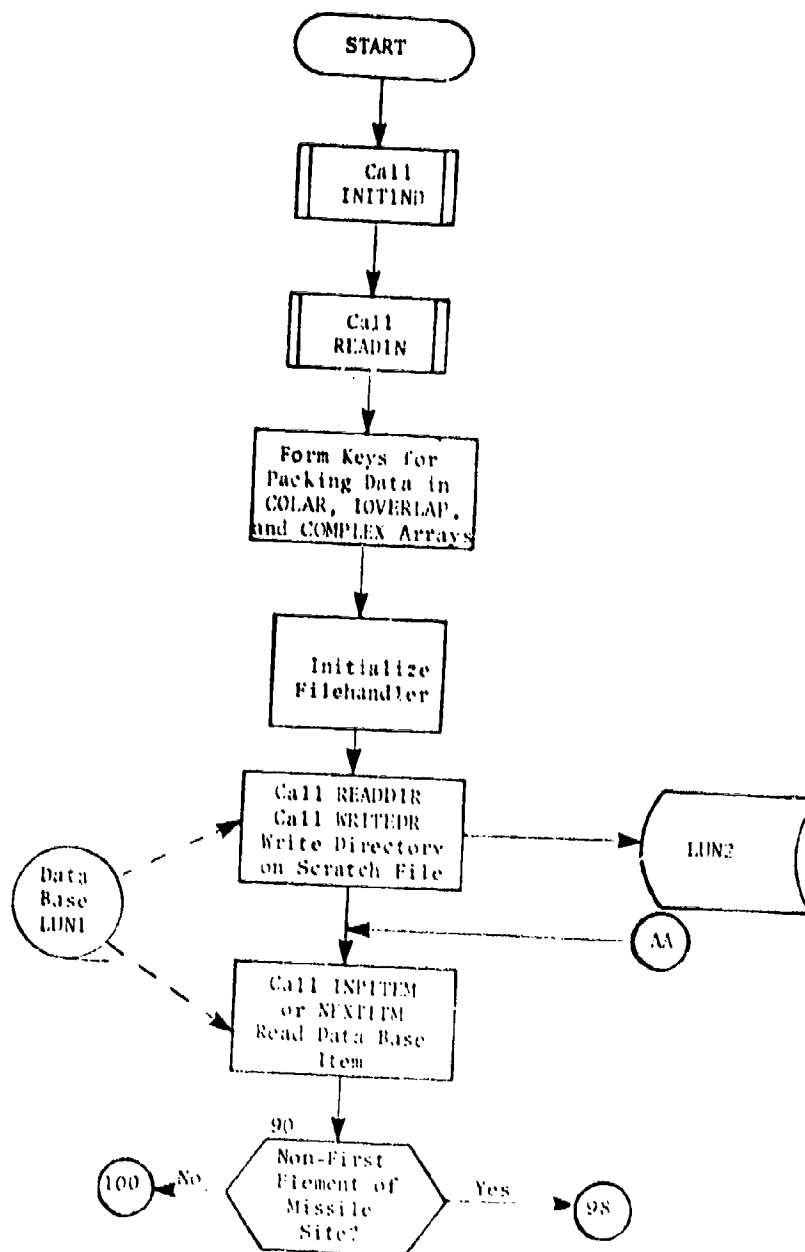


Fig. 111. Program INDIAN
(Sheet 1 of 18)

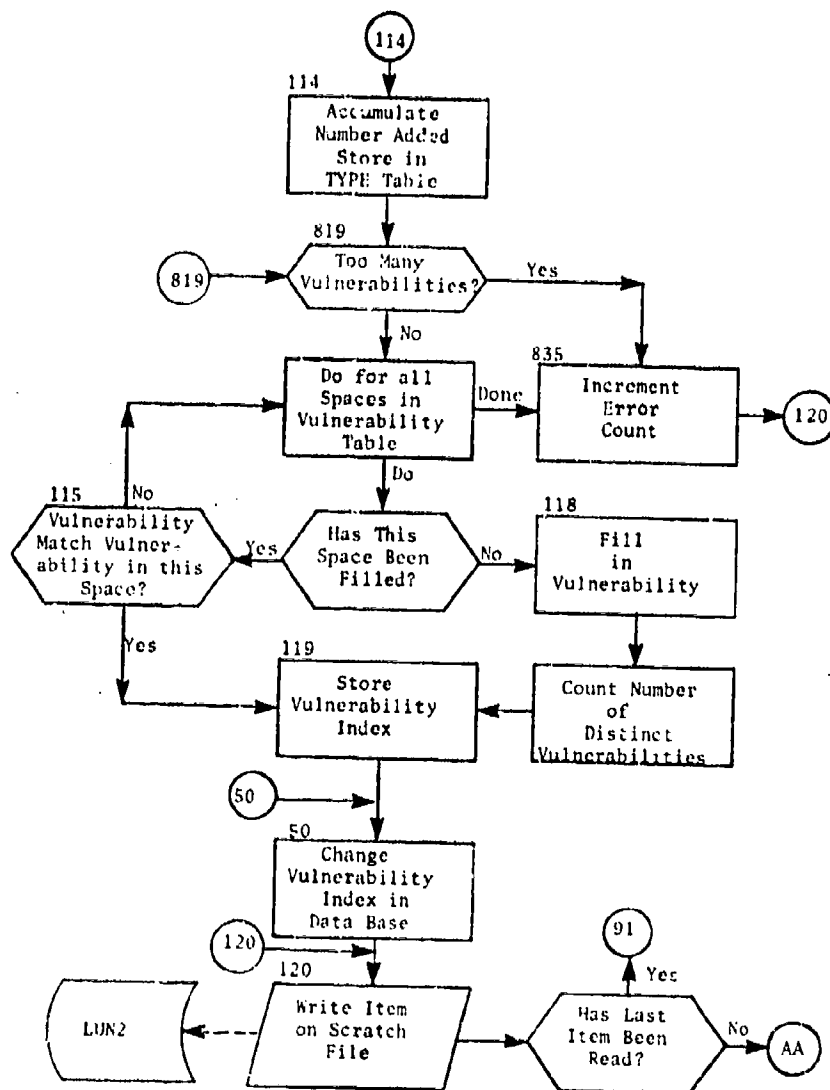


Fig. 111. (cont.)
(Sheet 3 of 18)

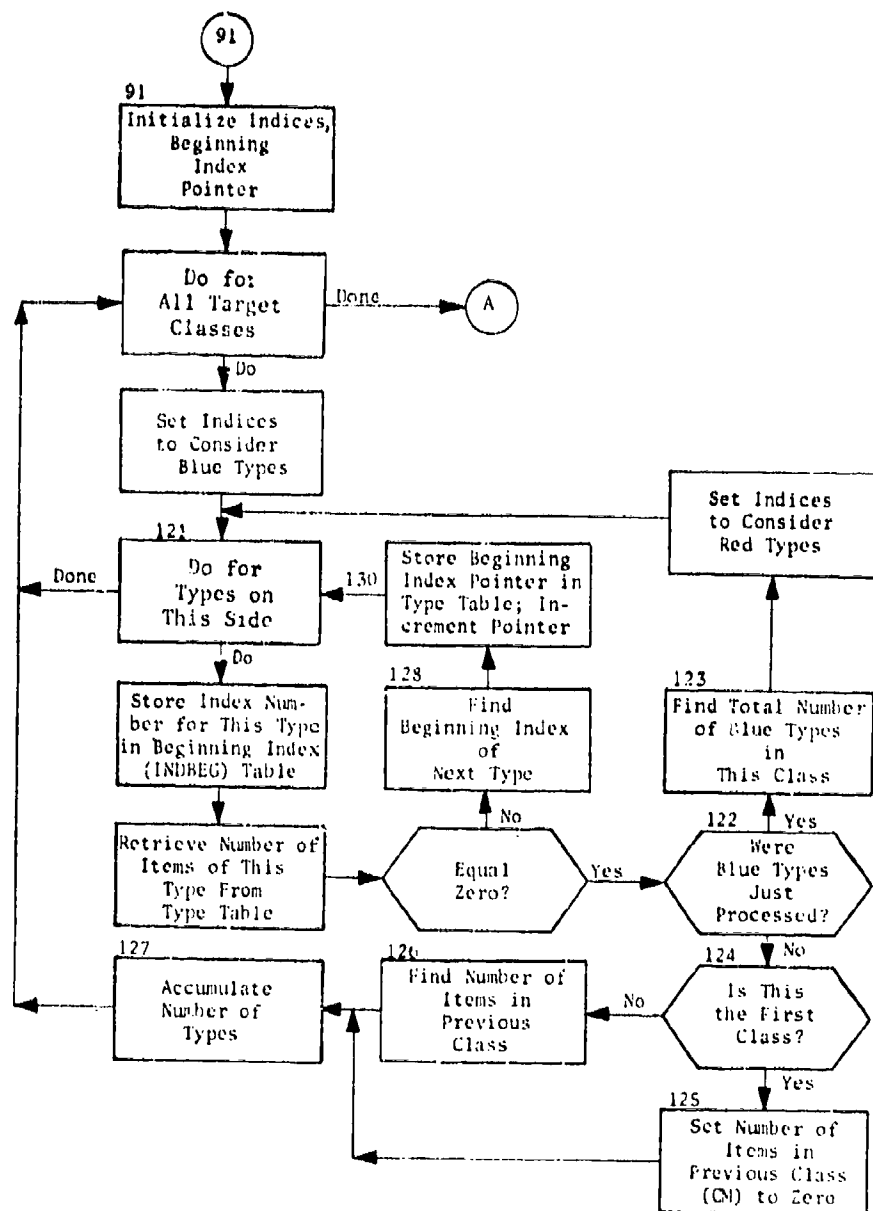


Fig. 111. (cont.)
(Sheet 4 of 18)

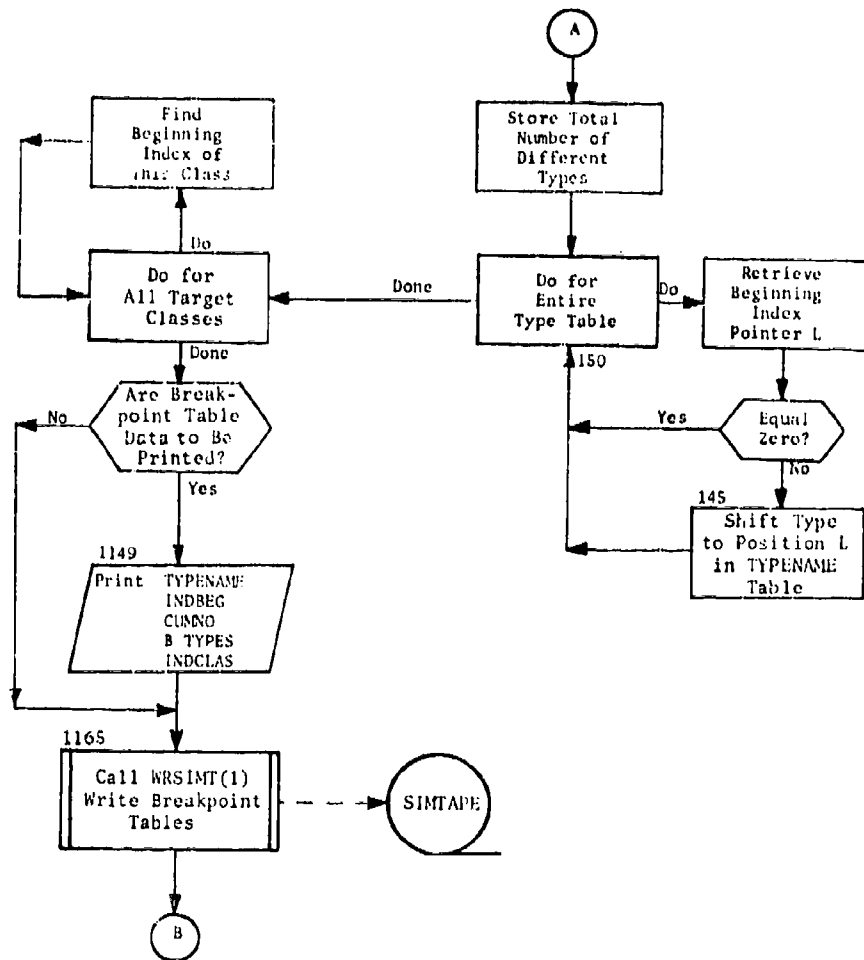


Fig. 111. (cont.)
(Sheet 5 of 18)

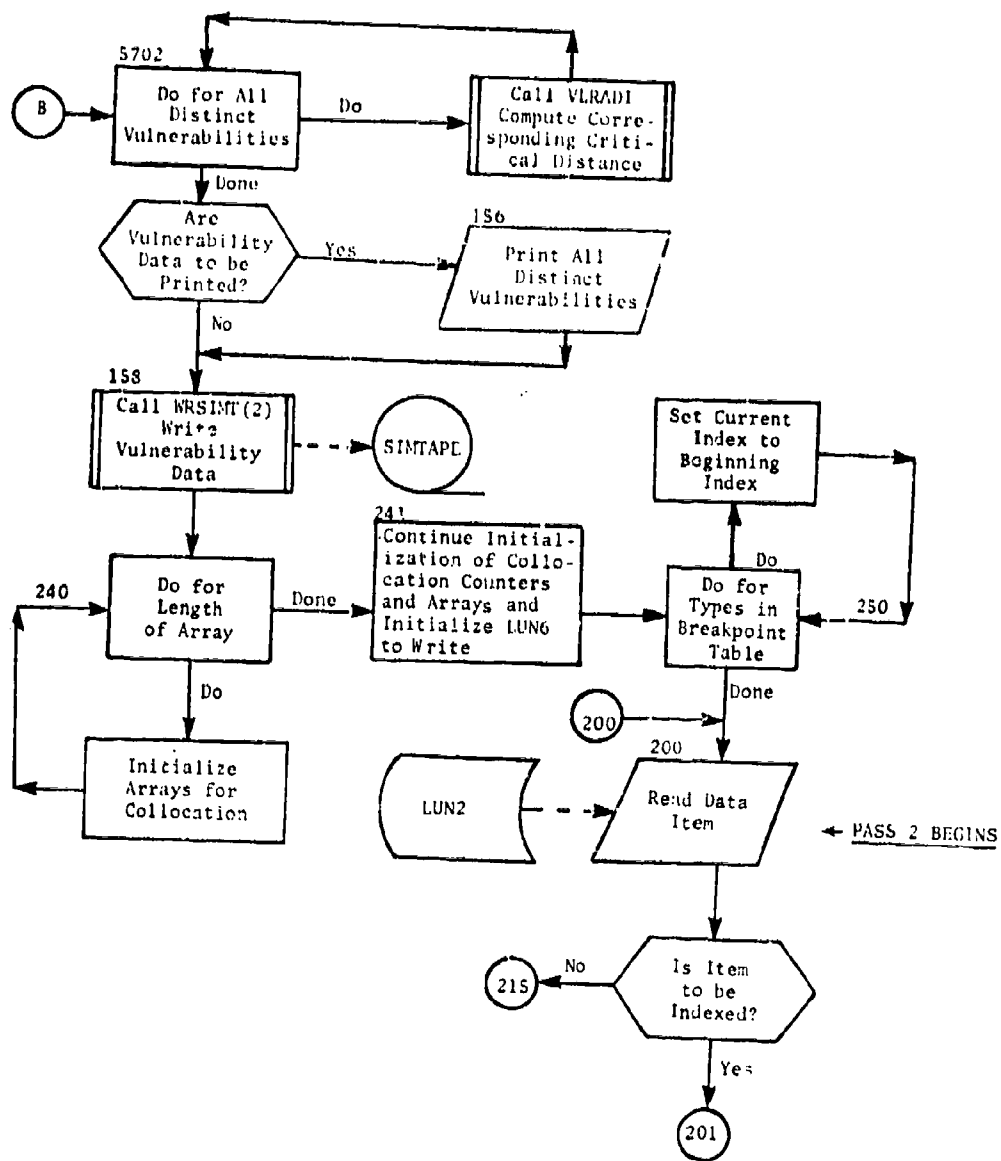


Fig. 111. (cont.)
(Sheet 6 of 18)

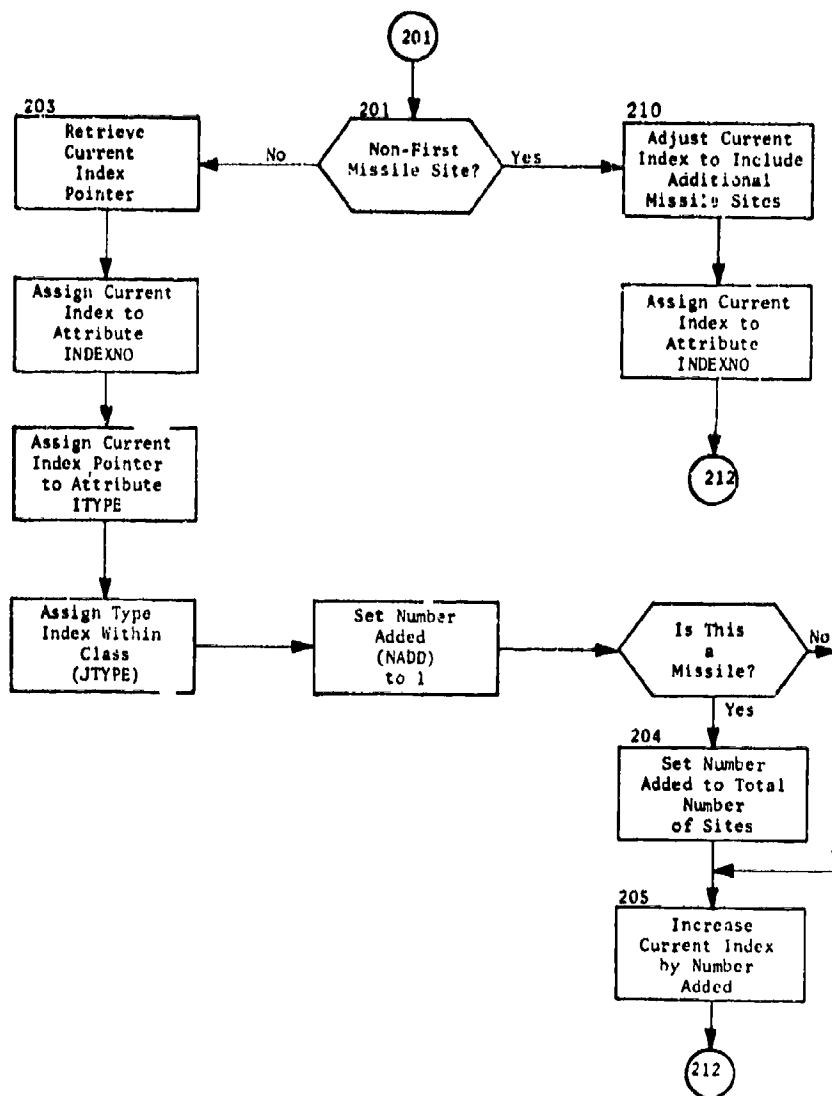


Fig. 111. (cont.)
(Sheet 7 of 18)

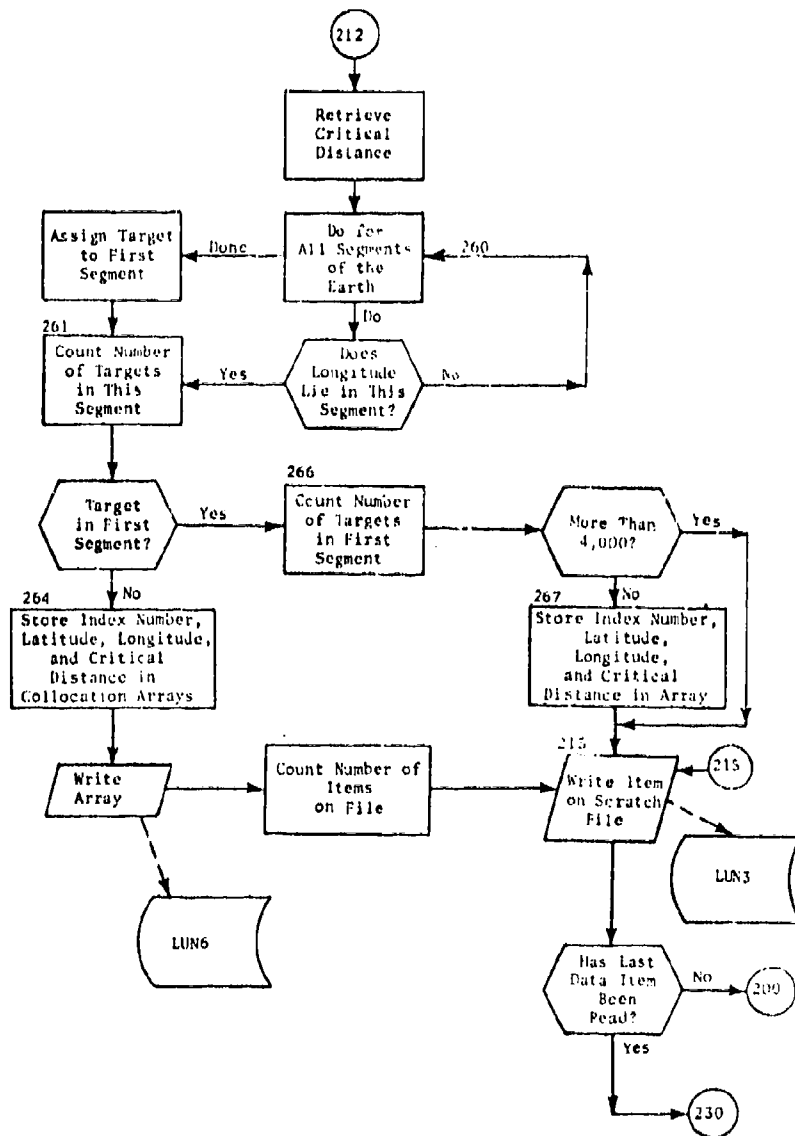


Fig. 111. (cont.)
(Sheet 8 of 18)

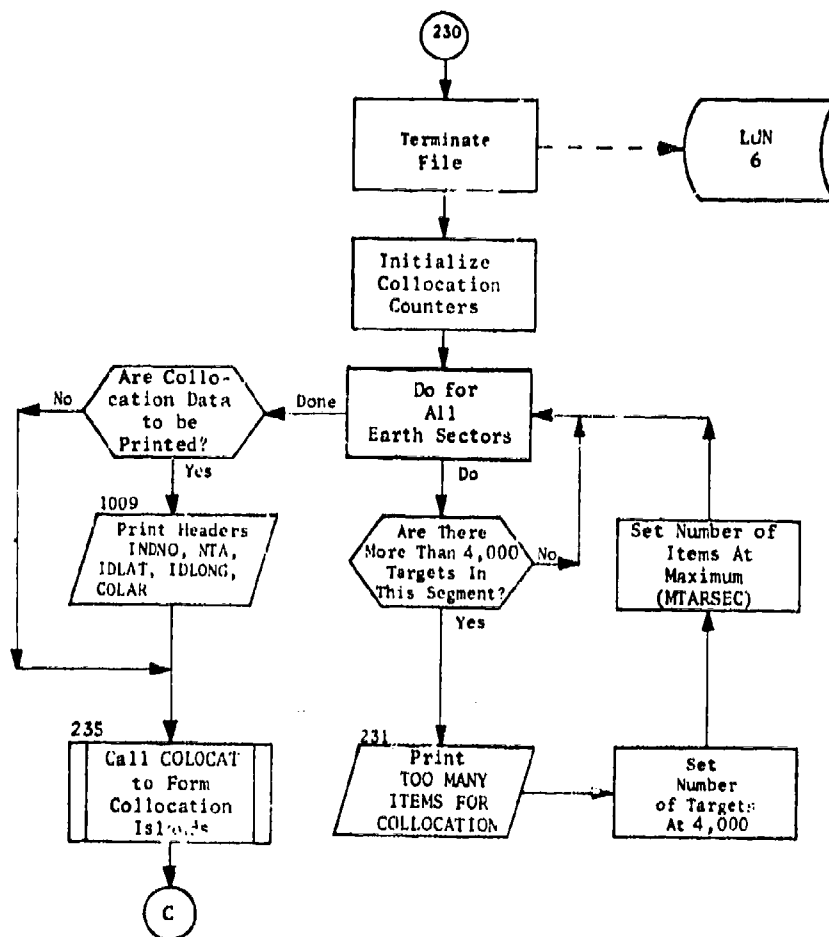


Fig. 111. (cont.)
(Sheet 9 of 18)

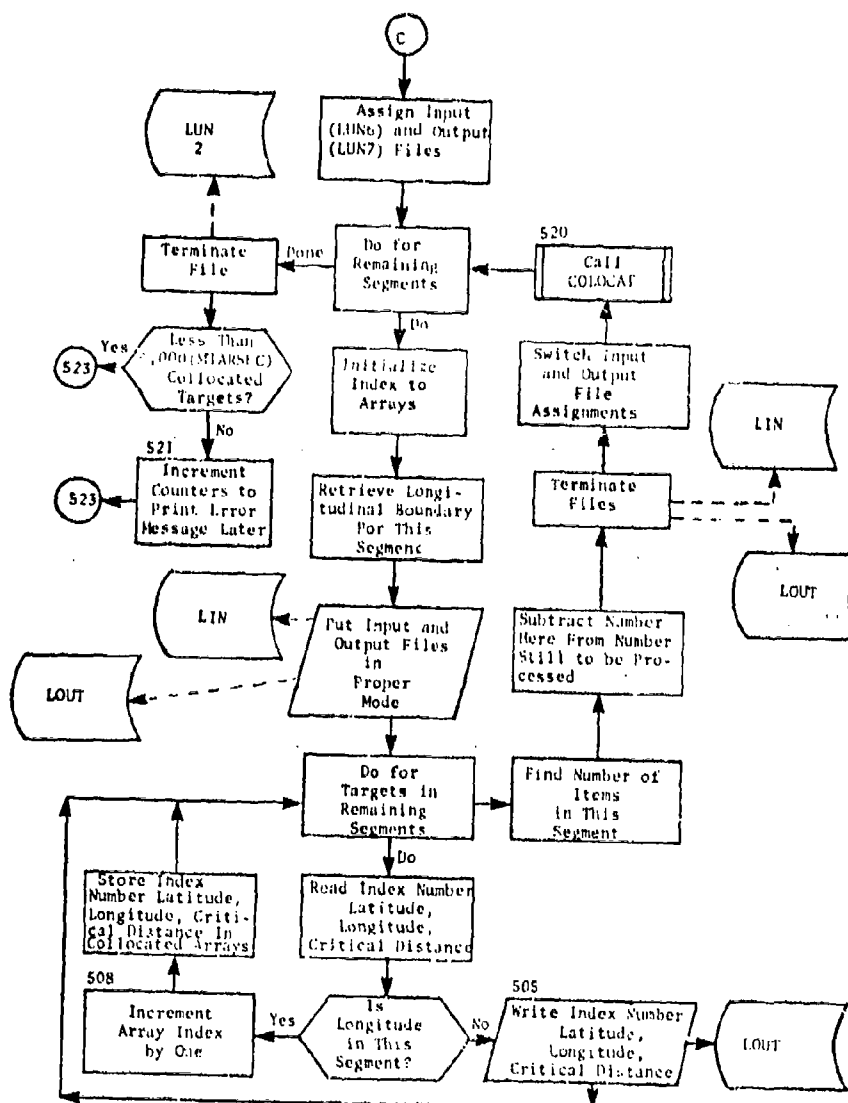


Fig. 111. (cont.)
(Sheet 10 of 18)

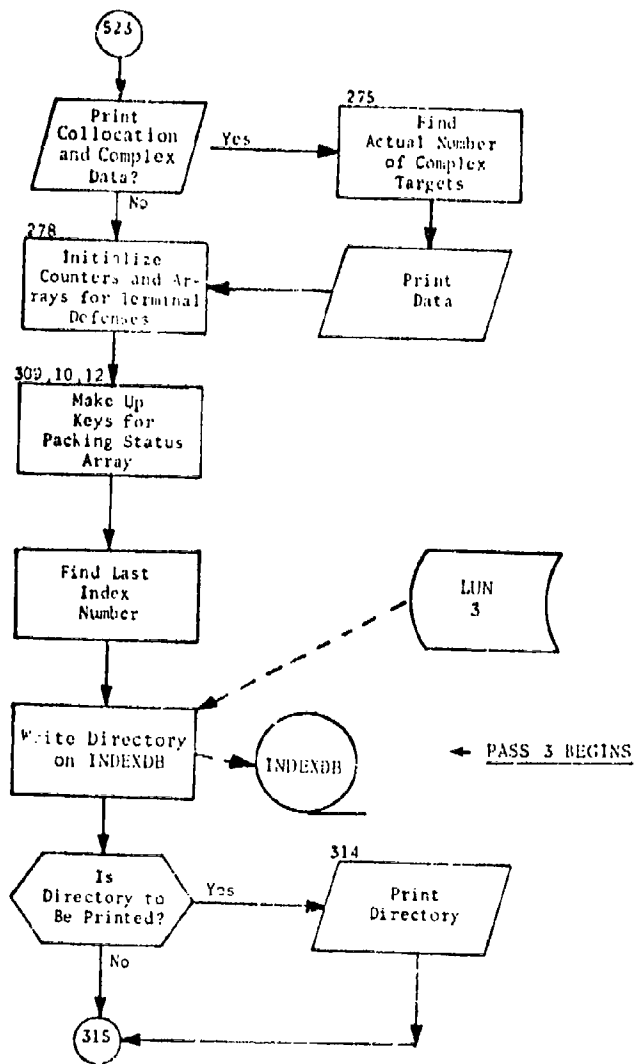


Fig. 111. (cont.)
(Sheet 11 of 18)

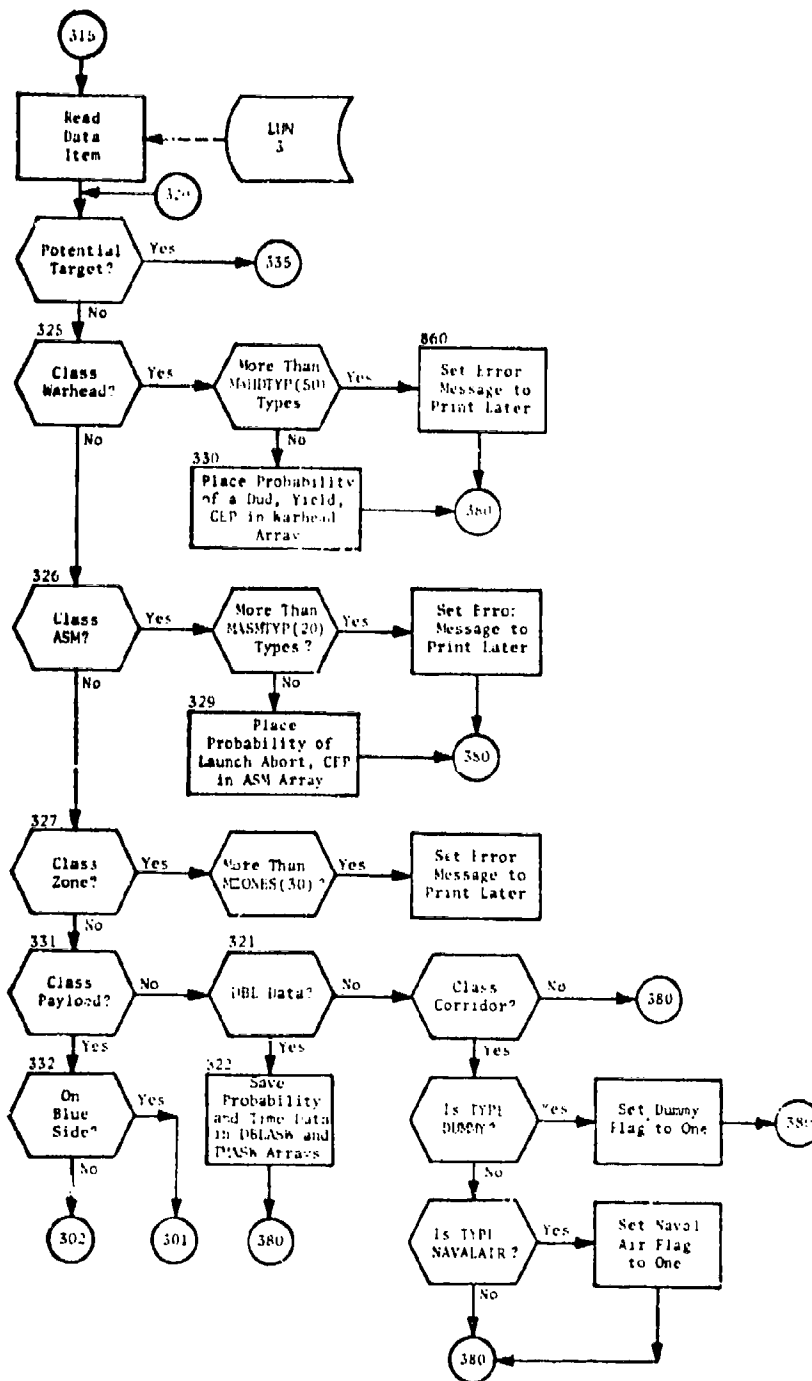


Fig. 111. (cont.)
(Sheet 12 of 18)

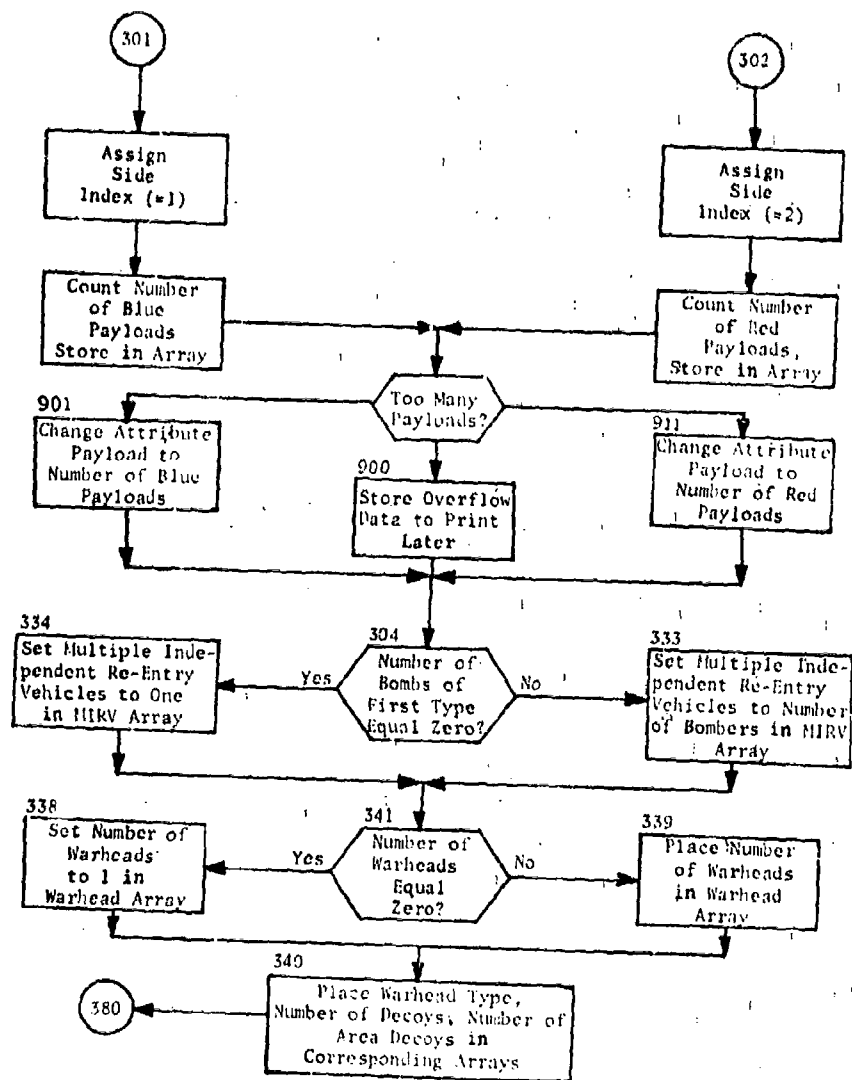


Fig. 111. (cont.)
(Sheet 13 of 18)

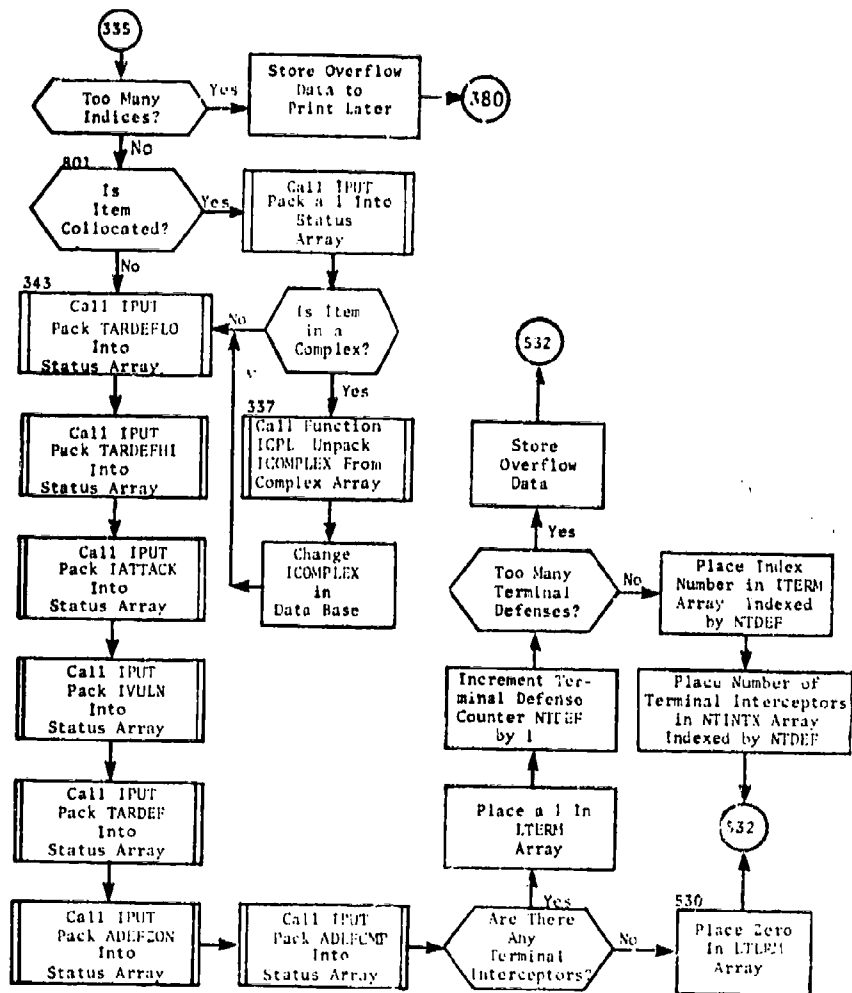


Fig. 111. (cont.)
(Sheet 14 of 18)

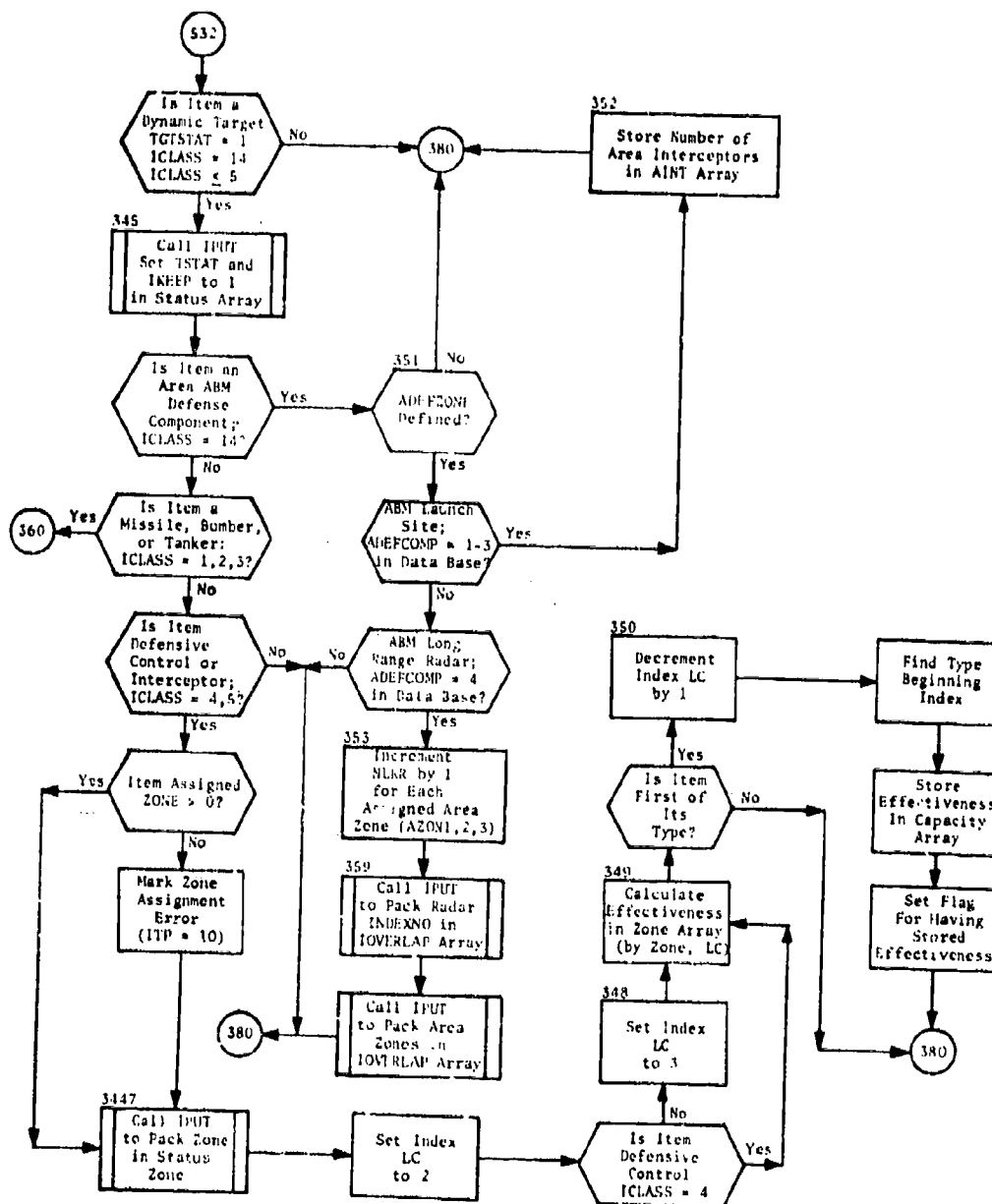


Fig. 111. (cont.)
(Sheet 15 of 18)

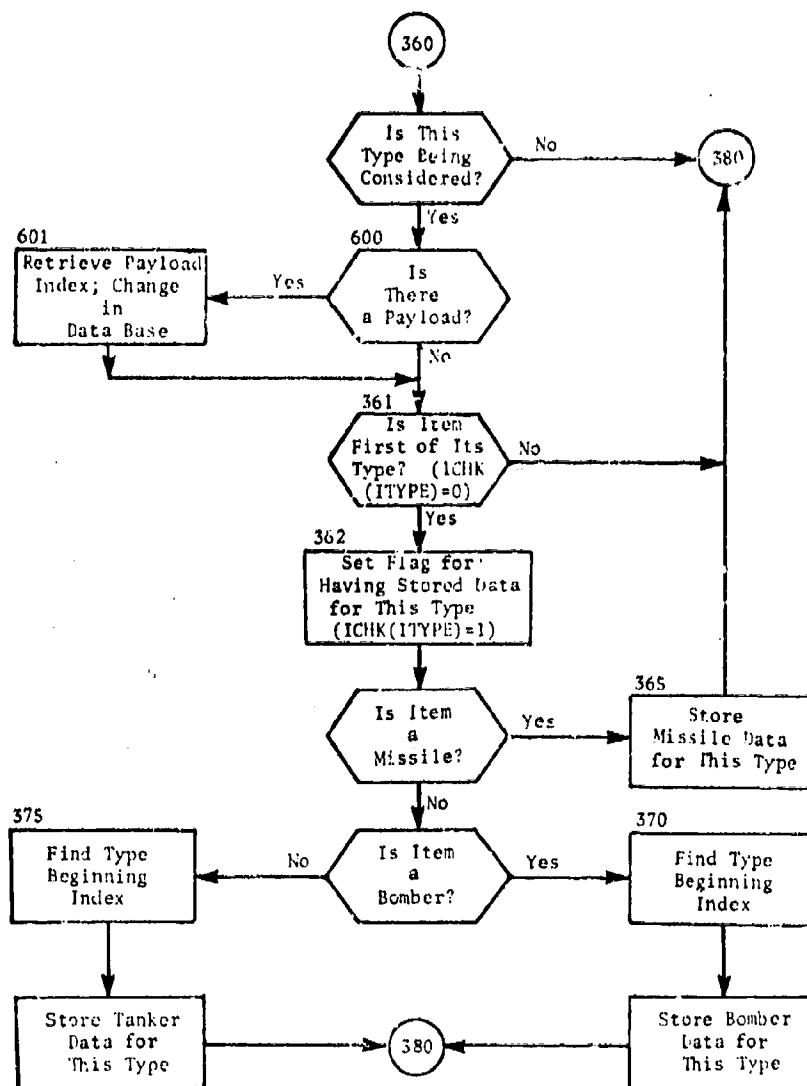


Fig. 111. (cont.)
(Sheet 16 of 18)

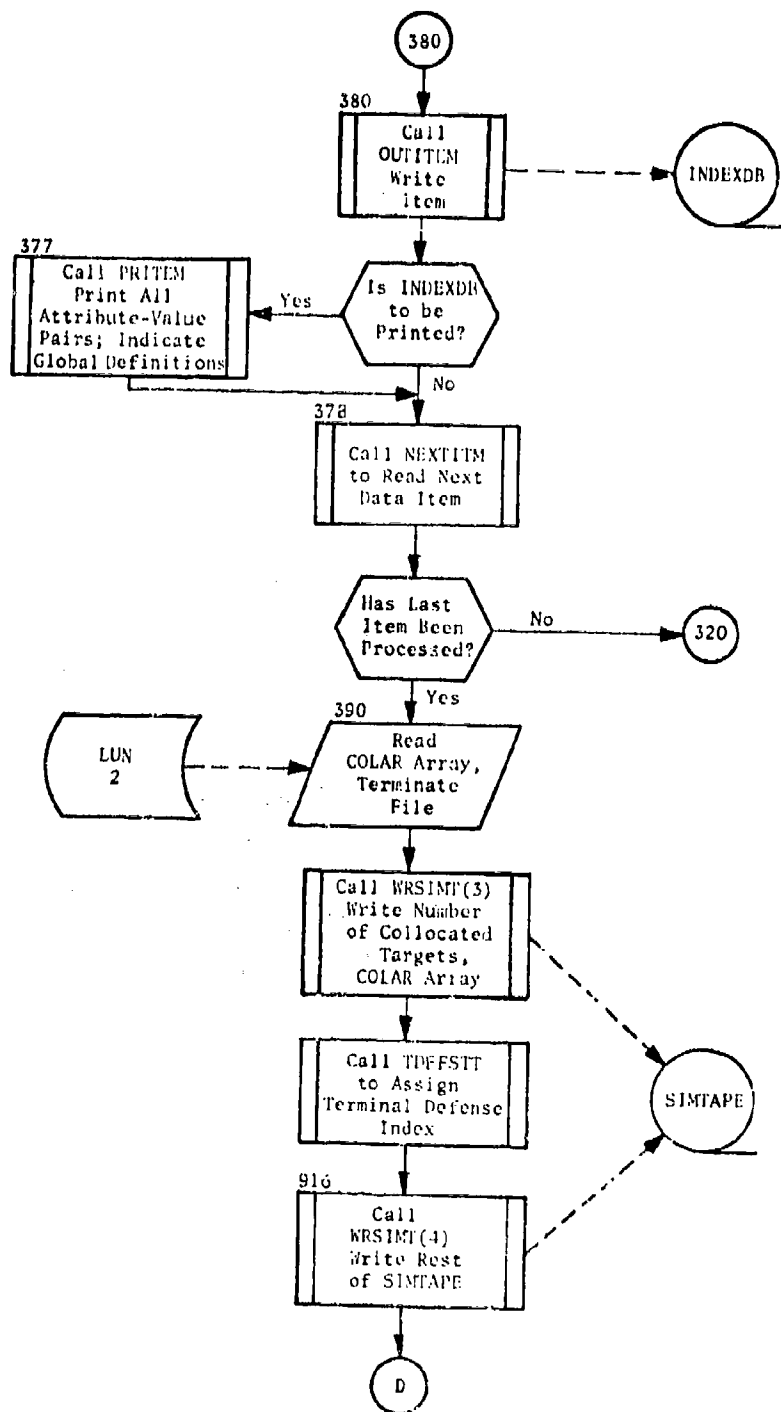


Fig. 111. (cont.)
(Sheet 17 of 18)

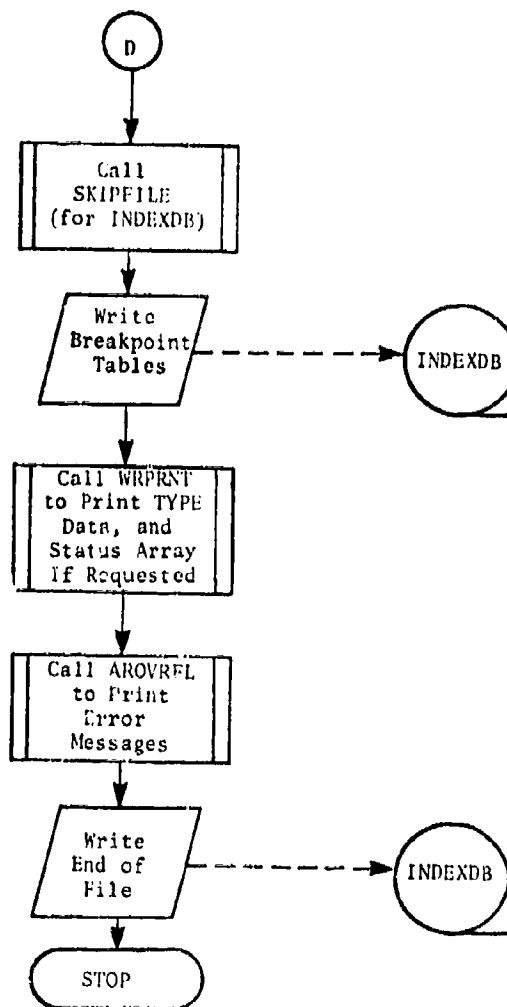


Fig. 111. (cont.)
(Sheet 18 of 18)

SUBROUTINE AROVRFL

PURPOSE: To examine the error message flags which are written during the running of program INDEXER, and to print the errors encountered.

ENTRY POINTS: AROVRFL

FORMAL PARAMETERS: None

COMMON BLOCKS: 9

SUBROUTINES CALLED: None

CALLED BY: INDEXER

Method

Subroutine AROVRFL (figure 112) checks the arrays JCHKFLG and NCHKFLG for the presence of error flags set by program INDEXER and, if present, writes an appropriate error message which indicates a data overflow. If a target class includes too many TYPES, the error message will indicate the number of excess TYPES, the SIDE (RED or BLUE), and ICLASS index involved. Overflows of other selected data arrays will be noted by a message identifying the array and the total number of items read. The latter message will be printed if the upper limit for any of the following is exceeded:

<u>CONSTRAINT</u>	<u>LIMIT (TOTAL)</u>
Target Types (TYPES)	250
Target Complexes (CPX TGT)	4,000
Collocated Targets (COL TGTS)	4,000
Warhead Types (WARHEADS)	50
Payload Types, BLUE (BLU PLDS)	40
Payload Types, RED (RED PLDS)	40
ASM Types (ASMS)	20
Zones (ZONES)	63
Index Numbers (INDEXNO)	12,000

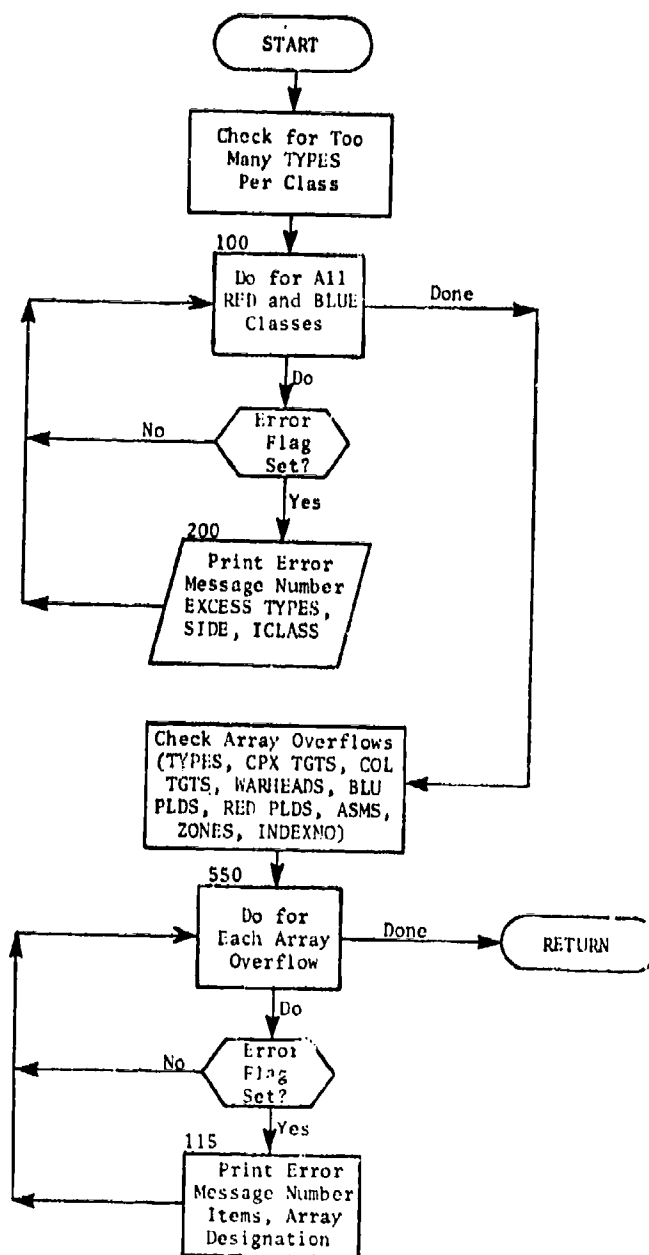


Fig. 112. Subroutine AROVRFL

SUBROUTINE COLOCATE

PURPOSE: To form collocation islands and complex targets for up to 4,000 targets in a given earth segment.

ENTRY POINTS: COLOCATE, COLOCAT (duplicate entry)

FORMAL PARAMETERS: None

COMMON BLOCKS: 1, 2, 3, 4, 7, KEYC, KEY, COMP, KNARRAY, PRNT, TRANS

SUBROUTINES CALLED: ORDER, REORDER, IPUT, IGET, IDXF, FILEHNR

CALLED BY: INDEXER

Method

The initial action of COLOCATE, as shown in figure 113, is to call subroutine ORDER to arrange the latitude array Y by increasing magnitude. Subroutine REORDER then is called to place the arrays IND, X, and Z in the corresponding order (sheet 1).

The search for collocation begins by comparing differences in latitude for consecutive targets in the ordered list, beginning with the first uncollocated target (sheets 2, 3). When a difference which is less than $.35^\circ$ is encountered, the arrays COL, CP, and CLT (see table 14) are tested to find the status of the second target (J). If target J has been collocated previously and is either a member of a complex or a nonmember of the current island, the test for latitude difference between the first target (I) and the next target on the list continues. Otherwise, the actual distance between targets I and J is calculated and compared with the sum of the critical distances for the two targets. Where two targets are found to be collocated, COL and CL are set to 1 for both, and CLT is set to 1 for the second. If the targets are sufficiently close to be members of a complex, CP is set to 1 for both, and the index J is entered in the array LCOMP.

Target I continues to be tested against subsequent targets on the list until a difference in latitude greater than $.35^\circ$ is encountered. The investigation of that target then is considered finished, and CLT(1) is set to zero (sheet 4). If there is an unfinished complex, the next target in the list LCOMP is compared in the same way to find additional members of the complex. The process is repeated until all targets in

the list LCOMP have been investigated, and the complex is complete. Each member is flagged in the array COMP as belonging to a complex (sheet 5). The complex then is assigned the next value of ICOMPLEX (beginning with 1) which, together with the index number of each member, is packed into the array COMPLEX.

The search for members of the current island continues until all targets have been checked for further collocation and complexing (sheet 6). The island then is considered complete. The directed horizontal and vertical distances to each target from the one preceding it in the list (in 50ths of nautical miles) are calculated and packed, together with index number, into the array COLAR (sheets 7 and 8). For the first target in the island, the distances are calculated to it from the last target. In addition, the number of targets in the island (NTA) is included in the data packed for the last target. For all other targets, NTA is set to zero. The array COLAR is written on a scratch file, LUN2; the number of collocated targets and collocation islands processed thus far is accumulated; and the investigation is restarted with the next uncollocated target (COL=0) on the list (sheet 2). When the list is exhausted, control is returned to INDEXER.

Table 14. Description of COLOCATE Arrays

<u>ARRAY</u>	<u>LENGTH</u>	<u>DESCRIPTION</u>
COLO	12,000 (logical)	Is set to 1 if corresponding target belongs to a collocation island
COMP	12,000 (logical)	Is set to 1 if corresponding target belongs to a complex
COLAR	100	Is packed with data required for collocation islands
COMPLEX	4,000	Is packed with data required for complex targets
COL(J)	4,000 (logical)	Is set to 1 if Jth item in array IND belongs to some collocation island
CL(J)	4,000 (logical)	Is set to 1 if Jth item in array IND belongs to current island
CLT(J)	4,000 (logical)	Is set to 1 if Jth item in array IND belongs to current island but has not been checked for further collocation
CP(J)	4,000 (logical)	Is set to 1 if Jth item in array IND belongs to a complex
LCOMP	40	Contains indices J of items in array IND which belong to the complex currently being investigated

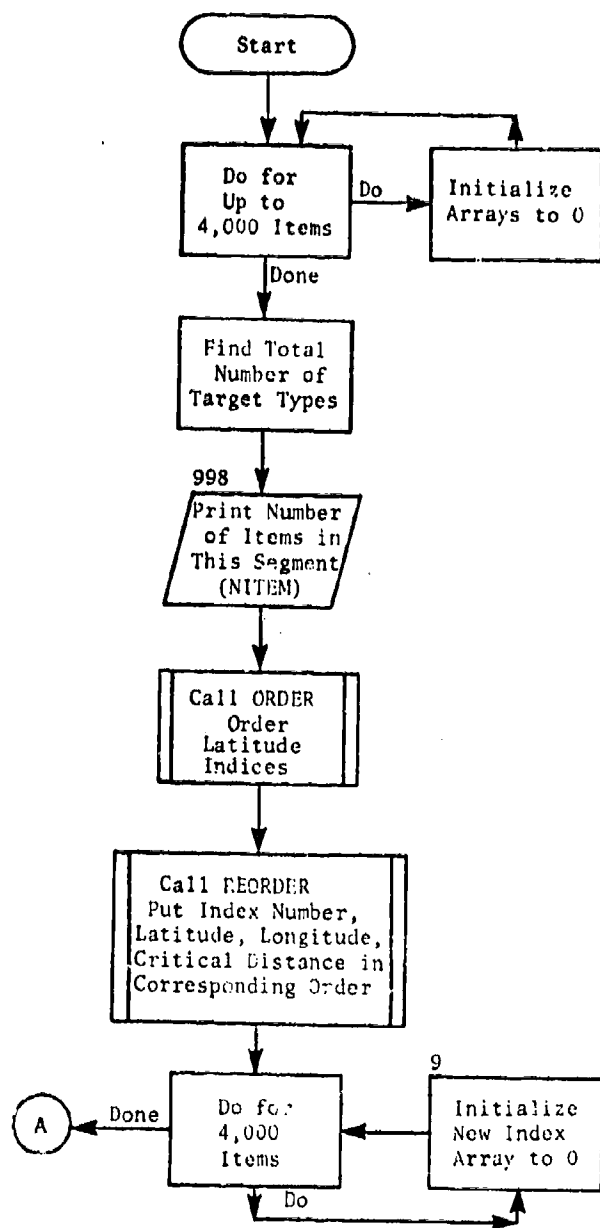


Fig. 115. Subroutine COLOCATE
(Sheet 1 of 8)

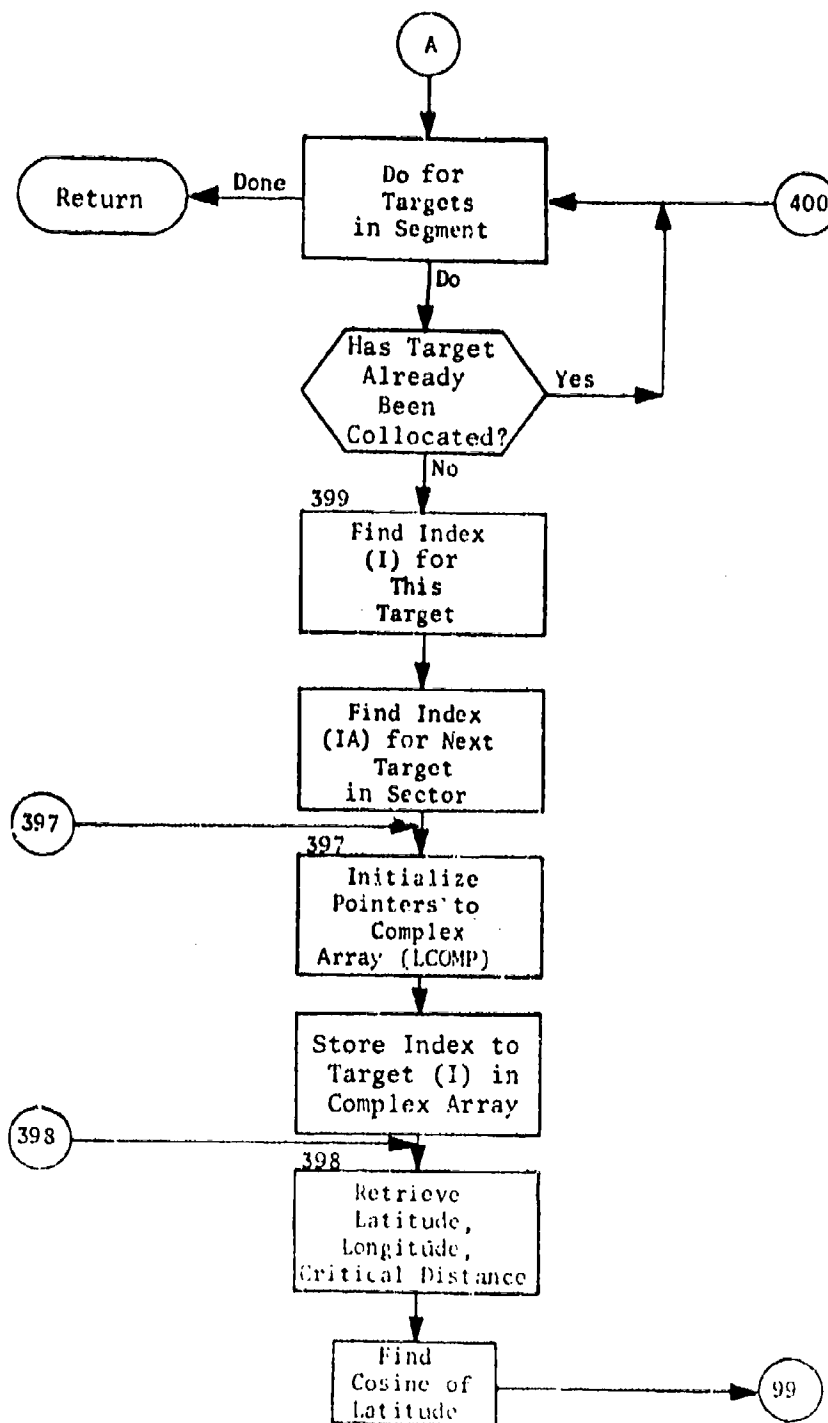


Fig. 113. (cont.)
(Sheet 2 of 8)

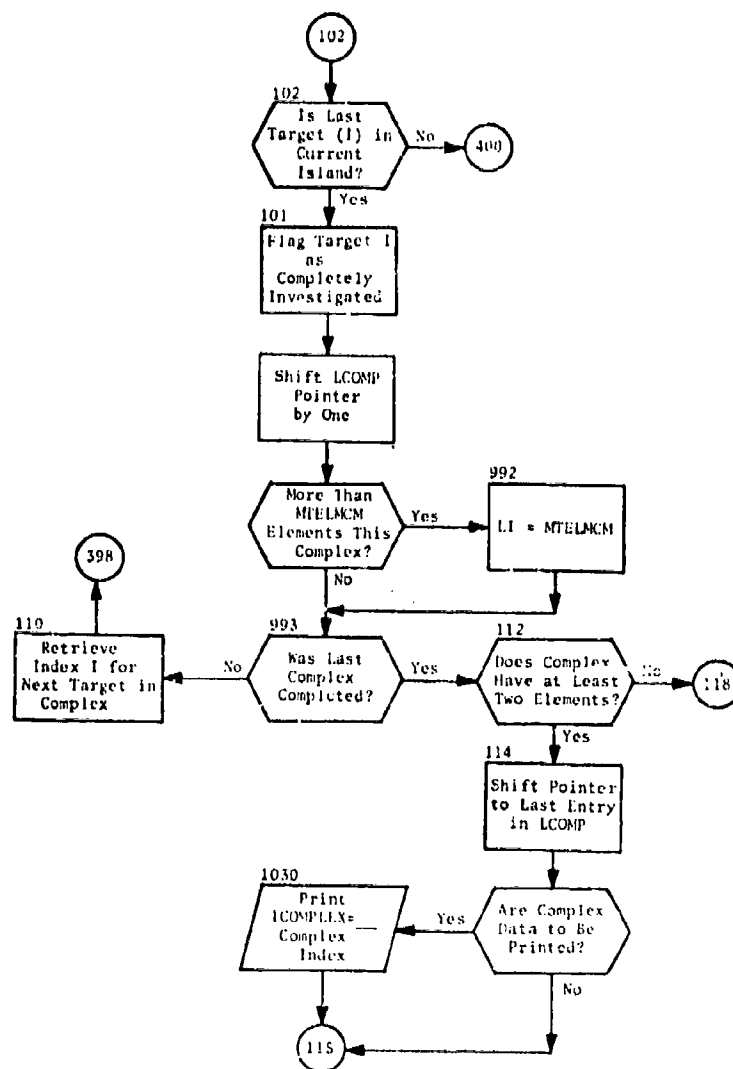


Fig. 113. (cont.)
(Sheet 4 of 8)

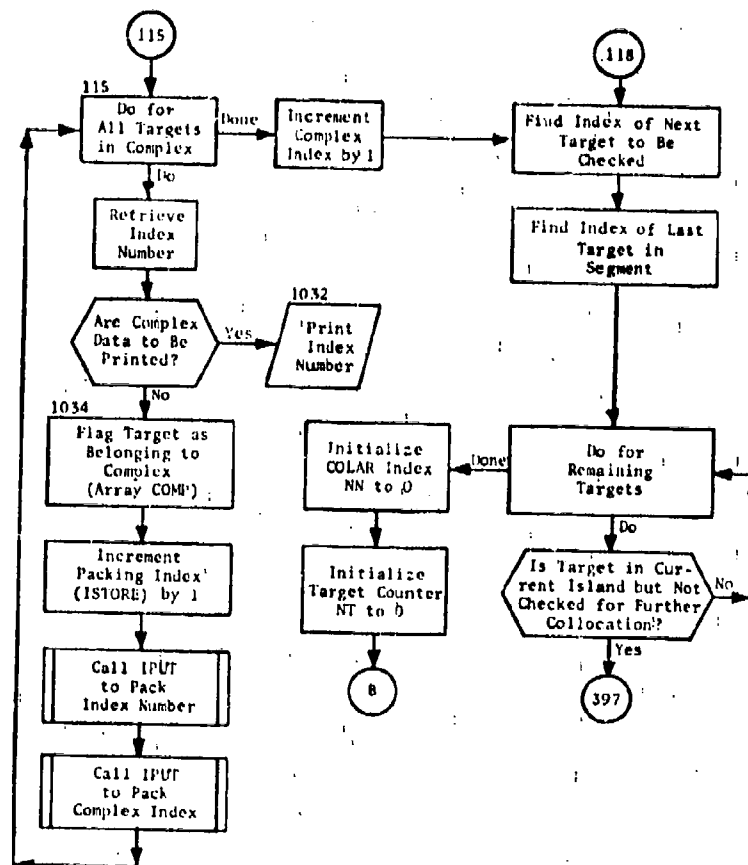


Fig. 113. (cont.)
(Sheet 5 of 8)

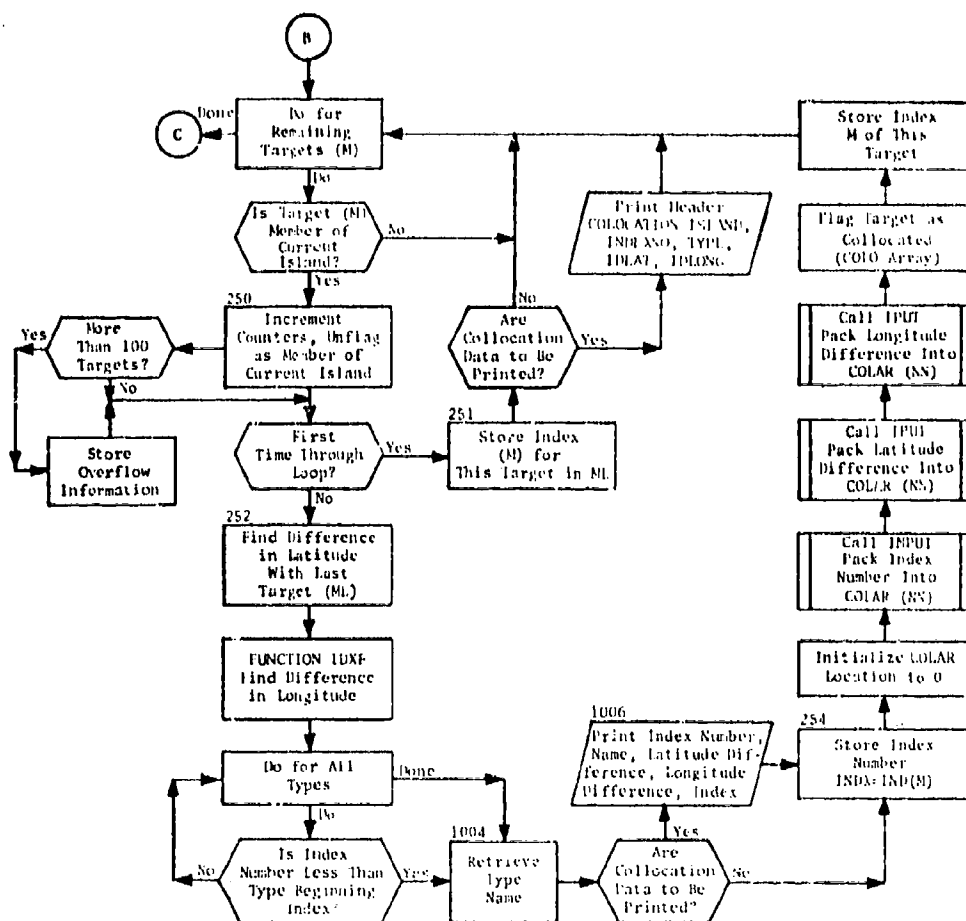


Fig. 113. (cont.)
(Sheet 6 of 8)

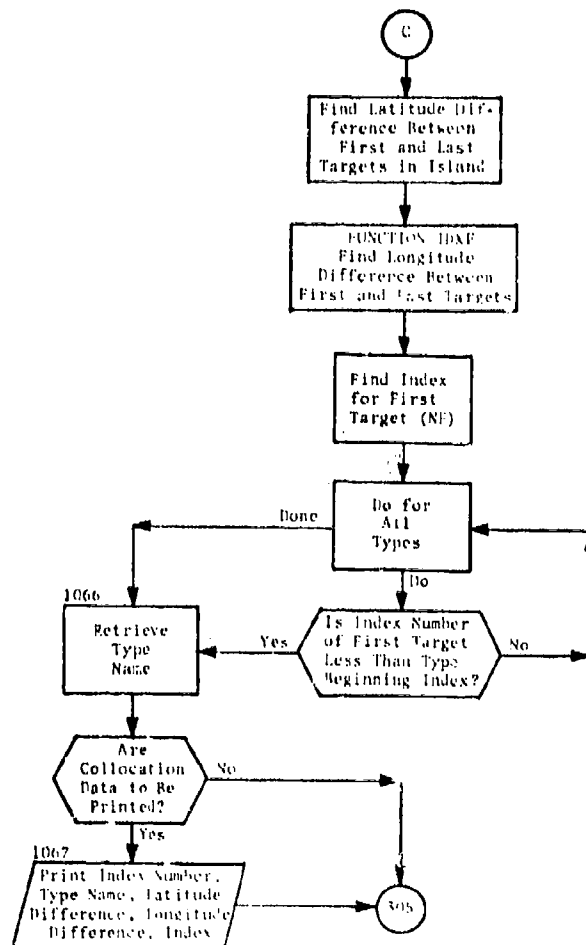


Fig. 113. (cont.)
(Sheet 7 of 8)

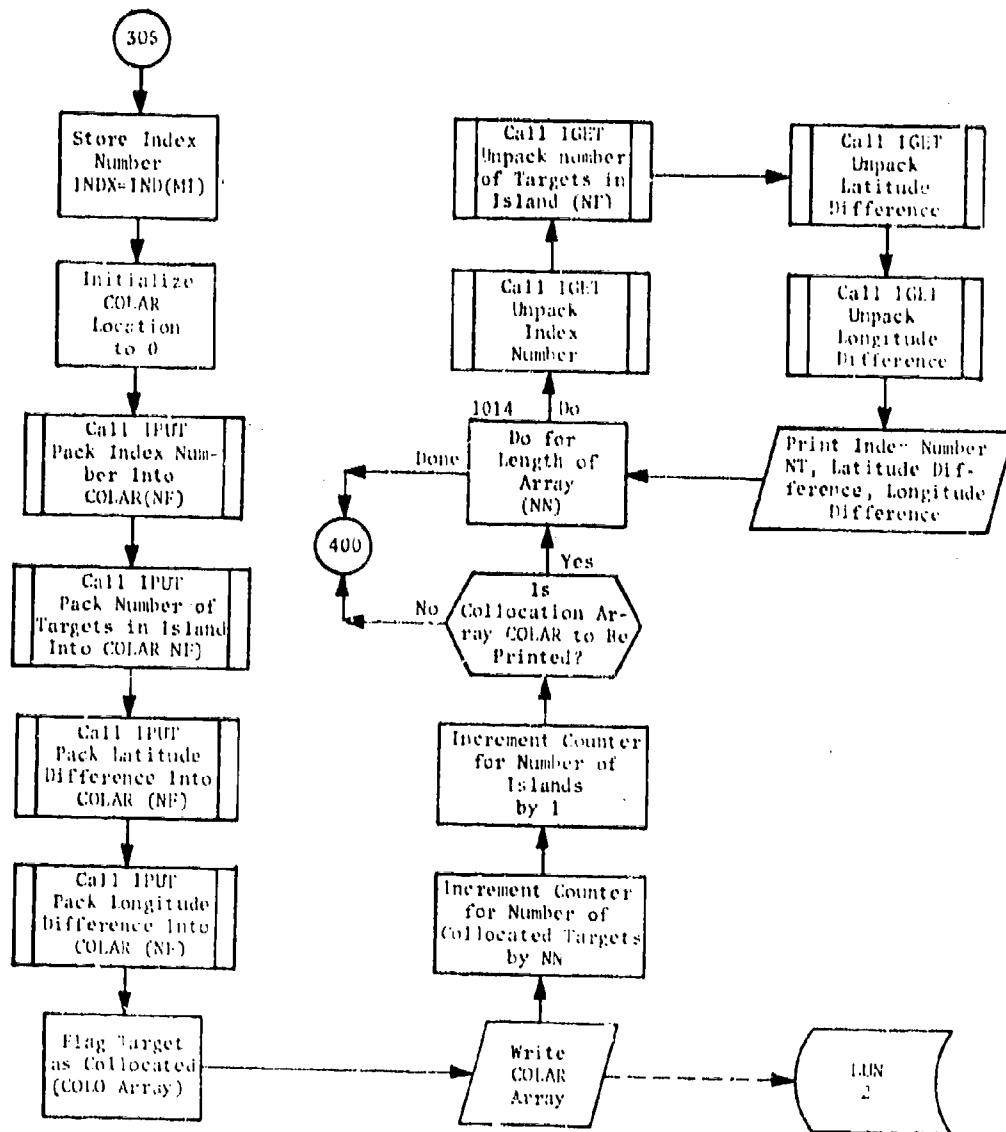


Fig. 113. (cont.)
(Sheet 8 of 8)

SUBROUTINE FINDIT

PURPOSE: To look up the index number INDEXNO of a target
in the collocation array COLAR.

ENTRY POINTS: FINDIT

FORMAL PARAMETERS: None

COMMON BLOCKS: 3

SUBROUTINES CALLED: ABORT

CALLED BY: TDEFSTT

Method

FINDIT receives the length of the array COLAR in ISTORE, and the value of INDEXNO to be matched in ICUR (both in common /3/). It executes a masked equality search through COLAR and normally returns, in ICUR, the index of the word in COLAR which contains the matching INDEXNO. If no match is found, ABORT is called to terminate the run. Subroutine FINDIT is illustrated in figure 114.

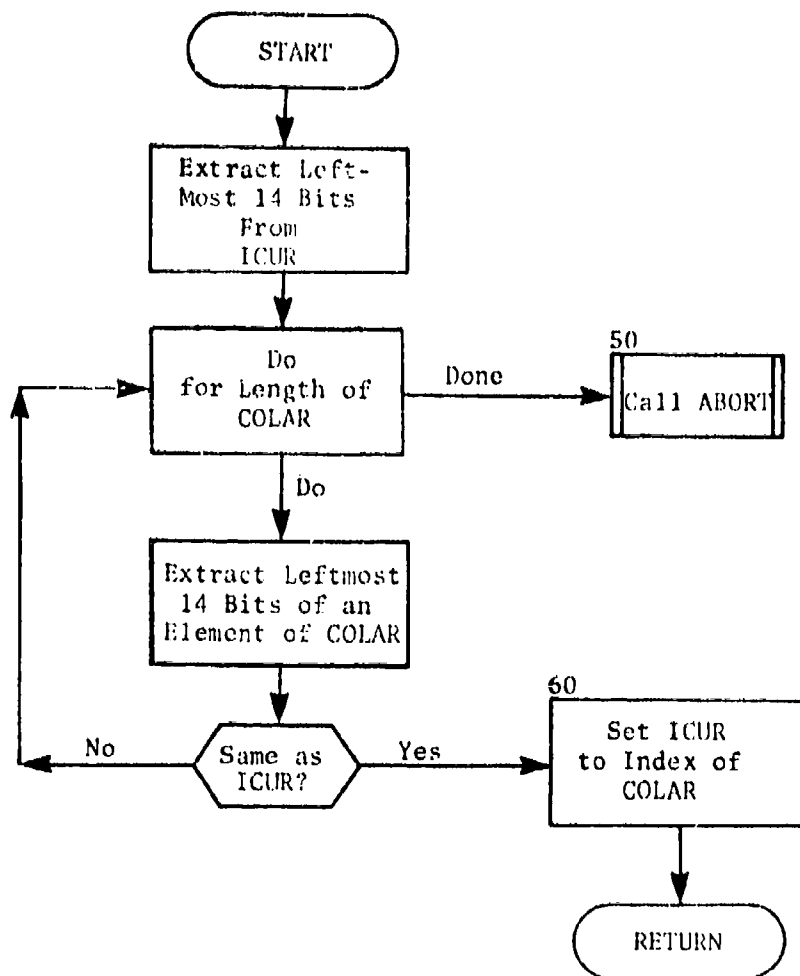


Fig. 114. Subroutine FINDI1

FUNCTION ICPL

PURPOSE: To unpack the index ICOMPLEX from array COMPLEX.

ENTRY POINTS: ICPL

FORMAL PARAMETERS: INDEX - The index number to be located in the
COMPLEX array
N - The number of words in the COMPLEX array
which have been filled

COMMON BLOCKS: 3, KEYC

SUBROUTINES CALLED: IGET

CALLED BY: INDEXER

Method

ICPL begins by masking all information except the five-digit index contained in the parameter INDEX and in the array COMPLEX. The array then is searched until two matching index numbers are encountered. If no match is found, ICPL is assigned the value zero. Otherwise, function IGET is called to unpack ICOMPLEX from the word in COMPLEX at which the search is stopped. Control then returns to INDEXER with this value of ICOMPLEX assigned to ICPL. Function ICPL is illustrated in figure 115.

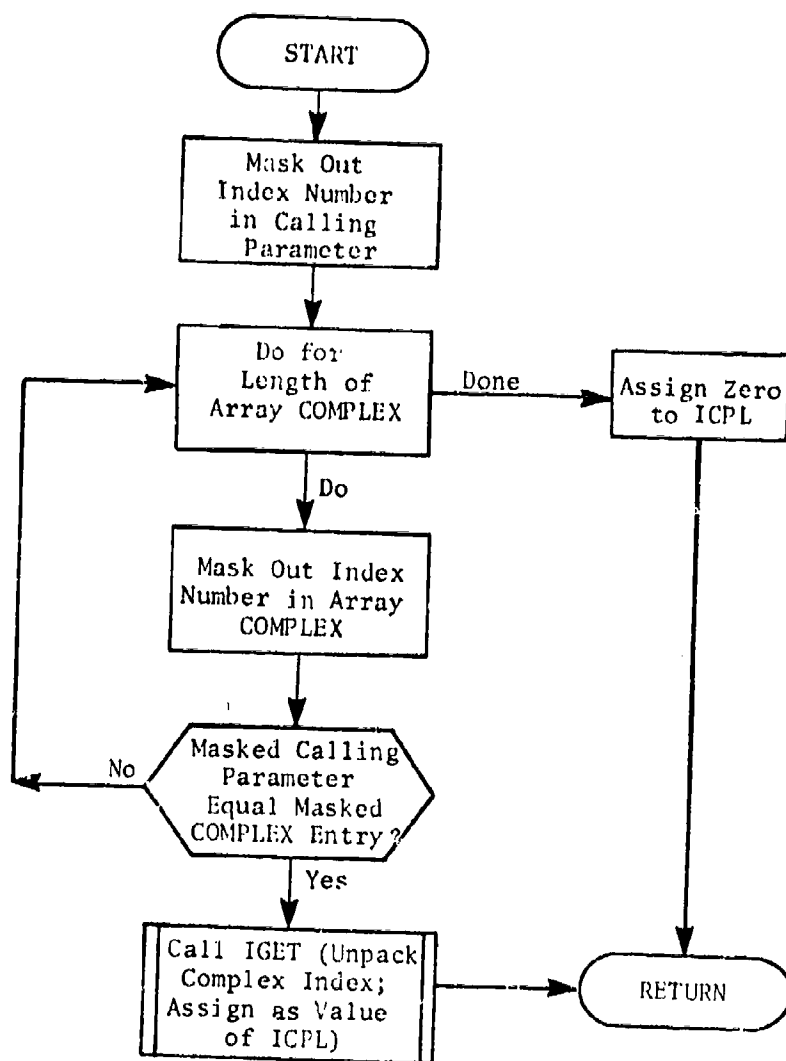


Fig. 115. Function ICPL

FUNCTION IDXF

PURPOSE: To compute the latitudinal distance between two consecutive targets in a collocation island.

ENTRY POINTS: IDXF

FORMAL PARAMETERS: J - Pointer to longitude of first target
K - Pointer to longitude of second target

COMMON BLOCKS: 1

SUBROUTINES CALLED: None

CALLED BY: COLOCATE

Method

The parameters J and K are indices to the array X in common /1/ which contains degrees of longitude. After the corresponding values have been retrieved from X, the difference between them is calculated and changed, if necessary, to represent the shortest distance around the earth and then converted to 50ths of nautical miles. The value is returned to COLOCATE as the value of IDXF. Function IDXF is illustrated in figure 116.

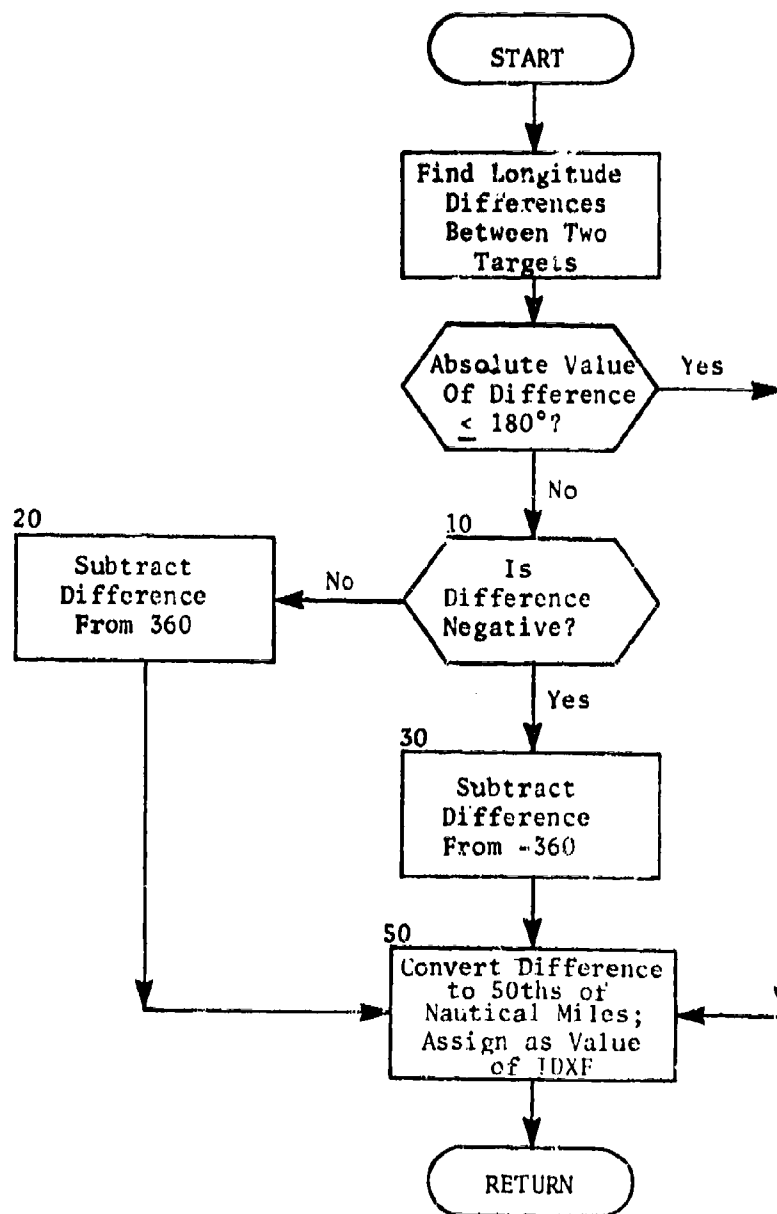


Fig. 116. Function IDXF

SUBROUTINE INITIND

PURPOSE: To data set INDEXER common constants and to clear all common working and storage arrays to zero.

ENTRY POINTS: INITIND

FORMAL PARAMETERS: None

COMMON BLOCKS: AREADAT, EDITAPE, EDITERM, IFTPRNT, ITP, KEY, KEYC, KEYS, MAX, MYIDENT, NAVALTB, NOPRINT, PROCESS, PRNT, RADATA, TRANS, TWORD, WRIT, 1, 2, 3, 4, 5, 7, 9, 10

SUBROUTINES CALLED: None

CALLED BY: INDEXER

Method

INITIND does no computation. Either through data statements or executable statements, it presets constants and arrays to their appropriate values. The principal constants preset in data statements are those which define the maximum limits of the system (common block /MAX/), the constants to determine earth sector boundaries (common block /WRIT/), and certain masks used in packing and unpacking (common block /KEYC/). Subroutine INITIND is illustrated below in figure 117.

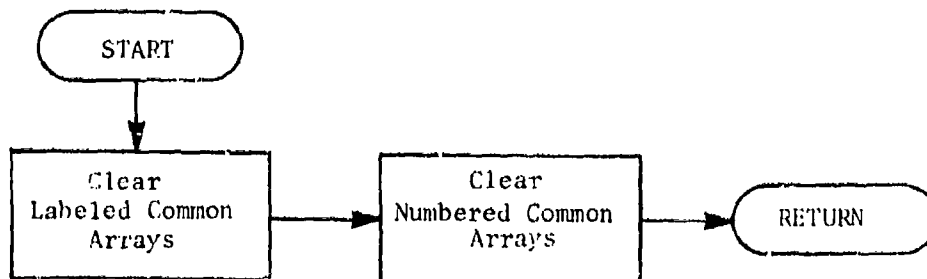


Fig. 117. Subroutine INITIND

SUBROUTINE READIN

PURPOSE: To read the card(s) containing print options and set print controls for INDEXER; to read the cards containing the VLRAID input parameters; and to read the card containing the missile vulnerability parameter.

ENTRY POINTS: READIN

FORMAL PARAMETERS: None

COMMON BLOCKS: PRNT, RADATA, 3

SUBROUTINES CALLED: None

CALLED BY: INDEXER

Method

This subroutine (figure 118) reads the input parameters contained in the execution deck. First, the cards indicating the selected print options are read. These print options are controlled by one or more data cards containing integers between 1 and 15. These integers, which must be right-justified in 10-column fields, control the printed output as follows:

<u>PRINT OPTION</u>	<u>DATA PRINTED</u>
1	Breakpoint tables
2	INDEXDB
3	Data base directory only
4	Type data for the Simulator
5	Collocation array
6	Status array/radar and ABM data
7	Collocation islands
8	Complex targets
9	Reserved for future developments
10	Terminal interceptor, payload, and IDBI tables
11-15	Not used.

Next, the VLRAD input parameter cards are read, and the data are stored in common block /RADATA/.

The final data card read by READIN contains the missile vulnerability parameter established to characterize missile launch sites in their softened condition immediately after lift-off. This parameter is stored in the first word of the CVULN array, and control is returned to INDEXER.

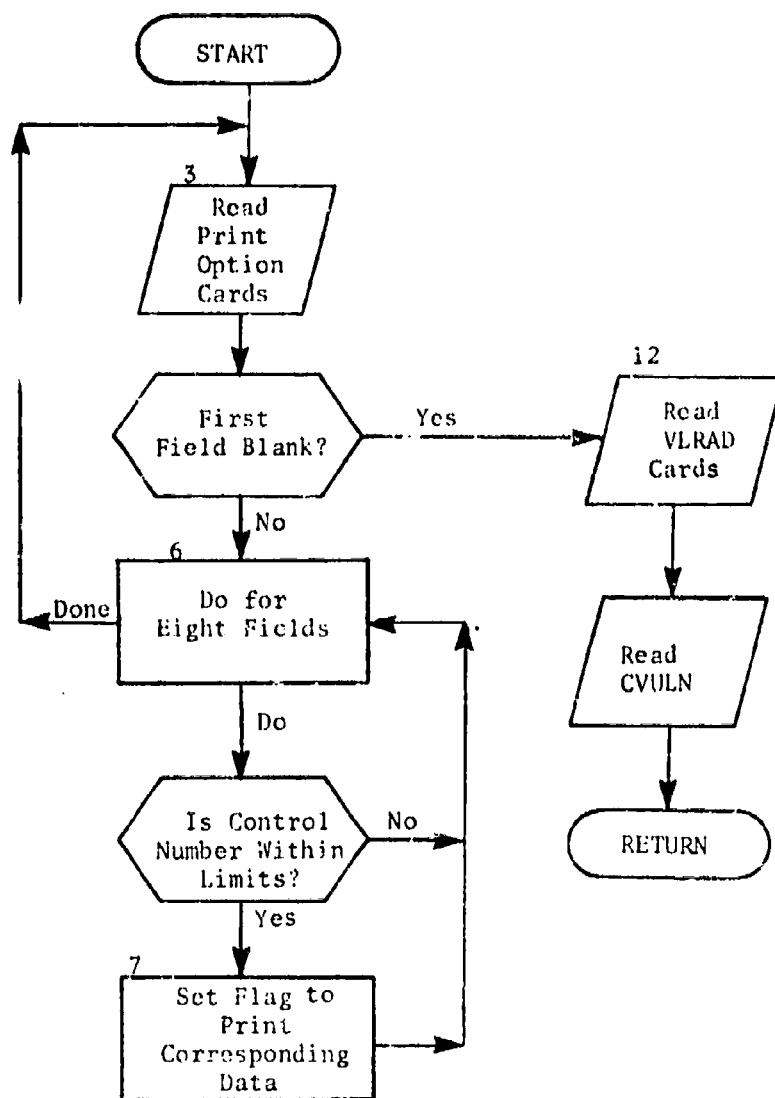


Fig. 118. Subroutine READ1N

SUBROUTINE TDEFSTT

PURPOSE: To make the final assignment of terminal defense indices so that collocation islands are treated as single targets for defense purposes.

ENTRY POINTS: TDEFSTAT, TDEFSTT (duplicate entry)

FORMAL PARAMETERS: None

COMMON BLOCKS: KEY, KEYS, 3, 5, 7

SUBROUTINES CALLED: FINDIT, IPUT, IGET

CALLED BY: INDEXER

Method

TDEFSTT begins by testing the first location in the array ITERM for a positive value of INDEXNO, which indicates that the item has terminal defenses. Whenever a nonpositive value is encountered, control is returned to INDEXER. Otherwise, the array NTINTX is tested to see if the item belongs to a collocation island to which a terminal defense index already has been assigned. This is indicated by negating the terminal defense index assigned to that island and storing it in array NTINTX. Thus, if the value in array NTINTX is found to be negative, it must be the terminal defense index to be assigned to all targets in the island. The index then is reset to its positive value and packed under KTERM in the status array.

If the value in array NTINTX is found to be positive, the item under consideration is tested for collocation. An uncollocated target retains the original terminal defense index 1, which is packed under KTERM in the STATUS array; a collocated target belonging to a collocation island with no previously assigned index is treated in the following manner.

Subroutine FINDIT is called to retrieve the index to the word in array COLAR in which the value of INDEXNO has been packed. The number of targets in the island is then unpacked from the word for use in cycling through all targets in the island. The attribute INDEXNO is unpacked from each consecutive word and tested against the original value of INDEXNO (in ITERM(1)) to see if the cycle is complete. If the two values do not match, the target is checked for terminal interceptors (logical

array LTERM). When terminal interceptors are indicated, the number NTINT is retrieved from array NTINTX(J) and added to the number being accumulated under the original target (NTINTX(I)). If no terminal interceptors are indicated, INDEXNO is added to the list in ITERM, since the target assumes the terminal interceptors of the collocation island and must be so indicated in the STATUS array. As each additional target in the island is encountered, the negative index I replaces the number of terminal interceptors for the target (NTINTX(J)) to indicate that the target already has been processed.

When the cycle through the island is complete, the index of the original target is packed under KTERM in the STATUS array. The next target in the list ITERM then is considered. When the next item on the list is zero, control is returned to INDEXER.

Subroutine TDEFSTT is illustrated in figure 119.

SUBROUTINE VLRADI

PURPOSE: To find the lethal radius of a weapon delivered against a target of a specified vulnerability, and to set FN for use by the calling subroutine.

ENTRY POINTS: VLRADI

FORMAL PARAMETERS:

YIELD	-	Yield of weapon in megatons
NVN	-	Vulnerability parameter of target
HOB	-	Weapon height of burst
FN	-	Parameter specifying shape of damage function

COMMON BLOCKS: RADATA

SUBROUTINES CALLED: EXPF

CALLED BY: INDEXER

Method

NVN is decoded into the appropriate vulnerability number VN, the letter (P or Q), and the K-factor XK. The cube root of the yield is extracted. Then the adjusted vulnerability number AVN is determined by methods described in "Computer Computation of Weapon Radius," B-139-61, Air Force Intelligence Center. FN is set to six or three for P and Q type targets, respectively.

Common block /RADATA/ contains four arrays (for the four combinations of P or Q vulnerability and air-or-surface burst) each of which contains the natural logarithm of the lethal radius (in nautical miles) of a one-megaton burst. The data are at intervals of five vulnerability numbers. Subroutine VLRADI interpolates in the appropriate array to find the logarithm of the one-megaton lethal radius for AVN. The lethal radius of the weapon is then determined by exponentiating and multiplying by the cube root of the yield.

A flowchart for VLRADI is shown in figure 120.

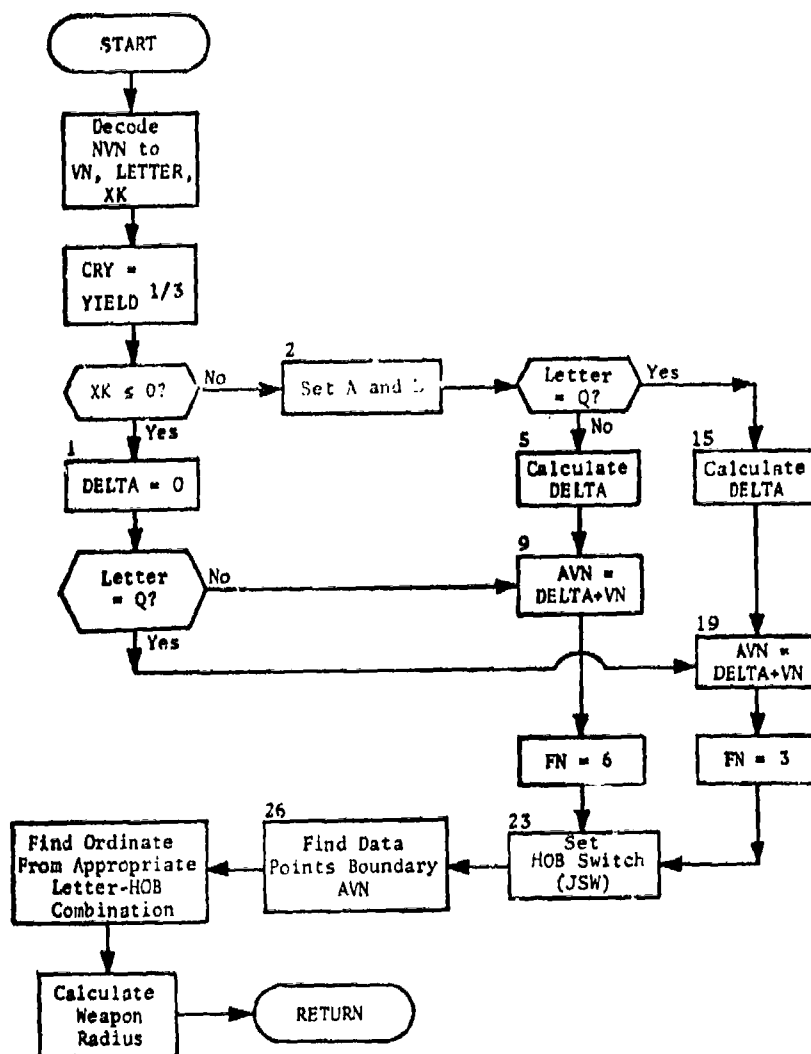


Fig. 120. Subroutine VLRADI

SUBROUTINE WRPRNT

PURPOSE: To execute the prints required when INDEXER print options 4, 6, or 10 are requested.

ENTRY POINTS: WRPRNT

FORMAL PARAMETERS: None

COMMON BLOCKS: AREADAT, IFTPRNT, ITP, KEY, KEYC, KEYS, MAX, MYIDENT, NAVALTB, NOPRINT, RADATA, PRNT, TRANS, TWORD, WRIT, 1, 2, 3, 4, 5, 7, 9, 10, PROCESS, EDITERM, EDITAPE

SUBROUTINES CALLED: IGET

CALLED BY: INDEXER

Method

Subroutine WRPRNT is used to print the contents of several data arrays associated with program INDEXER print options 4, 6, and 10. A description of these print options and the associated print formats is presented in the User's Manual (see Output, Program INDEXER, chapter 2 of Volume I and chapter 3 of Volume II).

As shown in figure 121, WRPRNT examines the print option switches to determine which prints, if any, are required. If no prints are required, control is returned to INDEXER. Print option switch 4 is examined first to determine if the simulation-type data are to be printed. If not, switch 6 is examined to determine if the STATUS array and antiballistic missile (ABM) data are to be printed. As indicated in the flowchart, switch 10 which controls the print of the terminal ABM data is only checked if print option 4 was also selected by the user (i.e., print option 10 can only be used in conjunction with print option 4). Subroutine WRPRNT performs no computation. It merely prints the indicated data arrays.

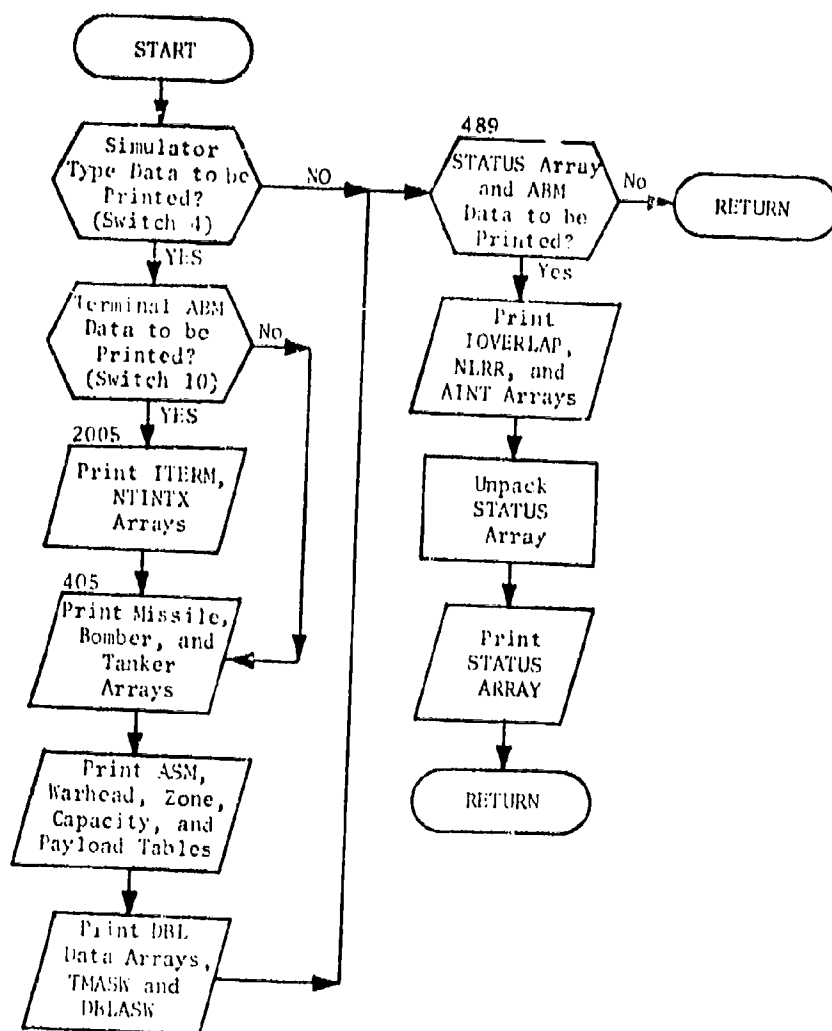


Fig. 121. Subroutine WRPRNT

SUBROUTINE WRSIMT

PURPOSE: To write the Simulator data tape SIMTAPE.

ENTRY POINTS: WRSIMT

FORMAL PARAMETERS: NOP - A flag which indicates the write operation to be performed

COMMON BLOCKS: AREADAT, IFTPRNT, ITP, KEY, KEYC, KEYS, MAX, MYIDENT, NAVALTB, NOPRINT, RADATA, PRNT, TRANS, TWORD, WRIT, 1, 2, 3, 4, 5, 7, 9, 10, PROCESS, EDITERM, EDITAPE

SUBROUTINES CALLED: SETWRIT, WRWORD, WRARRAY, TERMTAP

CALLED BY: INDEXER

Method

The data base information which is required by the Simulator is written onto tape SIMTAPE by subroutine WRSIMT. This information includes the index breakpoint tables; the COLAR array, which contains packed data for collocated targets; the STATUS array, which contains packed data on all potential targets; and several other arrays containing characteristics of weapon types, characteristics of warhead types, and defense capabilities. When called, this subroutine examines the write option flag NOP(1-4) and writes the appropriate data onto the SIMTAPE. Subroutine WRSIMT is illustrated in figure 122.

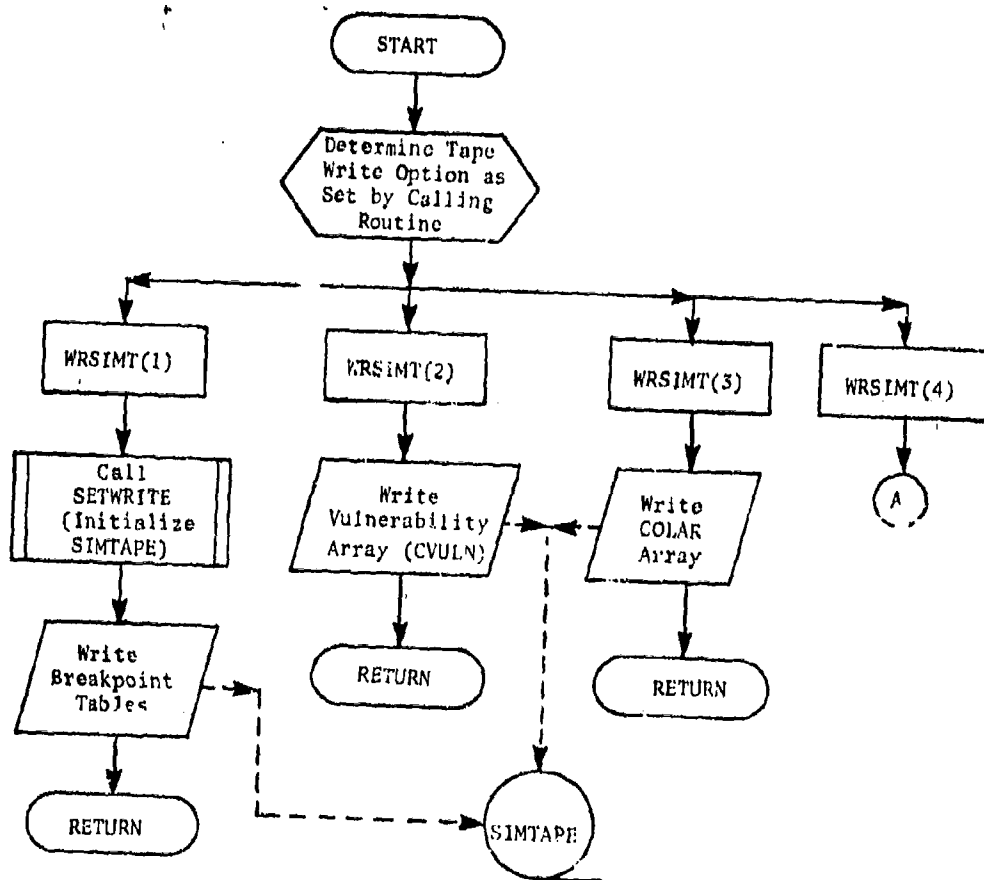


Fig. 122. Subroutine WRSINT
(Sheet 1 of 2)

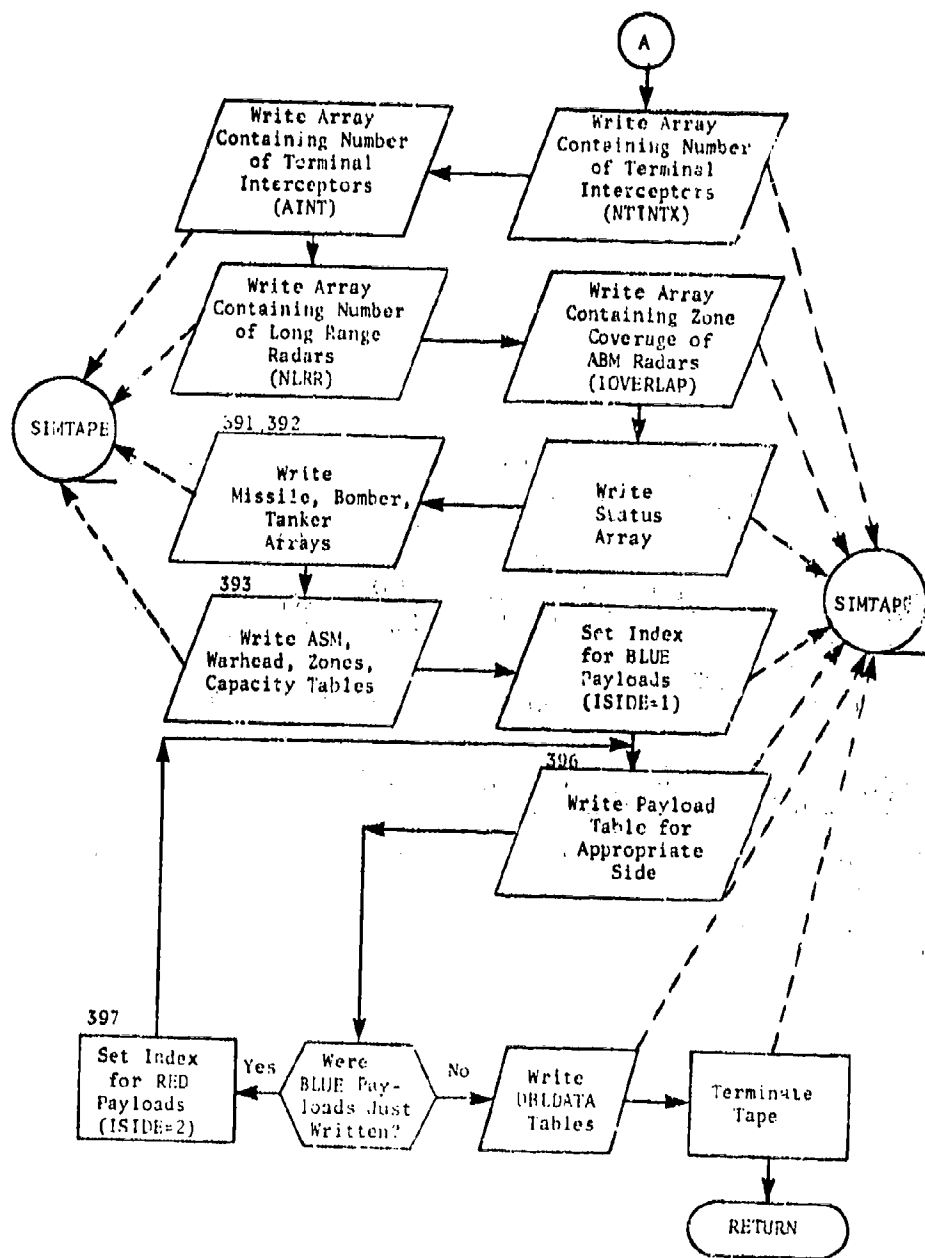


Fig. 121. (cont.)
(Sheet 2 of 2)

CHAPTER 8 PROGRAM BASESUM

<u>PURPOSE:</u>	To summarize the input data base and to print these summaries in tabular form.
<u>ENTRY POINTS:</u>	BASESUM
<u>FORMAL PARAMETERS:</u>	None
<u>COMMON BLOCKS:</u>	DIRECTRY, EDITERM, PROCESS, TABLES
<u>SUBROUTINES CALLED:</u>	INITEDIT, INITAPE*, INPITEM, ITLE, NEXTITEM, PAGESKP

Method

Program BASESUM provides the capability of summarizing the data base contained on the output tapes produced by programs QUIKBASE (QUIKDB tape), BASEMOD (QKMODDB or INMODDB tapes), or INDEXER (INDEXDB tape). The data base tape to be summarized is the only required user input, and there are no output files created by BASESUM.

To summarize the contents of a data base tape, two passes are made through the tape to summarize, in order, the Blue and Red sides. For each class; e.g., Bomber, and each side, a summary table is printed. Within these tables, the columns reflect the types within the class; e.g., B-52 and B-58, and the rows reflect the attributes; e.g., RANGE, defined for the class.

The following conventions apply to the values of the attributes printed in the table. All floating point values are the average value, and all integer and BCD values are the first one encountered in reading the data base tape. If the value changes within any type, the value in the matrix is marked with an asterisk.

Figure 123 depicts the logical flow of this program. As indicated, a flag is set to cause side Blue to be summarized first. Next, the array dimensions are set to correspond to the expected size of the data base. The division at the present time is 25 classes, 200 types, and 100 attributes per class per side. If any of these dimensions is exceeded in processing the data base, an appropriate error message is printed. The directory and the first item are now read in, and the

* See filehandler subroutines.

positions of CLASS, TYPE, and SIDE in the VALUE array are found by comparing these attribute names with the ATTNAM array. As each item is read in, it is examined in several ways. It is skipped if value of the attribute SIDE is not the side currently being processed. If this requirement is met, the class name is checked to determine whether it is a new class and whether storage is available. If so, the new class is stored and the class counter is incremented. Next, the type name is compared to those previously processed. If no match is found and space is available, the new name is stored; the type counter is incremented; and the index to NAMECLAS is stored. If a type name match is found, a check is made to ascertain whether the class is the same. If not, a further check is made to find a name match later in the table. If none is found, control goes to the new type case. At this point in the flow, the number-of-items counter is incremented, since both the type and the class are known. Now all the attribute names defined for this class are examined. The value of the attribute is stored directly for newly defined attribute names or the first appearance of an attribute in a type and class. The procedure differs, however, for attributes previously defined. A logical array ITEMS is examined for a previous change in the value of this attribute if it is not the first item encountered of this class and type. If the present value is equal to the previous value, the next item on the data base tape is processed. If the value is different, that fact is recorded in ITEMS. If the value is integer or BCD, nothing further is required. If it is floating point, the previous value is multiplied by the current number of items minus one, and the present value is then added in.

After the last item has been processed, the floating point values are averaged, and all of the tables for this side are printed out. The entire process is repeated for side Red.

Common Block Definition

Program BASESUM references the following utility routine common blocks, which are described in appendix A of this manual: /DIRECTRY/, /EDITERM/, and /PROCESS/.

In addition, BASESUM uses common block /TABLES/ which contains the arrays described in table 15.

Table 15. Program BASESUM Common Blocks

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
TABLES	NAMECLASS	Class names
	NAMETYPE	Type names
	CLASTYPE	Index of class to which each type belongs
	NAMEATT (equivalenced to NMATT)	For each class, pairs of locations in which the first is the name of the attribute, and the second is its location in the VALUE array
	NUMATT	Number of defined attributes for each class
	VALATT (equivalenced to MVALATT)	Value of each attribute for each type
	ITEMS	Set to 1 if value of attribute has changed; 0 if not
	PARRAY	Print array
	MFORMAT	Format of each print line

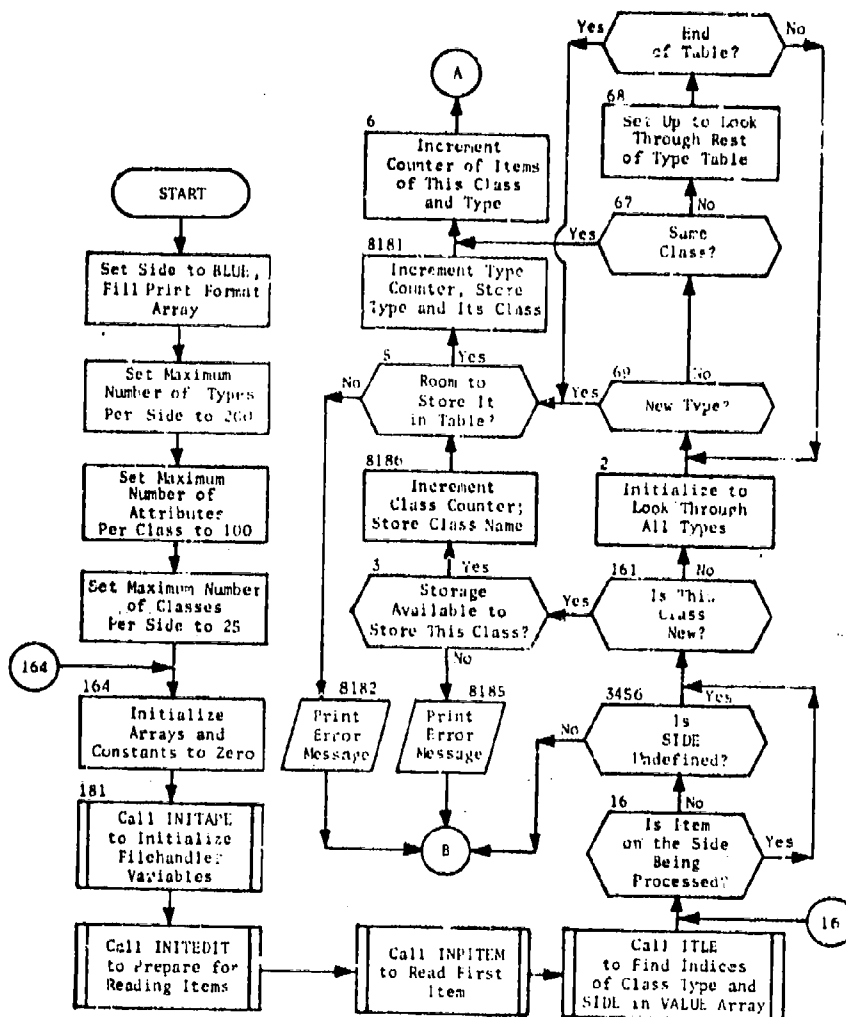


Fig. 123. Program BASISUM
(Sheet 1 of 3)

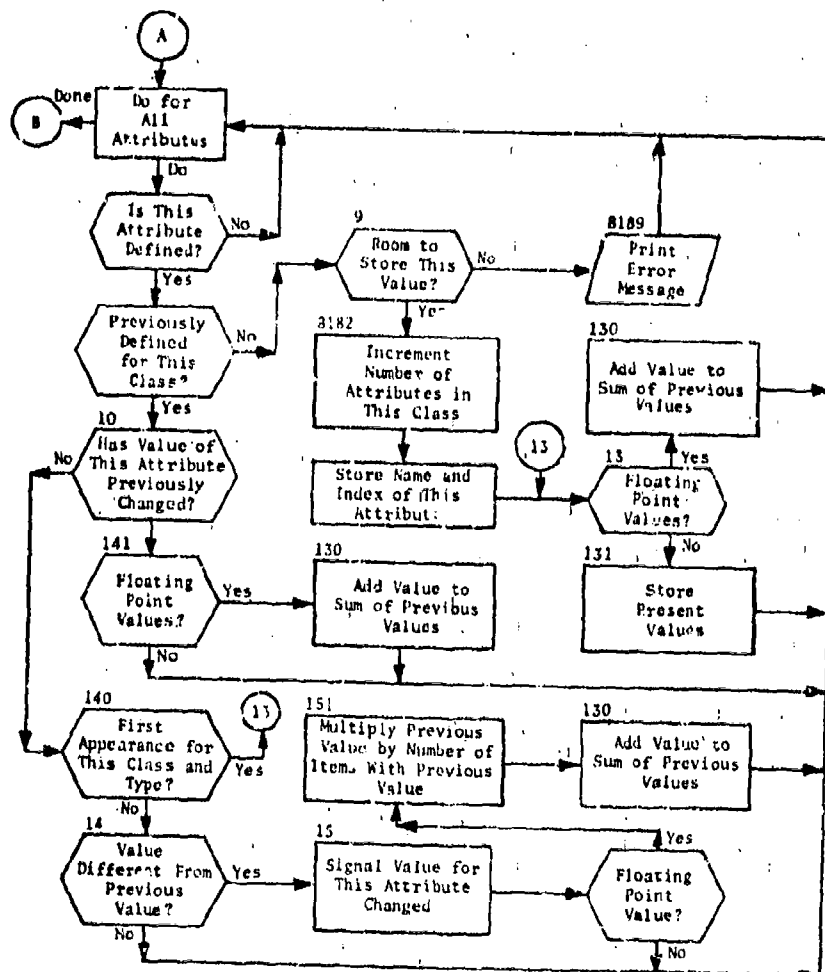


Fig. 123. (cont.)
(Sheet 2 of 3)

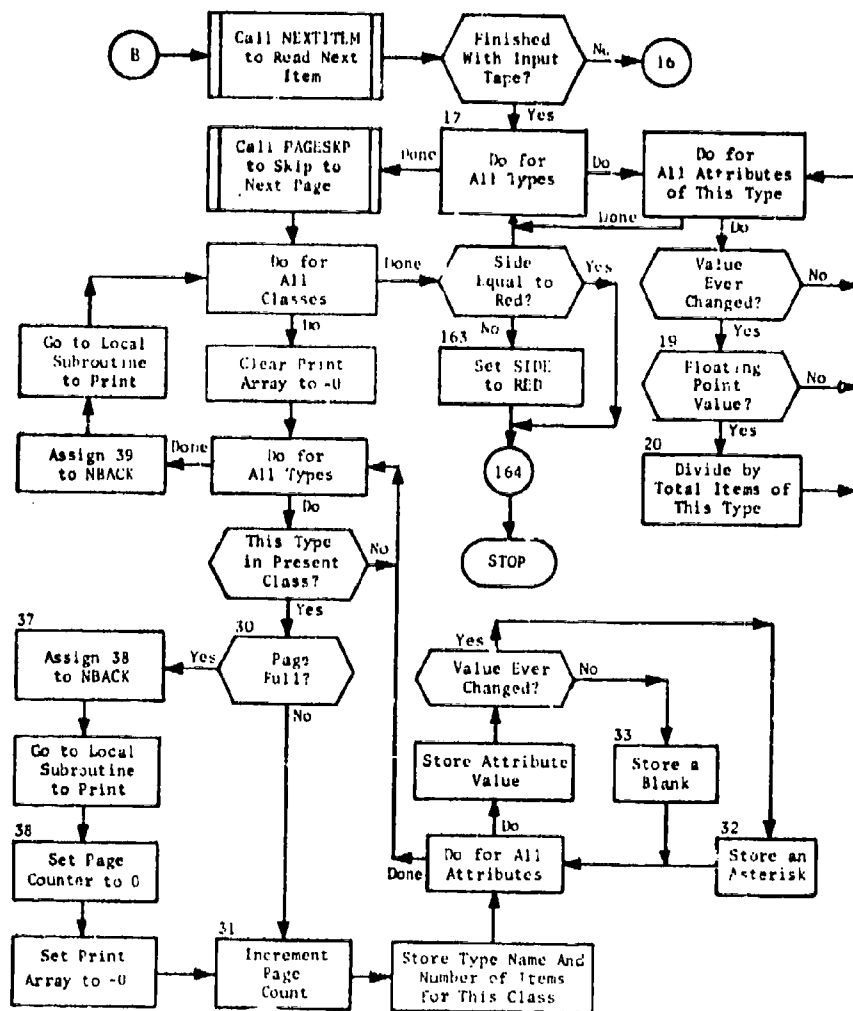


Fig. 123. (cont.)
(Sheet 3 of 5)

APPENDIX A UTILITY ROUTINE COMMON BLOCKS

The following is a description of the common blocks associated with all of the utility programs except for programs OUTFILE and RELOADF. In addition, only those filehandler common blocks used by the calling programs for data transfer are included in this appendix. For a complete list of filehandler common blocks see table 2.

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
A	R	Interpolation calculation matrix
B	S	Temporary interpolation matrix
BNKBND	IAMLOW	Lower storage limit in both banks
	IAMHIGH	Upper storage limit in both banks
C	SOL	Solution vector for interpolation calculation
DATA	X	Storage area for file data
DATPK	MASK	Table of masking variables
	ISHTAB	Table of shifting variables
DIRECTRY		Contents of directory
	IDEF	Index of last defined attribute in tables
	LASTLIST	Index of last entry in LISTVALS
	NDINDIR	Maximum size of tables
	NDIMLIST	Maximum size of LISTVALS

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
DIRECTRY (cont.)	ATTNAME	Hollerith name of attribute
	IFORMAT	Input/output conversion format (FORTRAN) for attribute values
	ICODE	Code specifying type of values and method of checking
	DEFAULT } IDEFAULT }	Undefined value for attribute
	N1 } FN1 }	Minimum allowed value (range check) or index of beginning of allowed list in LISTVALS (list checking)
	N2 } FN2 }	Maximum allowed value (range check) or index of end of allowed list in LISTVALS (list checking)
	LISTCHEK	Set to TRUE for list checking; FALSE for range checking
	GLOB	TRUE when global definition in force; FALSE otherwise
	LISTVALS	Contains allowable values for list checking
EDITAPE		Data base tape editing information
	INTP	Logical tape number of input data base
	NOUT	Number of output tapes
	ITOUT	Logical tape number of output data bases
	JOUT	Logical tape number of current output tape
EDITERM		End of data base tape signal
	ISWTERM	Set to 1 if not end of tape; set to 2 if end of tape
ERRCODE	KABORT	Abort condition code
	KWARN	Warning condition code

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
ERRMESS	IABORT	Abort message
	IWARN	Warning message
ERRORM		Error signals
	JERR	Logical unit number for output error messages
	IERSW	Set to 1 if no errors detected; to 2 if errors are found
FILABEL	INIDENT	Eight letters of (input) file name; read from file
	INRUNNO	Run number; read from file
	INDATE	Date generated; read from file
	INFORM	Format; read from file
	INSECR	Security; read from file
	INTIME	Time generated; read from file
	INLNTH	Length of file
	INCOMM	Five words available for user comments
FILEIN	NAME	Logical disk file name
IFTPRNT	IFTPRNT	Controls debug printout
IREC	IREC	Number of records read from current file
ITP	ITP	Tape currently being used (integer value 1 to 10)
MPRTOPT		Print control
	MPRTOPT	Set to 1 to print directory card images; 0 otherwise

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
MYIDENT	MYIDENT	Tape identifier
MYLABEL	MYFORM	Format designation
	MYSECR	Security designation
	MYLNTH	Maximum file length (words)
	MYCOMM	Comments, if any
NOPRINT	NOPRINT	1 to print tape label
OUTFILES	IOUTDEC	Logical tape number on which DECLARES writes the modified FORTRAN source program (source code on this tape is subsequently compiled)
	ILIST	Logical tape number on which the input source program is printed
	IBODY	Not used
POLITE	S1	First point latitude
	T1	First point longitude
	S2	Second point latitude
	T2	Second point longitude
	FACTOR	Fraction of distance to be interpolated
	SR	Interpolated point latitude
	TR	Interpolated point longitude
PROCESS		Stores item information
	NI	Number of attribute-value pairs in incoming item
	NV	Twice NI
	NC	Index of a changed attribute

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
	INITEM	Alternate attribute indices and values
	VALUE	Current value of all attributes
	DEF	TRUE if attribute currently defined; FALSE otherwise
	LGLOB	TRUE if attribute currently globally defined; FALSE otherwise
PRNTCOMM		Printing information
	SAMEARRY	Set 1 if attribute defined for item; 0 otherwise
	DEFINED	Unused
	INDXOTHR	Indices of already defined attributes
	PAGEDATA	Attributes and values changing on page
	COMATNAM	Names of attributes common to page
	COMATVAL	Values of attributes common to page
	ICOLSAVE	Number of columns on page
PRTOPT		Print control
	NPRTOPT	Set to 1 to print item card images; 0 otherwise
TODAY	NOWRUNO	Current run number (currently not used)
	NOWDATE	The date of the run; written on file label as 6th word
	NOWTIME	The time of the run; written on file label as 13th word
TWORD	TWORD	Word where next data stream word is placed
XPRT		DECLARES print control for listing input source deck; ALL if prints required; otherwise set to NONE

APPENDIX B
QUICK ATTRIBUTE NAMES AND DESCRIPTIONS

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
ABRATE	Probability of aircraft in-flight abort per hour of flying time
ADBLI	ALERTDBL probability for initiative attack
ADBLR	ALERTDBL probability for a retaliatory attack
ADEFCMP	Area ballistic missile defense (BMD) component index (radar or missile launch site)
ADEFZON	Area ballistic missile defense (BMD) zone number
AGX	Offset X-coordinate of AGZ (fiftieths of nautical miles)
AGY	Offset Y-coordinate of AGZ (fiftieths of nautical miles)
AHOB	Actual height of burst of weapon (air or ground)
ALERTDBL	Probability of destruction before launch (DBL) of alert delivery vehicle (missile or bomber)
ALERTDLY	Delay of alert vehicle before commencing launch (hours)
AREA	Area of a bomber defense ZONE (millions of nautical miles ²)
ASMTYPE	Air-to-surface missile type
ATTRCORR	Attrition parameter for a bomber corridor (probability of attrition per nautical mile)
ATTRLEG	Attrition parameter for each route leg in bomber sortie (probability of attrition per nautical mile)
ATTRSUPP	Amount of original attrition that remains after defense suppression

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
AZON1	First area defense zone covered by a BMD long-range radar
AZON2	Second area defense zone covered by a BMD long-range radar
AZON3	Third area defense zone covered by a BMD long-range radar
BCODE	Code indicating the outcome of a simulated bomber event
BENO	Bombing encyclopedia number
BLEGNO	Index to boundary line segment
CATCODE	Category Code as reflected in Joint Resource Assessment Data Base (JAD)
CCREL	Regional reliability of offensive command and control (probability)
CEP	Circular error probable (CEP), delivery error applicable to bomber and missile weapons (nautical miles)
CLASS	Class name assigned identify sets of TYPES in data base
CLASST	Target CLASS
CNTRYLOC	Country code for country where item is located
CNTRYOWN	Country code for country which owns the item
CNTYLOCT	Target country code for country where the target is located
CNTYOWNT	Target country code for country which owns the target
CODE	Outcome code for a general event used in simulation
CPACTY	Capacity of a bomber recovery base (number of vehicles)

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
DATEIN	Earliest date in inventory (year)
DATEOUT	Latest date in inventory (year)
DEFRANCE	Typical range of interceptors at defense bases near a corridor (nautical miles)
DELAY	Delay time (e.g., launch delay time) (hours)
DELTA	Time interval between successive vehicle launches from the same base (missile or bomber) (hours)
DESIG	Target designator code; e.g., AB100, which uniquely identifies each target element included in the data base
DGX	Offset X-coordinate of desired ground zero (DGZ) (fiftieths of nautical miles)
DGY	Offset Y-coordinate of DGZ (fiftieths of nautical miles)
DHOB	Height of burst of weapon (0-ground, 1-air)
EFECSN1 } EFECSN2 }	Attributes assigned to fighter interceptor units (ICLASS = 5 in the data base): the value EFECSN1 or EFECSN2 is assigned to the attribute EFFECTNES depending on value of BASEMOD input parameter POSTURE (if POSTURE=1, EFECSN1 is used; otherwise EFECSN2 value is assigned)
EFFECTNES	Air defense capability (arbitrary scale) established by user to indicate relative effectiveness of air defense command and control installations and fighter interceptor bases
EVENT	Index to event type
EVENTN	Index to type of event which did not occur
FFRAC	Fission fraction (fission yield/total yield)
FLAG	Numeric code (1 through 9 permitted) used to impose restrictions on the allocation of weapons within QUICK

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
FLTNO	Flight number for a sortie
FUNCTION	Operational application code for a weapon system (e.g., ICBM)
EVALH1	Fraction of value of target in first hardness component
EVALT1	Fraction of target value that disappears by T1 (percent)
EVALT2	Fraction of target value that disappears by T2 (percent)
H1	First hardness component of a target (VULN)
H2	Second hardness component of a target (VULN)
HILOATTR	The ratio of the low-altitude attrition rate to the high-altitude rate (decimal fraction)
IALERT	Alert status; 1 = alert, 2 = nonalert
IALT	Altitude index (1 = high, 0 = low)
IATTACK	Selection index for preferential area BMD; 1 forces target selection for defense.
ICLASS	Class index assigned for game
ICLASST	Target class index
ICOMPLEX	Complex index
ICORR	Bomber corridor index number assigned in program PLANSSET: <ol style="list-style-type: none"> 1 - Tactical (FUNCTION=TAC) aircraft corridor (TYPE name DUMMY in the data base) 2 - Naval attack corridor (TYPE name NAVALAIR in the data base) used by bomber units with PkNAV greater than zero >2 - Other corridors used by long range bombers (FUNCTION=LRA)

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
IDBL	Index to data tables for time-dependent destruction before launch probability
IDUD	Dud warhead indicator; assigned to weapons which arrive at the target but fail to detonate; 1=dud warhead
IGIW	Indices of General Industrial Worth (IGIW) (dollars)
IGROUP	Group index assigned for weapon grouping during game
IMIRV	Identifying index for system with multiple independently targetable re-entry vehicles
INDEXNO	Index of a data base item (potential target) used during processing to identify the item
INDV	Vehicle index within base
INTAR	Target index (corresponds to INDEXNO)
IPENMODE	Penetration mode; 1 = aircraft uses penetration corridor, 0 = penetration corridor not used
IPOINT	Index to a geographic point
IRECMODE	Recovery mode; 1 = aircraft should plan recovery, 0 = aircraft recovery not planned
IREFUEL	Bomber refueling code
IREG	Index to identify a geographic region
IREP	Reprogramming index (capability of missile squadron)
ISITE	Site number
ITGT	Target index number assigned by Plan Generation subsystem
ITIME	Index to time periods in time dependent DBL data tables

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
ITYPE	Type index assigned for game
ITYPET	Target type index
IVULN	Index to vulnerability number table
IWTYP2	Second warhead type
JTYPE	Type index within class
JTYPET	Target type index within class
KORSTYLE	Parameter to adjust mode of corridor penetration
LAT	Latitude (degrees)*
LEGNO	Index to line segment
LINK	The index of a leg linked to the current point
LONG	Longitude (degrees)*
MAJOR	Major reference number as reflected in the Joint Resource Assessment Data Base (JAD)
MAXFRACV	Maximum value of weapon resources to be used relative to target value (in processing MAXCOST=MAXFRACV)
MAXKILL	Desired maximum damage expected for a target
MINKILL	The required minimum damage established for a target

* Latitude and longitude are carried internally in the QUICK system in the following format:

North latitude	0. (equator) to +90. (North Pole)
South latitude	0. (equator) to -90. (South Pole)
East longitude	180. to 360. (Greenwich Meridian)
West longitude	0. (Greenwich Meridian) to 180.

These attributes may be input in either the above format or in standard degree, minute, second, direction format.

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
MINOR	Minor reference number as reflected in JAD to identify an item
MISDEF	Number of terminal ballistic missile interceptors for a target
MVA	Manufacturing value added (MVA); indicates the amount of value added by manufacture within a specific area (expressed in U.S. dollars)
MWHDS	Number of missile warheads penetrating area defenses to terminal defense
NADBLI	NALRTDBL for initiative attack
NADBLR	NALRTDBL for retaliatory attack
NAINT	Number of area ballistic missile interceptors at an interceptor launch base
NALRTDBL	Probability of destruction before launch (DBL) of non-alert vehicle
NALRTDLY	Delay of non-alert vehicle before commencing launch (hours)
NAME	Arbitrary alphameric descriptor for any item included in the data base
NAREADEC	Number of decoys per independent re-entry vehicle for area BMD
NASMS	Number of ASMs carried by a bomber
NCM	Number of countermeasures carried by vehicle
NDECOYS	Number of decoys on a bomber or number of decoys per independent re-entry vehicle for terminal BMD
NDET	Number of warheads detonating in current event
NEXTZONE	The adjacent zone to a side of a defense zone
NMPSITE	Number of missiles per site

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
NOALERT	Number of vehicles on alert at a base
NOBOMB1	Number of first bomb type carried by vehicle
NOBOMB2	Number of second bomb type carried by vehicle
NOINCOM	Number of delivery vehicles in commission
NOPERSQN	Number of weapon vehicles per squadron
NOPERSQ1 } NOPERSQ2 } NOPERSQ3 }	Attributes used in program BASEMOD to compute the value of the attribute NOPERSQN for bomber units; numbers 1, 2, and 3 specify surprise, initiative, and retaliatory attack plans respectively
NPEN	Number of warheads penetrating in current event
NTARG	Number of targets in missile launch event
NTINT	Number of terminal BMD interceptors at target
NWHDS	Number of warheads per independent re-entry vehicle (missiles)
NWPNS	Number of weapons in a group
NWTYPE	Warhead type
PARRIVE	Probability of bomber arrival in current event
PAYLOAD	Index which identifies entire weapon and penetration aid complement on a vehicle
PDES	Probability that launch failure destroys missile
PDUD	Probability a warhead will fail to detonate
PEN	Penetration probability for a weapon
PFPF	Probability of failure during powered flight (missiles)
PINC	Probability that a missile is in commission

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
PKMIS	Probability a missile fails to penetrate terminal defense
PKNAV	Single shot kill probability of a weapon against a naval target (a value greater than zero restricts weapon use to naval targets)
PLABT	Probability of vehicle launch abort
PLACE	Index to geographic location of an event
PLACEN	Index to geographic location of an event which did not occur
POP	Population (cities) (thousands)
POSTURE	Force readiness condition
PRABT	Probability of refueling abort
PRIMETAR	Prime target flag; 1 signifies priority target in a complex
PSASW	Destruction before launch probability assigned a weapon for a specified time period
RADIUS	Size descriptor for area targets (nautical miles)
RANGE	Vehicle range (nautical miles)
RANGEDEC	Range decrement for low-altitude aircraft flight (high range/low range)
RANGERE	Range (nautical miles) of bomber with refueling
REL	Reliability - probability that weapon system will arrive at target given successful launch
RESERVE	Technique used to remove certain targets from weapon allocation when RESERVE = 0
SIDE	Item side name, currently either "RED " or "BLUE"

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
SITENO	Site number (currently for individual missile sites)
SPDLO	Speed at low altitude (knots)
SPEED	Speed (knots)
SQNNQ	Squadron number
T1	Time of departure of first value component of a target
T2	Time of departure of second value component of a target
T3	Time of departure of third value component of a target
TAIM	Number of aim points perceived by terminal defense in current event
TARDEFHI	Level of local bomber defense at high altitude*
TARDEFLO	Level of local bomber defense at low altitude*
TASK	Target task code indicating targeting priority
TGTSTAT	Indicates target status as dynamic or nondynamic; in simulation status (alive/dead) is maintained for dynamic targets
TIME	Game time at which event occurred (hours)
TYMEN	Time planned for event which did not occur (hours)
TNDEL	Mean delay time to relaunch after a nondestructive aircraft abort (hours)

* Arbitrary units scaled by user-input parameter in Plan Generation subsystem. Minimum value 0 for no defense. Highest allowed defense level is + 7

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
TPASW	Time at which a time period ends for DBL data tables; there may be up to 10 time periods for each table
TRETARG	Time required to retarget for known in-flight missile aborts (hours)
TTOS	Total time on station (for a tanker) (hours)
TVUL	Time a missile remains within vulnerable range of launch site (hours)
TYPE	Arbitrary alphameric designator (type name) to identify smallest sets in data base
TYPET	Target TYPE
TYPE1 } TYPE2 }	Attributes assigned fighter interceptor units (ICLASS=5 in the data base): attribute TYPE is assigned the TYPE1 or TYPE2 value based on BASEMOD input parameter POSTURE (POSTURE=1 TYPE1 is used; otherwise TYPE2 value used)
VAL	Relative value of an item within its CLASS as established in the data base by the user
VALU	Game value of an item (assigned in plan generation based on user-input parameters)
VAL1 } VAL2 }	Attributes assigned fighter interceptor units (ICLASS=5 in the data base): attribute VAL is assigned the VAL1 or VAL2 value based on BASEMOD input parameter POSTURE (POSTURE=1, VAL1 is used; otherwise VAL2 value is assigned)
VULN	Vulnerability number
WACNO	World aeronautical chart number
WHDTYPE	Warhead type index assigned in the data base
WHDTYPEN	Warhead type index (used with EVENTN)
YIELD	Yield (MT)
ZONE	An area bomber defense zone enclosed by a set of linked boundary points

APPENDIX C
ENTRY POINTS FOR QUICK UTILITY ROUTINES

This appendix contains an alphabetic listing of the entry points associated with all utility programs and subroutines. Subroutines associated with each of these entry points are indicated below.

<u>ENTRY POINT</u>	<u>TO SUBROUTINE</u>
ABORT	ABORT
ALOC DIR	FILEHNR
ANOTHER	ANOTHER
ATN2PI	ATN2PI
CHANGE	CHANGE
CLOSPIL	CLOSPIL
CLRCMON	CLRCMON
DEACTIV	FILEHNR
DECLARES	DECLARES
DELLONG	DELLONG
DIFFLNG	DIFFLONG
DIFFLONG	DIFFLONG
DISTF	DISTF
DSTF	DSTF
ENDDATA	ENDDATA
ENDTAPE	ENDTAPE
ERAZE	ERAZE
EQUIV	EQUIV
FILEDUMP	FILEDUMP
FILEHNR	FILEHNR
GETCLK	GETCLOCK

ENTRY POINT

GETCLOCK
GETDATE
GETDF
GETLIMIT
GETLOC
GETVALU
IGET
INBUFDK
INERRDK
INERRTP
INITAP
INITAPH
INITEDIT
INITEDT
INLABEL
INPITEM
INTERP
INTERPGC
INTRPGC
INPUT
ITLE
IWANT
KEYMAKE
LOCIF
LOCREAD
LOCWRIT
LOCWRITE
NEWUNIT
NEXTAPE
NEXTFILE

TO SUBROUTINE

GETCLOCK
GETDATE
GETDF
GETLIMIT
GETLOC
GETVALU
IGET
INBUFDK
INERRDK
INERRTP
FILEHNR
FILEHNR
INITEDIT
INITEDIT
INLABEL
INPITEM
INTERP
INTERPGC
INTERPGC
INPUT
ITLE
IWANT
KEYMAKE
LOCIF
LOCREAD
LOCREAD
LOCREAD
NEWUNIT
NEXTAPE
NEXTFILE

<u>ENTRY POINT</u>	<u>TO SUBROUTINE</u>
NEXTITEM	INITITEM
NEXTITM	INITITEM
NODIRC	NODIRC
NUMGET	NUMGET
OPENSPL	OPENSPL
ORDER	ORDER
OUTBFDK	OUTBFDK
OUTBFTP	OUTBFTP
OUTDF	OUTDF
OUTERDK	OUTERDK
OUTERTP	OUTERTP
OUTFILE	OUTFILE
OUTITEM	OUTITEM
OUTWORDS	OUTWORDS
OUTWRDS	OUTWORDS
PAGESKIP	PAGESKP
PAGESKP	PAGESKP
PRITEM	PRITEM
PRNTBAS	PRNTBASE
PRNTBASE	PRNTBASE
PRNTBSE	PRNTBASE
PRNTDATA	PRNTDTA
PRNTDTA	PRNTDTA
PRNTDIRC	PRNTDIRC
PRNTDRC	PRNTDIRC
PRNTLAB	FILEINR
PRNTPAGE	PRNTPGE
PRNTPGE	PRNTPGE
RDARRAY	RDARRAY
READDIR	READDIR
RELOADF	RELOADF

ENTRY POINT

REORDER
SETHLAD
SETREAD
SETWRIT
SETWRITE
SKIP
SSKPC
STORAGE
TERMTAP
TERMTAPE
TERMPIE
TIMEDAY
TIMEME
WARNING
WRARRAY
WRITEDIR
WRITEDR
WRWORD

TO SUBROUTINE

REORDER
SETHLAD
SETREAD
FILEINR
FILEINR
SKIP
SSKPC
STORAGE
TERMTAP
TERMTAP
TERMTAP
TIMEDAY
TIMEME
ABORT
RDARRAY
WRITEDIR
WRITEDIR
FILEINR

APPENDIX D UTILIZATION OF GENERAL UTILITY ROUTINES

The following is a list of the programs and subroutines which call the general utility routines described in chapter 4.

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
ABORT	ALOC BOUNDARY ENDTAPE FILEINR FINDIT FOOTEST GETDATA GETGROUP GETLOC GOPRINT GRPSORT IGET INDMOD INERRDK INERRTP INLABEL MISASGN NEWDATA NODIRC PREPALOC PRNTNOW PROCSIMP QUIBASE READDIR READIN SETDATA SETREAD SIMULATE SORTOPT ZABORT
ANOTHER	CLOSPIL ENDTAPE OUTERTP SETHEAD

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
ATN2PI	INTERPGC
CHANGE	DBMOD INDEXER
DELLONG	CORRPARM GETDATA
DIFFLONG	FINDZONE LAUNCH MYZONE TARDEFS
DISTF	ADJUST CORRPARM DIFF DISTIME FLYPOINT GENRAID GETDATA INITOPT INPUTGT INTERPGC LNCHDATA MISASGN NEWCOOR NOCORR PLANTANK ROUTING SNAPOUT TARDESS TGTPREP TIMELNCH WEAPPREP
ENDDATA	ENDTAPE INPITEM MAKEBAS MAKEIT READSUM
ERAZE	OUTERTP SETWRITE TERMTP WRARRAY

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
GETCLOCK	TIMEME
GETDATE	FILEHNR INITAPE INPIITEM QUIKBASH RELOADF OUTFILE
GETLIMIT	STORAGE
GETVALU	PREPALOC RDALGRD RDCARDF RDPRCMP SETFILE
IGET	BDAMAGE BDAMX BLAUN CLAUN COLOCATE FILTROUTE IPCL LATTRIT MLAUN NAVCAI NEXTFLT PRERAID RECOVERY STATSUM STRKOUT TDEFSTT TEFMBMD TKYLAUN WRPRNT WRRDSTRK
INTEDIT	BASESUM INDMOD PRNTBASE TABGEN TABLE

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
INPTM	BASESUM DBMOD INDEXER INDMOD PRNTBASE PRNTDTA TABGEN TABLE
INTERP	ADJUST ENDTAPE NOCORR
INTERPGC	INTERP
IPUT	ALOCOUT BDAMAGE CLAUN COLOCATE INDEXER MLAUN MOREDATA NAVATR NAVCAL PLANTANK TDEFSTT TRYLAUN
ITLE	BASESUM COLFIND CYCLER DBMOD EQUIV EVAL2 GETDATA INTREL LOCREST MAKEIT NEWBASE NEWDIR RDALCRD RDCARDF RDFRCMP ROWFIND

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
ITL (cont.)	SETUP TABLE TGTMODIF TGTPREP
IWANT	GETVALU
KEYMAKE	ALOCOUT BDAMX INDEXER MOREDATA PRERAID SIMULATE
LOCF	BLAUN LOCREAD RDARRAY TERMBMD WRARRAY
NEWUNIT	OUTERTP
NEXTFILE	FILEDUMP
NEXTITEM	TABGEN
NUMGET	BUFFIT CARDCK DBMOD DECLARES DEFALOC FILEDUMP FIXWEAP FLAGRST GETDATA GETVALU INDMOD INPRTCL ENCHDATA LOCREST MIRVREST MULCON NEWBASE NEWDATA NEWDIR RDALCRD

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
NUMGET (cont.)	RDCARDF RDPRCMP RNGEMOD SETFILE SETID SETUP TABLINPT VALUMOD
ORDER	ALOCOUT BOOSTIN CALCOMP COLOCATE DECOYADD EVAL EVAL2 FINDZONE FOOTPRNT GENRAID INITOPT NEWCOOR OPTBOOST PACK PLANTANK PROCCOMP ROUTINE SORTMIS STRKOUT TGTSORT
OUTITEM	DBMOD ENDTAPE INDEXER INDMOD
OUTWORDS	INPITEM
PAGESKP	BASISUM DBMOD ENDGAME EVAL2 FASTDATA INDMOD NEWDATA PAGESKP

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
PAGESKP (cont.)	PRNTDIRC PRNTPAGE PRTCOUNT SETUP SIMULATE STATSUM TABGEN
PRITEM	DBMOD INDEXER INDMOD PRINTIT PRNTBASE
PRNTBASE	MAKEBAS PRONLY
PRNTDTA	MAKEBAS PRONLY
PRNTDIRC	PRNTBASE PRNTDTA
PRNTPGE	PRNTDTA
RANORDER	MLAUN
READDIR	DECLARES INDEXER INITEDIT
REORDER	ALOCOUT CALCOMP COLOCATE DECOYADD EVAL EVAL2 FINDZONE FOOTPRNT GENRAID NEWCOOR PACK PROCCOMP SORTMIS STRKOUT

<u>UTILITY ROUTINE</u>	<u>CALLED BY</u>
SKIP	ALOC FOOTPRNT GETDATA GETGROUP SETDATA
SKIPFILE	ENDGAME INDEXER INDMOD MOREDATA NEXTAPE OUTERTP ZABORT
SSKPC	EVALPLAN MOVE VMARG
STORAGE	ALOC FOOTPRNT INDMOD PREPALOC
TIMEDAY	FILEINR INITAPE RELOADF
TIMEME	ALOC ALOCOUT ENDTAPE FLTRROUTE GETDATA MULCON OPTRAID PLANTMIS PRINTIT PRNTALL PRNTNOW PROCCOMP READSUM RECON SNAPIT SNAPOUT STRKOUT TGTASSN TIMEPRT WRDSTRK

UTILITY ROUTINE

CALLED BY

WRITEDIR

INITEDIT
MAKEBAS



DEFENSE COMMUNICATIONS AGENCY
NATIONAL MILITARY COMMAND SYSTEM
SUPPORT CENTER
WASHINGTON, D. C. 20301

IN REPLY
REFER TO: B221

8 November 1972

TO: DISTRIBUTION

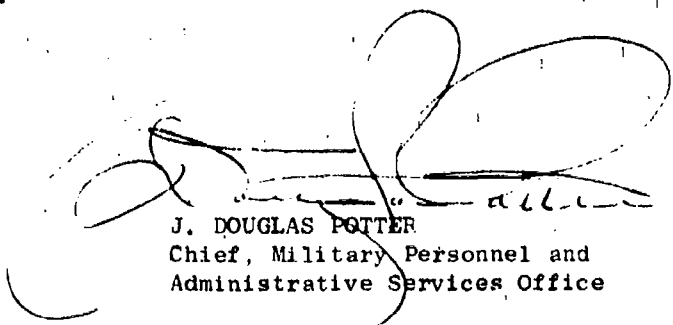
SUBJECT: Change 2 to Programming Specifications Manual CSM PSM
9A-67, Volume I, Data Input Subsystem, Part A

1. The purpose of this set of change pages is to document changes necessary to correct a deficiency in identifying the variable CCREL as a Quick-Reacting General War Gaming System (QUICK) attribute. These change pages reflect the currently operational version of QUICK at the National Military Command System Support Center. Insert the enclosed change pages and destroy the replaced pages according to applicable security regulations.

2. A list of Effective Pages to verify the accuracy of the Manual is enclosed. This list should be inserted before the title page. When this change has been posted, make an entry in the Record of Changes on the inside cover.

FOR THE COMMANDER:

6 Enclosures
Change 2 pages


J. DOUGLAS POTTER
Chief, Military Personnel and
Administrative Services Office

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. Department of Commerce
Springfield, VA 22151

AD-742783

4

DISTRIBUTION

<u>Addressee</u>	<u>Copies</u>
NMCSSC Codes	
B121	3
B122 (stock)	6
B200	1
B210	2
B220	29
B600	1
DCA Codes	
920	1
950	1
System Engineering Facility, ATTN: T221	
Reston, Virginia 22070	1
OJCS	
Studies, Analysis and Gaming Agency, ATTN: SFD, Room 1D957, Pentagon, Washington, D.C. 20301	5
Commander-in-Chief, North American Air Defense Command	
ATTN: NPPG, Ent Air Force Base, Colorado 80912	2
Commander, U.S. Air Force Weapon Laboratory (AFSC)	
ATTN: AWL, Kirtland Air Force Base, New Mexico 87117 . . .	2
Director, Strategic Target Planning	
Offutt Air Force Base, Nebraska 68113	2
Chief of Naval Operations, ATTN: OP963G	
Room 5E531, Pentagon, Washington, D.C. 20350	2
Defense Documentation Center, Cameron Station,	
Alexandria, Virginia 22314	<u>12</u> 70

6

EFFLCTIVE PAGES - 20 September 1972

This list is used to verify the accuracy of PSM 9A-67, Volume I after change 2 pages have been inserted. Original pages are indicated by the letter O, change 1 by the numeral 1, and change 2 by the numeral 2.

<u>Page No.</u>	<u>Change No.</u>
Title Page, Part A	0
ii-xiii, Part A	0
1-101	0
102-103	1
104-252	0
253	1
254-268	0
269-270	1
271-272	0
273	1
274-278	0
279	1
280	0
281-283	1
284	0
285-286	1
287	0
288-289	1
290	0
291	1
292	0
292.1	1
293-301	0
302	1
303	0
304	1
305-335	0
336-337	1
338-447	0
448	2
449-470	0
471	2
472-472A	0
Title Page, Part B	0
ii-vii, Part B	0
473-840	0
Title Page, Part C	0
ii-vi, Part C	0
841-1219	0

APPENDIX B
QUICK ATTRIBUTE NAMES AND DESCRIPTIONS

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
ABRATE	Probability of aircraft in-flight abort per hour of flying time
ADBLI	ALERTDBL probability for initiative attack
ADBLR	ALERTDBL probability for a retaliatory attack
ADEFCMP	Area ballistic missile defense (BMD) component index (radar or missile launch site)
ADEFZON	Area ballistic missile defense (BMD) zone number
AGX	Offset X-coordinate of AGZ (fiftieths of nautical miles)
AGY	Offset Y-coordinate of AGZ (fiftieths of nautical miles)
AHOB	Actual height of burst of weapon (air or ground)
ALERTDBL	Probability of destruction before launch (DBL) of alert delivery vehicle (missile or bomber)
ALERTDLY	Delay of alert vehicle before commencing launch (hours)
AREA	Area of a bomber defense ZONE (millions of nautical miles ²)
ASMTYPE	Air-to-surface missile type
ATTRCORR	Attrition parameter for a bomber corridor (probability of attrition per nautical mile)
ATTRLEG	Attrition parameter for each route leg in bomber sortie (probability of attrition per nautical mile)
ATTRSUPP	Amount of original attrition that remains after defense suppression

<u>ATTRIBUTE NAME</u>	<u>DESCRIPTION</u>
AZON1	First area defense zone covered by a BMD long-range radar
AZON2	Second area defense zone covered by a BMD long-range radar
AZON3	Third area defense zone covered by a BMD long-range radar
BCODE	Code indicating the outcome of a simulated bomber event
BENO	Bombing encyclopedia number
BLEGNO	Index to boundary line segment
CATCODE	Category Code as reflected in Joint Resource Assessment Data Base (JAD)
CEP	Circular error probable (CEP), delivery error applicable to bomber and missile weapons (nautical miles)
CLASS	Class name assigned identify sets of TYPES in data base
CLASST	Target CLASS
CNTRYLOC	Country code for country where item is located
CNTRYOWN	Country code for country which owns the item
CNTYLOCT	Target country code for country where the target is located
CNTYOWNT	Target country code for country which owns the target
CODE	Outcome code for a general event used in simulation
CPACTY	Capacity of a bomber recovery base (number of vehicles)

DISTRIBUTION

<u>Addressee</u>	<u>Copies</u>
NMCSSC Codes	
B121	3
B122 (stock)	6
B200	1
B210	2
B220	29
B600	1
DCA Codes	
920	1
950	1
System Engineering Facility, ATTN: T221 Reston, Virginia 22070	1
OJCS	
Studies, Analysis and Gaming Agency, ATTN: SFD, Room 1D957, Pentagon, Washington, D.C. 20301	5
Commander-in-Chief, North American Air Defense Command ATTN: NPPG, Ent Air Force Base, Colorado 80912	2
Commander, U.S. Air Force Weapon Laboratory (AFSC) ATTN: AWL, Kirtland Air Force Base, New Mexico 87117	2
Director, Strategic Target Planning Offutt Air Force Base, Nebraska 68113	2
Chief of Naval Operations, ATTN: OP963G Room 5E531, Pentagon, Washington, D.C. 20350	2
Defense Documentation Center, Cameron Station, Alexandria, Virginia 22314	<u>12</u> 70

Security Classification

DOCUMENT CONTROL DATA - R & D

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

1. ORIGINATING ACTIVITY (Corporate author) National Military Command System Support Center (NMCSSC) Defense Communications Agency (DCA) The Pentagon Washington, DC 20301		2a. REPORT SECURITY CLASSIFICATION	
3. REPORT TITLE The NMCSSC Quick-Reacting General War Gaming System (QUICK) Programming Specifications Manual, Volume I, Data Input Subsystem		2b. GROUP	
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) N/A			
5. AUTHOR(S) (First name, middle initial, last name) NMCSSC: Yvonne Mapily Donald F. Webb		Lambda Corp: Betty J. Ellis Jack A. Sarseen	
6. REPORT DATE 29 February 1972		7a. TOTAL NO. OF PAGES 486	7b. NO. OF REFS 4
8a. CONTRACT OR GRANT NO. DCA 100-70-C-0065		8b. ORIGINATOR'S REPORT NUMBER(S) NMCSSC COMPUTER SYSTEM MANUAL CSM PSM 9A-67	
9. PROJECT NO. NMCSSC Project 631		9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None	
10. DISTRIBUTION STATEMENT This document is approved for public release; its distribution is unlimited.			
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY National Military Command System Support Center/Defense Communications Agency The Pentagon, Washington, DC 20301	
13. ABSTRACT This is one of three volumes describing the computer programming specifications for the Quick-Reacting General War Gaming System (QUICK). This volume addresses computer programs of the QUICK Data Input Subsystem. It is intended to serve as the basis for program maintenance activities. Accordingly, it describes the program functions and contains flow charts for each program and subprogram of the Data Input Subsystem. Based upon suitable data base and user control parameters, QUICK will generate individual bomber and missile plans suitable for war gaming, and simulate the planned events. The generated plans are of a form suitable for independent review and revision. Subsequently, the planned events are simulated; various statistical summaries are produced to reflect the results of the war game. A variety of force postures and strategies can be accommodated. QUICK is documented extensively in a set of Computer System Manuals (series 9-67) published by the National Military Command System Support Center (NMCSSC), Defense Communications Agency (DCA), The Pentagon, Washington, DC 20301.			

RECEIVED
NATIONAL TECHNICAL
INFORMATION SERVICE

DD FORM 1473

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



DEFENSE COMMUNICATIONS AGENCY
NATIONAL MILITARY COMMAND SYSTEM
SUPPORT CENTER
WASHINGTON, D. C. 20301

IN REPLY
REFER TO 8221

1 September 1972

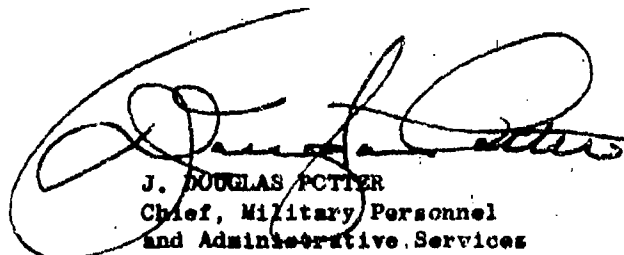
TO: DISTRIBUTION

SUBJECT: Change 1 to Computer System Manual CSM-PSM 9A-67, Volume I,
Part A, The NMCSSC Quick-Reacting General War Gaming System
(QUICK), Data Input Subsystem, Programming Specifications
Manual, 29 February 1972.

1. Insert the enclosed change pages and destroy the replaced pages according to applicable security regulations.
2. A list of Effective Pages to verify the accuracy of this manual is enclosed. This list should be inserted before the title page.
3. When this change has been posted, make an entry in the Record of Changes on the inside cover.

FOR THE COMMANDER:

21 Enclosures
Change 1 pages


J. DOUGLAS POTTER
Chief, Military Personnel
and Administrative Services
Office

DISTRIBUTION

NMCSSC Codes

B121	3
B122 (stock)	6
B200	1
B210	2
B220	19
B600	1

DCA Codes

920	1
950	1

Reproduced by
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. Department of Commerce
Springfield VA 22151

D D C
RECEIVED
DEC 18 1972
B

DEFENSE SECRETARIAT A

Approved for public release;
Distribution Unlimited

EFFECTIVE PAGES - 17 July 72

This list is used to verify the accuracy of CSM PSM 9A-67, Volume I, Part A, after change 1 pages have been inserted. Original pages are indicated by the letter O, change 1 by the numeral 1.

<u>Page No.</u>	<u>Change No.</u>
Title Page	O
ii-xiii	O
1-101	O
102-103	1
104-252	O
253	1
254-268	O
269-270	1
271-272	O
273	1
274-278	O
279	1
280	O
281-283	1
284	O
285-286	1
287	O
288-289	1
290	O
291	1
292	O
292.1	1
293-301	O
302	1
303	O
304	1
305-335	O
336-337	1
338-470	O
471	1

NMCSSC Letter, B221, Change 1 to Computer System Manual CSM PSM 9A-67,
 Volume I, Part A, The NMCSSC Quick-Reacting General War Gaming System
 (QUICK), Data Input Subsystem, Programming Specifications Manual,
 29 Feb 72.

OJCS

Studies, Analysis and Gaming Agency, ATTN: SFD, Room ID957, Pentagon, Washington, D.C. 20301	5
Commander in Chief, North American Air Defense Command, ATTN: NPPG, Ent Air Force Base, Colorado 80912	2
Commander, U.S. Air Force Weapon Laboratory (AFSC), ATTN: AWL, Kirtland Air Force Base, New Mexico 87117	2
Director, Strategic Target Planning, Offutt Air Force Base, Nebraska 68113	2
Chief of Naval Operations, ATTN: OP963G, Room 5E531, Pentagon, Washington, D.C. 20350	2
Defense Documentation Center, Cameron Station, Alexandria, Virginia 22304	<u>12</u>
	59

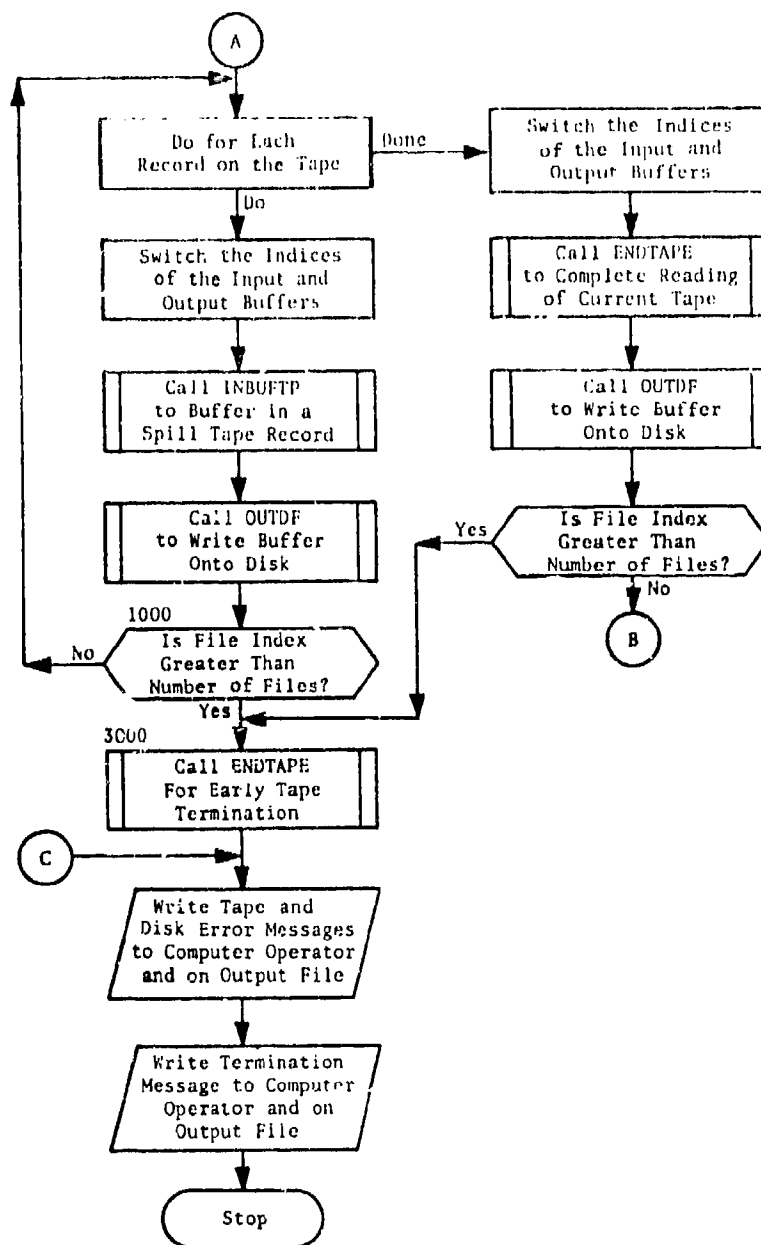


Fig. 19. (cont.)
(Sheet 2 of 2)

SUBROUTINE ENDTAPE

PURPOSE: To complete the reading of the current spill tape.

ENTRY POINTS: ENDTAPE

FORMAL PARAMETERS: None

COMMON BLOCKS: DICTARY, ERRNUM, LOCATOR, MACHINE, SUPRVIS

SUBROUTINES CALLED: INERRTP, ANOTHER, NEWUNIT

CALLED BY: RELOADF

Method

Subroutine ENDTAPE checks the status of the last tape read operation for the current tape. If the operation terminated with an end-of-file or parity error, it calls INERRTP to attempt to reread the record.

When the tape has been successfully read, ENDTAPE calls ANOTHER or NEWUNIT to release it and then checks to see if the tape unit will be needed for another spill tape. If so, ENDTAPE instructs the computer operator to mount the next spill tape on the tape unit.

The subroutine returns after printing out error counts which indicate the number of tape read errors encountered on the last tape and on all the tapes read so far.

Subroutine ENDTAPE is illustrated in Figure 26.

CH-1

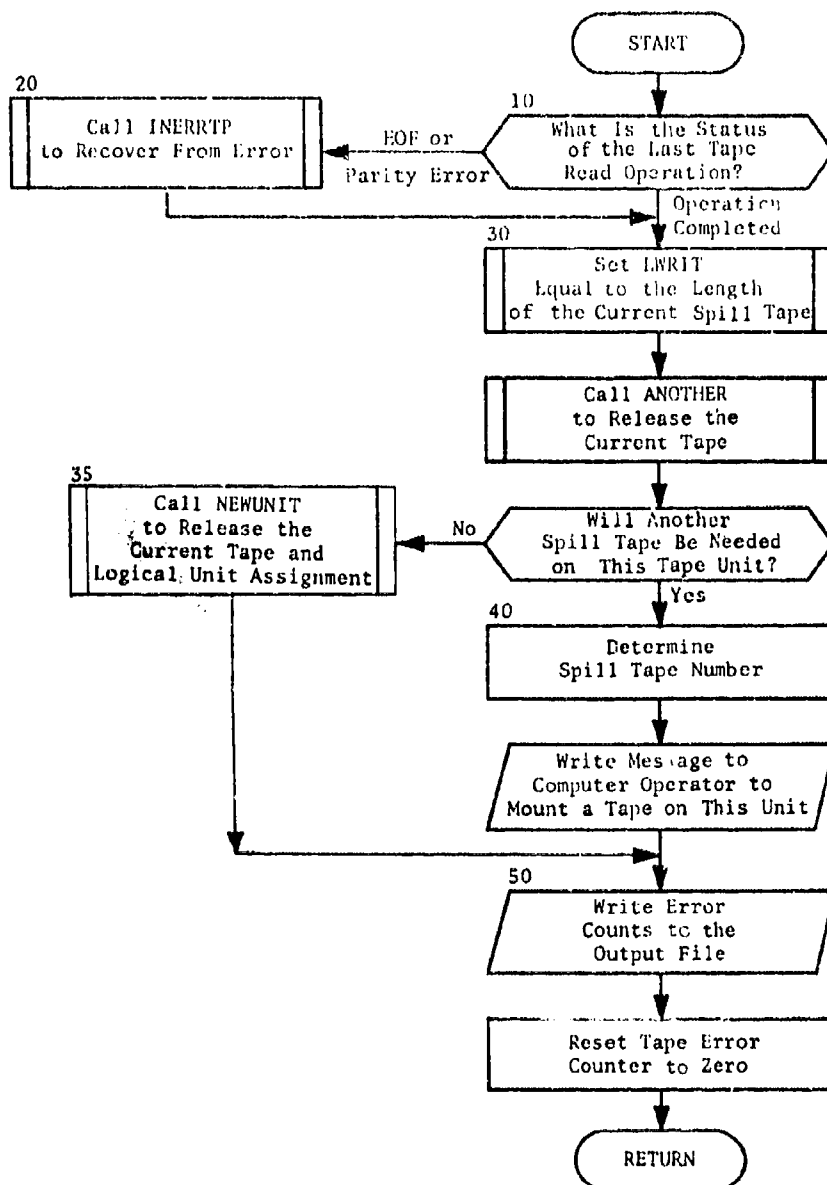


Fig. 20. Subroutine ENDTAPE

SUBROUTINE GETLOC

PURPOSE: To read the major directory to the spill tapes (common /LOCATOR/) from the first spill tape.

ENTRY POINTS: GETLOC

FORMAL PARAMETERS: None

COMMON BLOCKS: DRCS162, ERRMESS, LOCATOR, MACHINE, SUPRVIS, TAPHARD

SUBROUTINES CALLED: NODIRC, ABORT

CALLED BY: RELOADF

Method

Subroutine GETLOC first attempts to buffer in the major directory -- common /LOCATOR/ -- from the first spill tape. If an end-of-file or parity error is encountered during the buffer operation, the tape is rewound and the buffer attempt is repeated. If this buffer process is repeated MTIMESR, common /SUPRVIS/, times without success, GETLOC determines how many words have been read successfully from the tape and places them (the words) into the master directory (common /DRCS162/) so that it can then use NODIRC to print out error information and abort the run.

If, on the other hand, the buffer operation is successful, GETLOC checks the value of MAXFILE (which is the maximum number of disk files contained on the spill tapes) against the value of MAXFILE set earlier in RELOADF. If they are not equal, then either RELOADF has not been updated to correspond to program OUTFILE, or the value read from the tape is in error for some other reason. In either case, the discrepancy is encoded into an error message and ABORT is called to abort the run. The computer operator is also informed of the discrepancy.

In the normal case where the major directory is read successfully and MAXFILE does not have an unexpected value, GETLOC writes the major directory on the standard output file and processing control is returned to RELOADF.

Subroutine GETLOC is illustrated in figure 21.

Table 6 . (cont.)
(Sheet 5 of 5)

<u>BLOCK</u>	<u>VARIABLE OR ARRAY</u>	<u>DESCRIPTION</u>
OPTIONS (cont.)		
	ISETSIZ	Largest allowable line number in a set made by default option of SETID
SETIDD	ID	Array where set number to be printed is stored
	INDEX	9999 if all sets are to be printed; 0 if none are to be printed; count of number of sets to be printed if list read in
	JINDEX	0 if no sets to be printed; equal to INDEX if list read in; equal to number of sets updated if IAUTO=1
	IAUTO	1 if updated sets are to be printed; 0 if not
SIDECC	IXSD	Position of SIDE attribute in ATTNAME array
XYZ	XMIN	Value of attribute MINKILL for current item
	XMAX	Value of attribute MAXKILL for current item

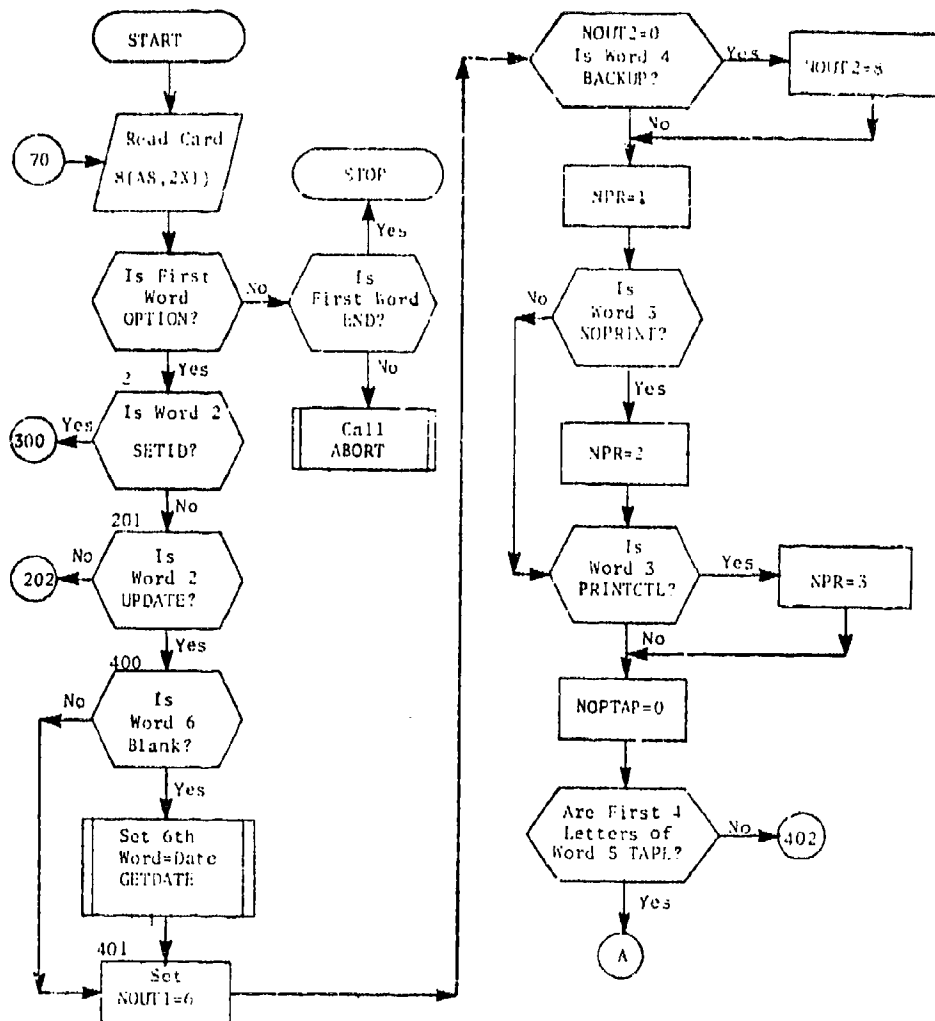


Fig. 77. Program QUIRBASE
(Sheet 1 of 4)

SUBROUTINE FASTSET

PURPOSE: FASTSET is the main control and monitoring routine for an update run.

ENTRY POINTS: FASTSET

FORMAL PARAMETERS: None

COMMON BLOCKS: ERRORM, ITP, MYIDENT, MYTAPES, NOERRORS, NOPRINT, OPTIONS, ICKTST, SETIDD, TWORD

SUBROUTINES CALLED: CARDCK, COPYDB, FILEHNR, INITFAST, INPRCTL, MAKEIT, NEWDATA, PAGESKP, PRTCONT

CALLED BY: QUIKBASE

Method

Subroutine FASTSET calls INITFAST to initialize all arrays. It assigns file names and initializes the filehandler and all pertinent tapes to read and write. If manual print control was requested on the option card, subroutine INPRCTL is called. Subroutine CARDCK is called to read in and check all the update data. If that subroutine has found errors in the input deck, FASTSET prints the error messages and aborts the run. If the update data have no discernible errors, subroutine MAKEIT is called to perform the major functions of the program. Control is returned to FASTSET which again looks for and prints, if present, any error messages. The subroutine to print the target-region summary (PRTCONT) is called and, if requested, a second copy of the updated data base is created by subroutine COPYDB. Subroutine FASTSET is illustrated in figure 83.

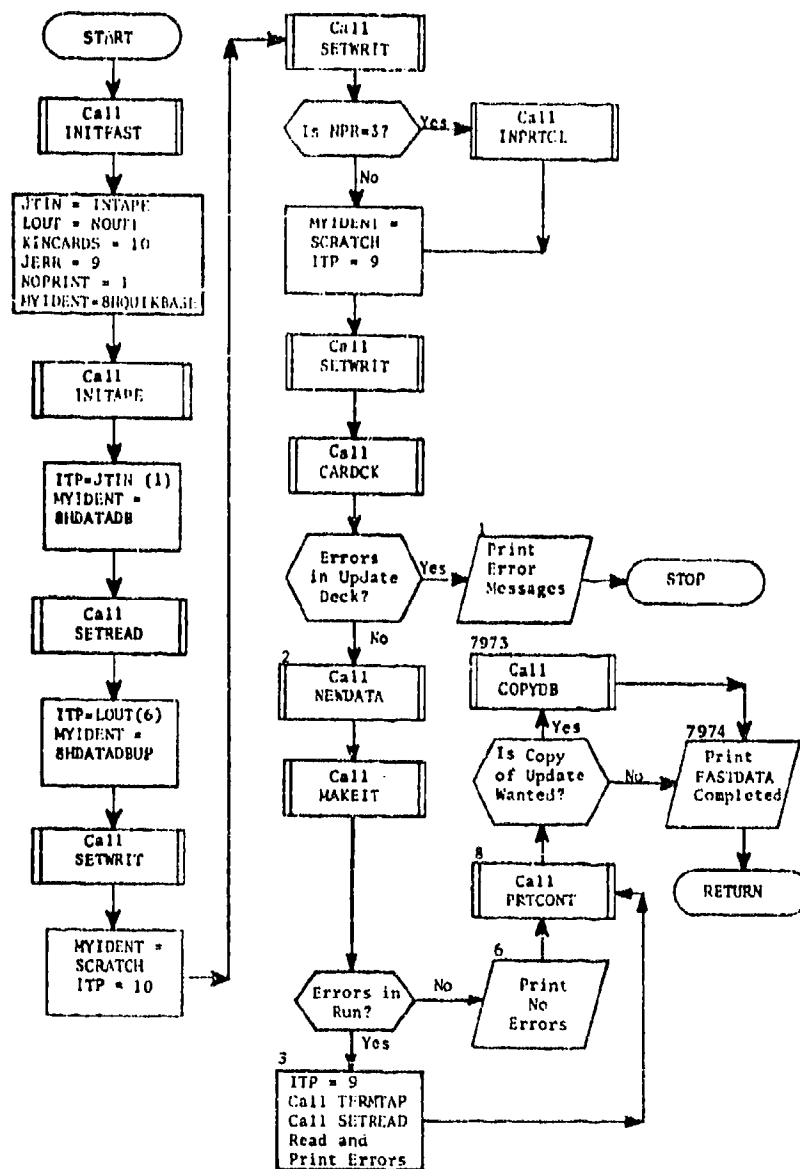


Fig. 83. Subroutine FASTSET

SUBROUTINE INITFAST

PURPOSE: To set constants and clear arrays for subroutines associated with QUIKBASE.

ENTRY POINTS: INITFAST

FORMAL PARAMETERS: None

COMMON BLOCKS: DIRECTRY, ERRORM, IENDSET, IWSIDE, KKSET, MPRTOPT, MYGOODS, MYPRINT, MYTAPES, NEWSET, NODESIGS, PRTOPT

SUBROUTINES CALLED: None

CALLED BY: FASTSET

Method

INITFAST does no computation. Either through data statements or executable statements, it presets constants and arrays to their appropriate values. Subroutine INITFAST is illustrated in figure 85.

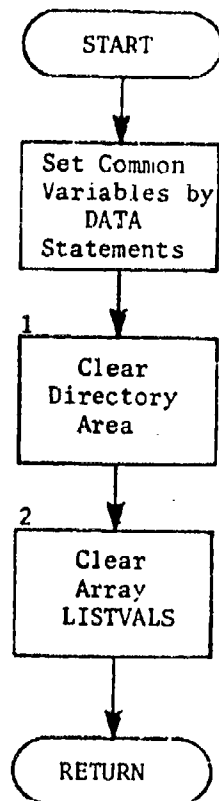


Fig. 85. Subroutine INITFAST

SUBROUTINE MAKEBAS

PURPOSE: To call the subroutines required to prepare a game data base tape QUIKDB from an input data library tape DATADB when updating of the DATADB file is not required.

ENTRY POINTS: MAKEBAS

FORMAL PARAMETERS: None

COMMON BLOCKS: ERRORM, HIST, ITP, LOGFLAG, MYIDENT, MYTAPES, OPTIONS, TWORD

SUBROUTINES CALLED: ENDDATA, FILEHNR, INITFAST, NEWBASE, NEWDATA, NEWDIR, PRNTBASE, PRNTDATA, WRITEDIR

CALLED BY: QUIKBASE

Method

Subroutine MAKEBAS is a driver routine which controls the sequencing of operations required to create the QUIKDB tape when the QUIKDBG option is exercised. As indicated in figure 60, MAKEBAS calls the filehandler (FILEHNR) to initialize the read, write, and scratch files. Then, MAKEBAS calls, in order, the other subroutines required to write the QUIKDB tape. In effect, MAKEBAS is essentially the same as a null UPDATE run.

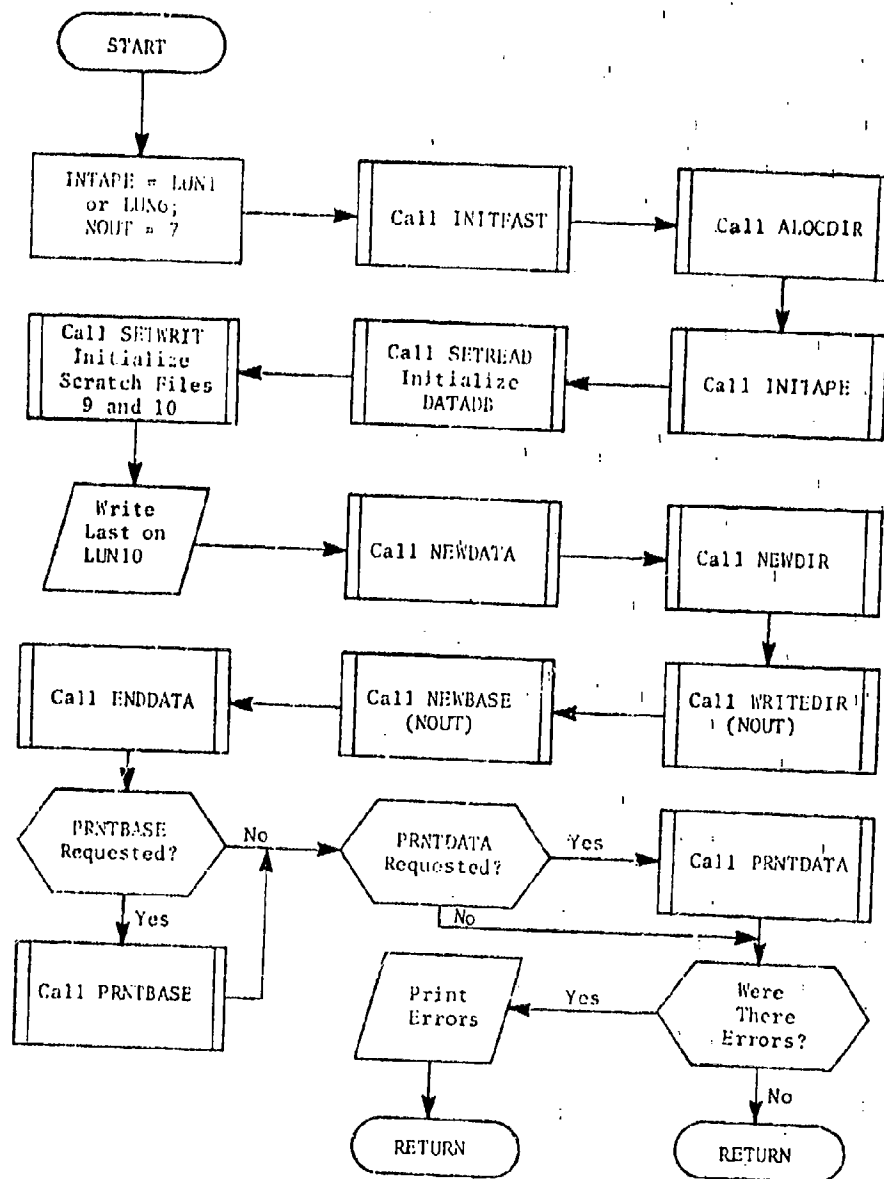


Fig. 88. Subroutine MAKEBAS

SUBROUTINE MAKEIT

PURPOSE: A driver to call the subroutines necessary to make a new data base.

ENTRY POINTS: MAKEIT

FORMAL PARAMETERS: NT1 - The tape where the QUIKDB tape will be written

COMMON BLOCKS: MYIDENT

SUBROUTINES CALLED: ENDDATA, NEWBASE, NEWDIR, WRITEDIR

CALLED BY: FASTSET

Method

MAKEIT calls the data base generation subroutines, NEWDIR, WRITEDIR, NEWBASE, and ENDDATA, which create the game base file, QUIKDB. Its only computation function is to look up and store for later use the index number of the attribute DESIG in the data base directory (array ATTNAME). Subroutine MAKEIT is illustrated in figure 89.

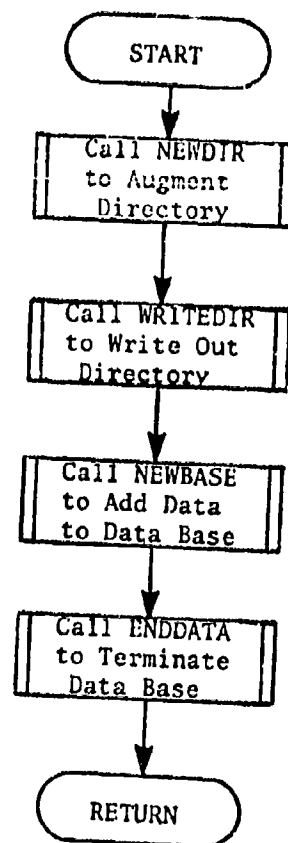


Fig. 89. Subroutine MAKEIT

SUBROUTINE MOVEIT

PURPOSE: To move update data to output buffer; add the desired data or update identification; and to add current set and line number to output record.

ENTRY POINTS: MOVEIT

FORMAL PARAMETERS: IHCWTO - Switch to indicate whether input buffer is to be moved

COMMON BLOCKS: ERRORM, MYOUT, MYINPUT, MYGOODS

SUBROUTINES CALLED: FILEHNR

CALLED BY: NEWDATA

Method

Subroutine MOVEIT increases the current line number within the set by one. It examines the input parameter IHOWTO to determine which of two functions it is to perform. If the argument is equal to a one, MOVEIT transfers a data record from common block /MYINPUT/ to block /MYOUT/. It next sets the tenth word of the block to the update identification. The set and line number are encoded as two four-digit numbers into the ninth word. If the input argument was a two, only the last function is performed by MOVEIT; i.e., the set and line number are encoded into the data record. Subroutine MOVEIT is illustrated in figure 90.

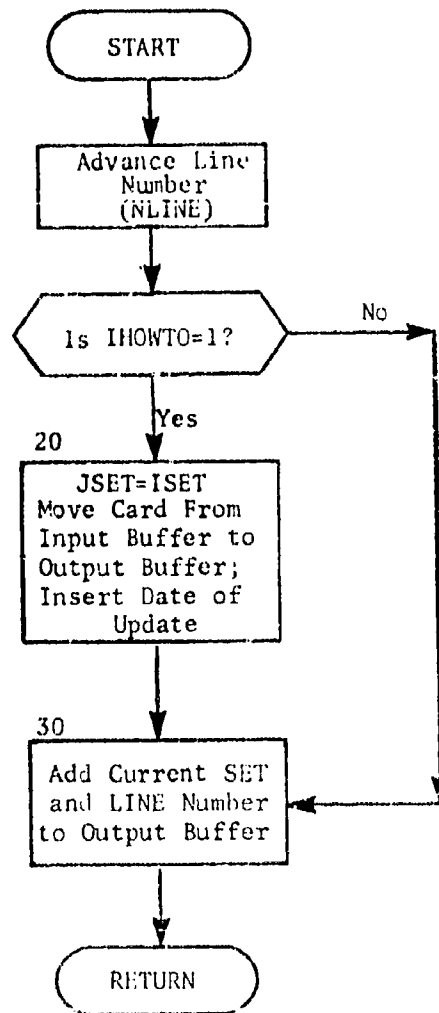


Fig. 90. Subroutine MOVEIT

SUBROUTINE NEWBASE

PURPOSE: To prepare or augment the item portion of a data base tape.

ENTRY POINTS: NEWBASE

FORMAL PARAMETERS: NT1 - the logical tape number where the base is to be written.

COMMON BLOCKS: DIRECTRY, ERRORM, ICONTROL, IENDSET, ITP, JDESTEST, KKSET, MYGOODS, MYOUT, NOTEST, PRTOPT, SIDECC, TWORD, XYZ

SUBROUTINES CALLED: COUNTDS, ITLE, NEWDATA, NUMGET, WRARRAY*, WRWORD*

CALLED BY: MAKEIT

Method

Subroutine NEWBASE employs subroutine NEWDATA to read the item portion of the data base, check for errors, and write each item on the specified output tape, NT1.

Four commands are recognized: DEFINE, UNDEFINE, ITEM, and ENDINPUT. In the case of DEFINE the succeeding fields on the card beginning in columns 11, 21, 31, etc. contain attribute-value pairs which are to be made into global definitions in which the first field of the pair is the BCD name of the attribute, left-justified, followed by the value in the second field. The sequence of attribute-value pairs occurring on a card is terminated by a blank field.

The ITEM card is as described above except that the definition is local and the entire sequence of cards is terminated only upon detection of another command in the first field of a card.

The UNDEFINE card removes global definitions with the names of the attributes to be undefined occurring in succeeding fields on the card, terminated by a blank.

The deck of input cards is terminated by ENDINPUT which also causes NEWBASE to return to the calling program.

* See subroutine FILEINR.

All cards read by the routine are checked for consistency unless checking has been turned off by an update PRNTCL option card (see subroutine INPRCTL). That is, the attribute specified is checked to determine that it is in fact defined in the data base directory and that the value associated satisfies any range or list check specifications for that attribute. Appropriate error messages are emitted when such inconsistencies are detected. The flowchart (figure 91) consists of four parts. Part I shows the processing sequence used in NEWBASE. Parts II, III, and IV show the operations of three local subroutines used by NEWBASE to perform the data checks and, if required, to write error messages. Part II shows the local subroutines used to signal undefined attributes (see statement 110) and to signal an error in the assigned attribute value (see statement 120). Part III shows the procedures used to convert and check the attribute-value pair. Part IV shows the local subroutine check on the MINKILL and MAXKILL values for the item.

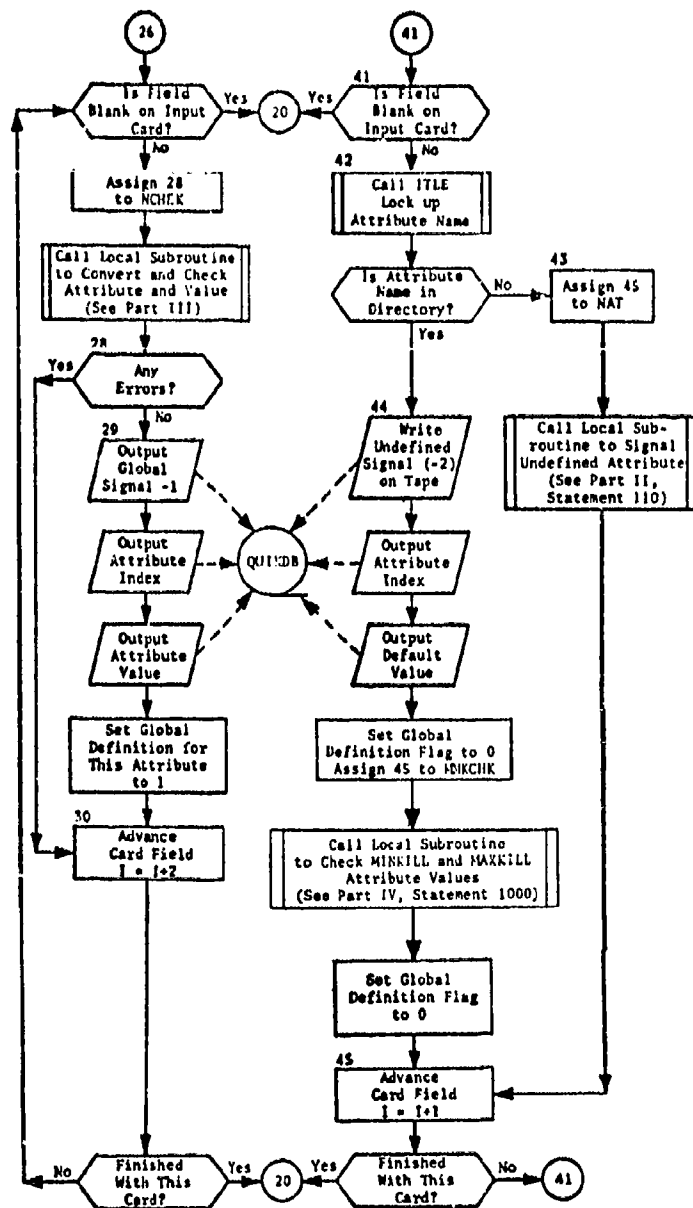


Fig. 91. (cont.)
Part I: (cont.)
(Sheet 2 of 3)

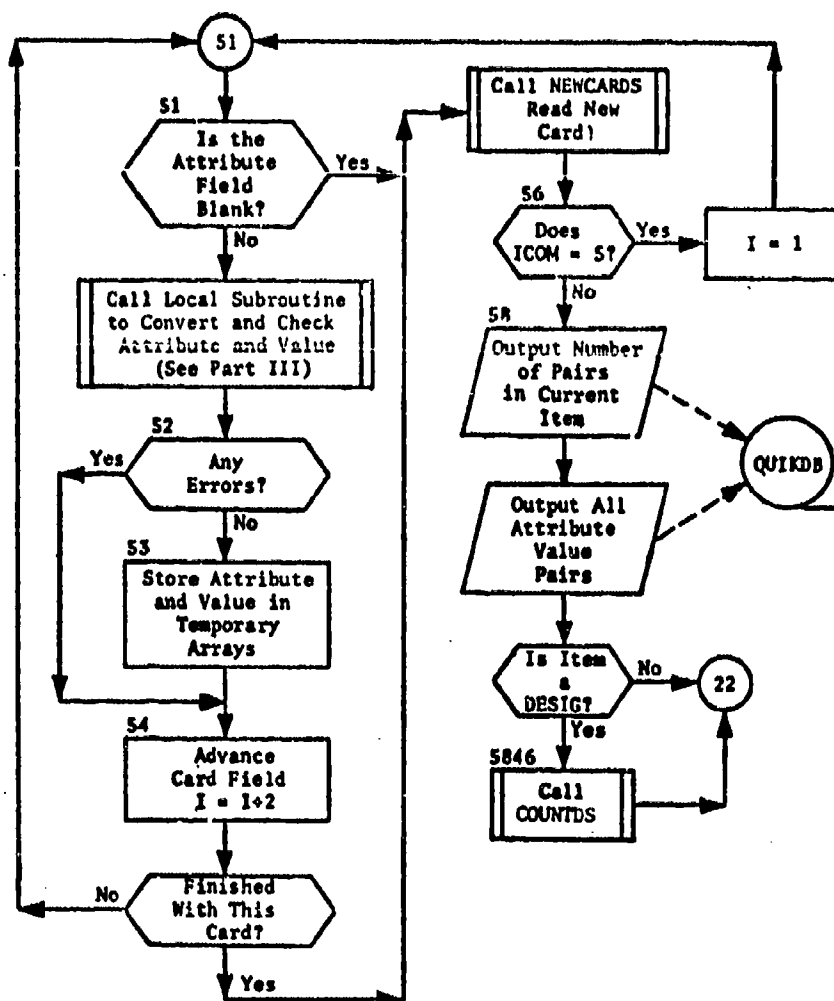


Fig. 91. (cont.)
Part I: (cont.)
(Sheet 3 of 3)

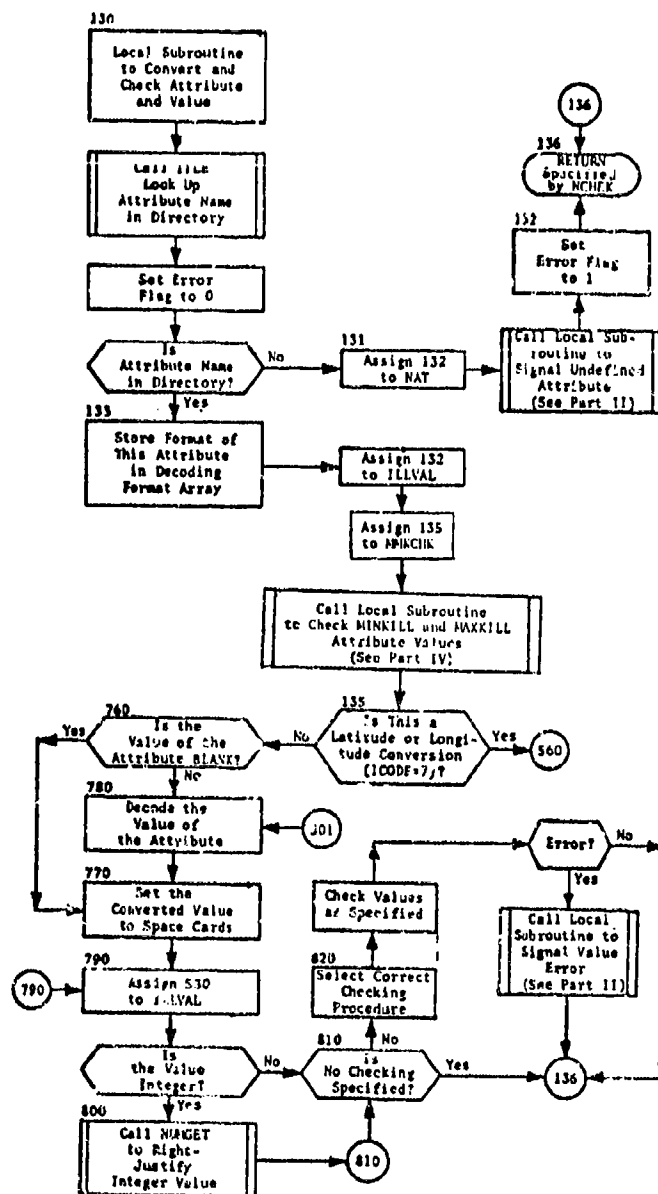


Fig. 91. (cont.)
Part III: Local Subroutine for
Attribute/Value Checking
(Sheet 1 of 2)

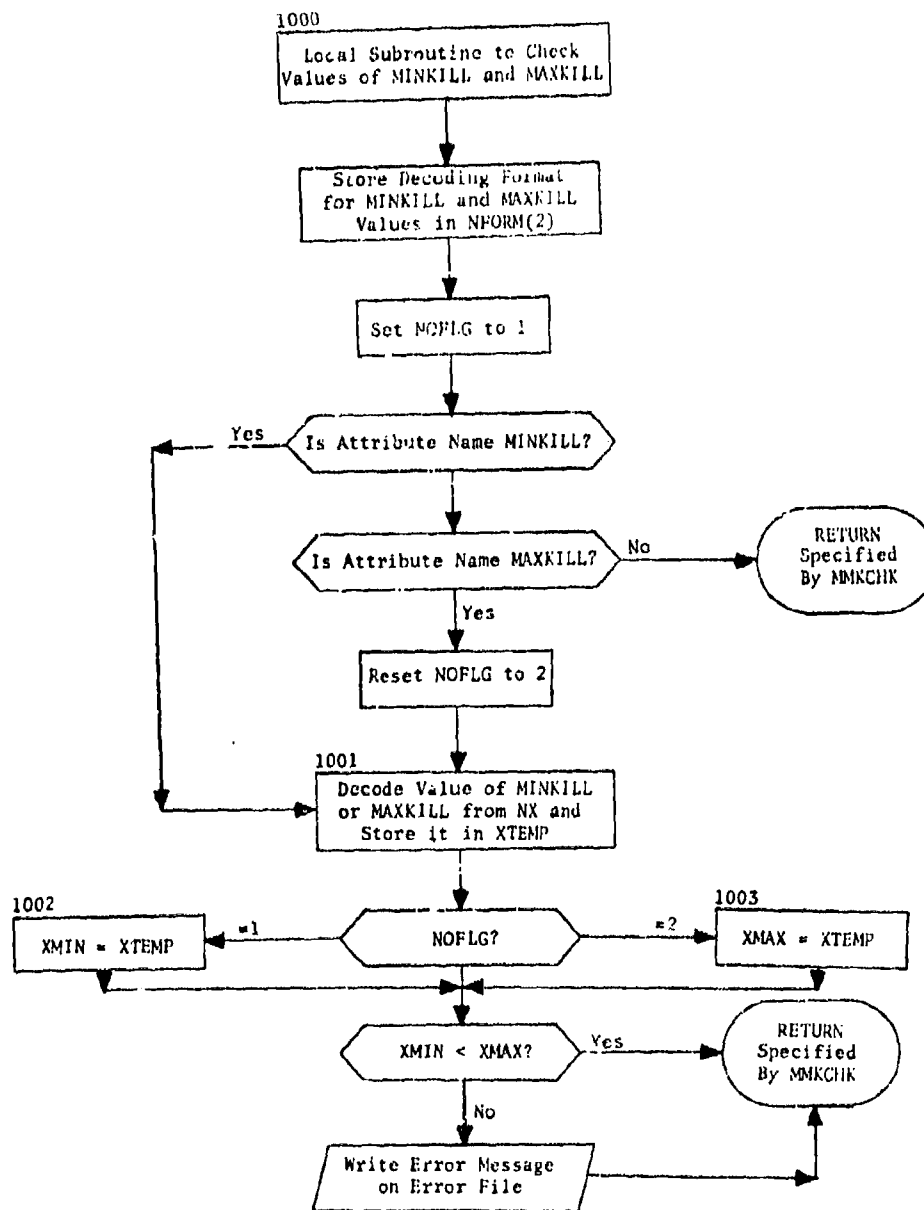


Fig. 91. (cont.)
Part IV: Local Subroutine for
MINKILL and MAXKILL
Value Checking

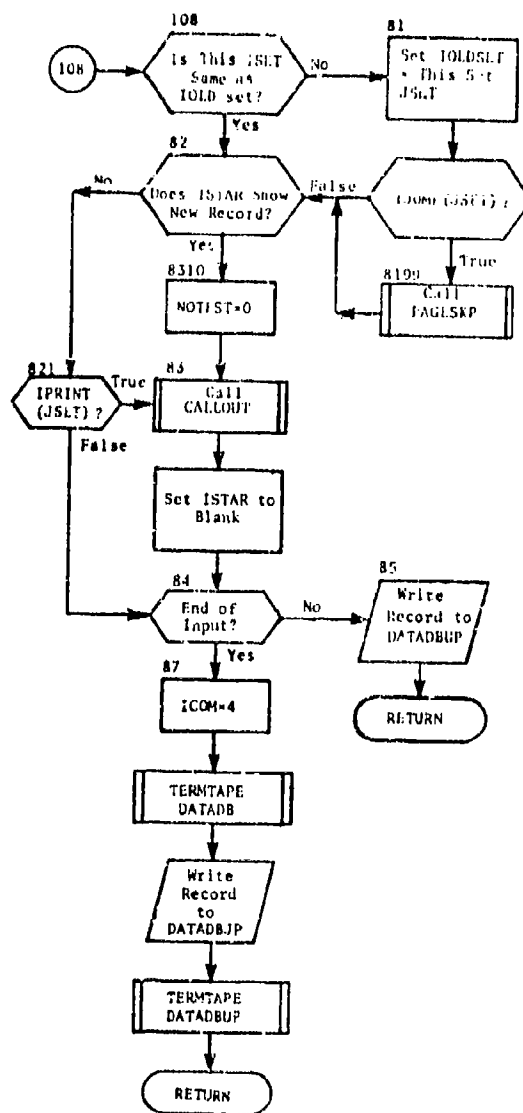


Fig. 92. (cont.)
(Sheet 7 of 7)

SUBROUTINE NEWDIR

PURPOSE: To create a new data base directory.

ENTRY POINTS: NEWDIR

FORMAL PARAMETERS: None

COMMON BLOCKS: DIRECTRY, ERRORM, ITP, JDESTEST, MPRTOPT, MYOUT, SIDECC, TWORD, XYZ

SUBROUTINES CALLED: ITLE, NUMGET, NEWDATA (Entry NEWCARDS)

CALLED BY: MAKEIT

Method

This subroutine employs subroutine NEWDATA (entry NEWCARDS) to read a new directory or modify an existing directory. The card format is eight fields of 10 columns each with all quantities left-justified. Two commands in the first field are recognized: ADD and ENDIRECT. The ENDIRECT card serves to terminate the subroutine and causes a return to the calling program.

The ADD command is used to add a new attribute to the directory, or, in conjunction with a prior delete command, to change an already existing attribute in the directory.

With the ADD command there are, in addition to the first field, six further fields of data on the input card:

1. The name of the attribute in BCD.
2. The input/output conversion format (FORTRAN) associated with the values for that attribute.
3. Code number specifying the type of checking to be conducted for a particular attribute (see below).
4. The default value of the attribute, in the appropriate input/output format for that attribute as specified by item 2. This is the value that will be associated with the attribute when it is in an undefined state.

5. Checking specifications. This field may contain the word LIST, which specifies list checking with the list of allowable attribute values to follow on subsequent cards; the word NOCHECK, which specifies no checking of the attribute values; or the lower value of the allowable range of values for this attribute in the case of range checking.
6. This field is unused in the event of list checking, or no checking, and contains the upper value for the range of allowable values of the attribute in the case of range checking.

If list checking is specified on the ADD card, this card is followed by any number of cards containing the list of allowable values for that attribute, eight per card, in the format specified for the particular attribute. The fields for these values are the first eight columns of each 10-column field. The series of allowable values in these cards is terminated by the first blank field. A blank field can be specified as an allowable attribute value by including the value BLANK in the list.

Appropriate error messages are written on an error message tape to point out inconsistent operations such as attempting to issue a command other than ENDIRECT or ADD, or attempting to add attributes which already exist in the directory. The error-checking codes permissible for the third extra field are:

<u>CODE</u>	<u>TYPE OF DATA TO BE INPUT</u>	<u>CHECKING SPECIFIED</u>
1	Floating point numeric	Range (Min-Max)
2	Floating point numeric	List
3	Fixed point numeric	Range (Min-Max)
4	Fixed point numeric	List
5	Alphameric	List
6	Alphameric	No checking
7	Special conversion for latitude, longitude	Range

Subroutine NEWDIR is illustrated in figure 93.

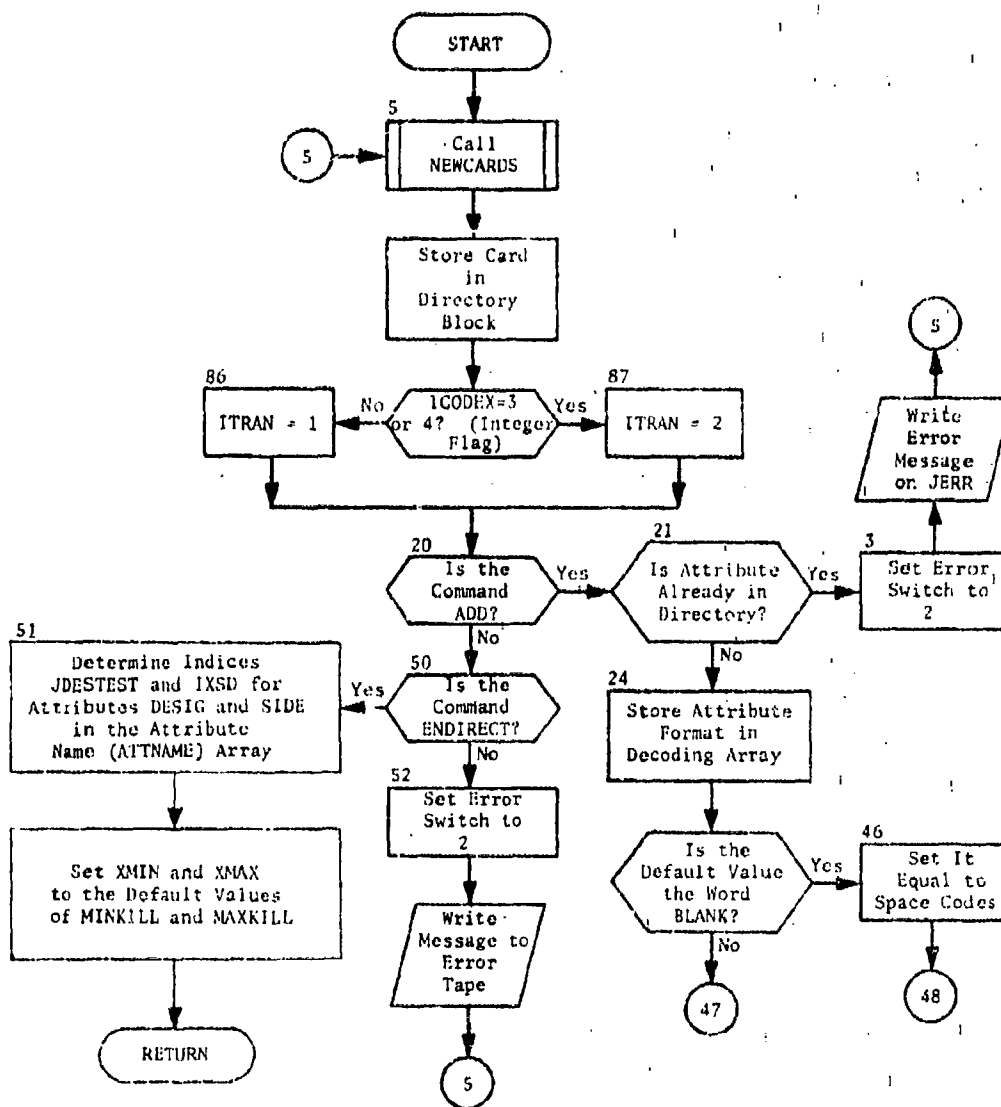


Fig. 93. Subroutine NEWDIR
(Sheet 1 of 3)

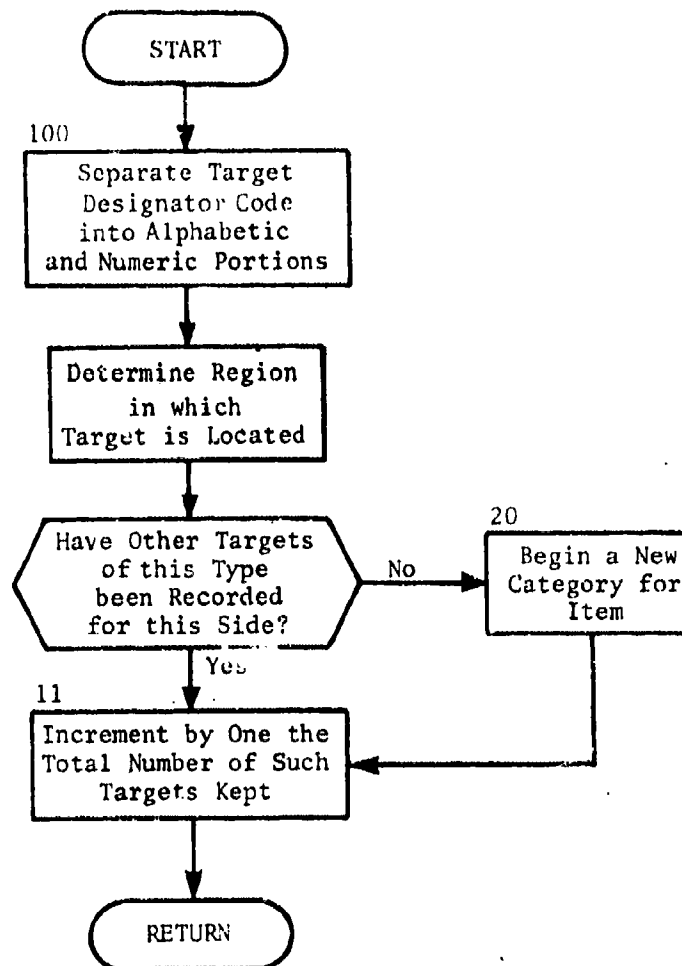


Fig. 100. Subroutine COUNTDES

SUBROUTINE DBMOD

PURPOSE: To control the information processing when program BASEMOD is run post-QUIKBASE.

ENTRY POINTS: DBMOD

FORMAL PARAMETERS: None

COMMON BLOCKS: CUTIGIW, EDITAPE, EDITERM, IDESIGS, ITP, LDESIGS, LODESIGS, MYIDENT, MYSIDE, NODESIGS, NOPRINT, NRTPES, PRINTS, PROCESS

SUBROUTINES CALLED: ADDVAL, COUNTDES, INITAPE*, INITEDIT, INPITEM, MYZONE, NEXTITEM; NUMDEL, NUMGET, OUTITEM, PAGESKP, PRINTIT, PRNTVAL, RDTPES, STKRIN, TARDEFS

CALLED BY: BASEMOD

Method

Subroutine DBMOD effects a sequential examination of each item in the game data base (contained on the QUIKDB tape). Each item is read in, filtered through a series of tests, and assigned appropriate values for certain of its attributes. The item is then either retained or deleted from the game data base. The 12 tasks accomplished by this processing are:

1. Targets which are inappropriate for the plan under consideration, i.e., those targets assigned the attribute RESERVE=0, are excluded from further consideration.
2. The appropriate number of bombers or tankers for each bomber or tanker squadron (NOPERSQN) is selected, depending upon the particular plan being developed (Initiative, Surprise, or Retaliatory).

* See subroutine FILEHNR.

3. The number of bombers or tankers in commission (NOINCOM) for each bomber or tanker squadron is calculated by specifying that NOINCOM is equal to a user-specified fraction of NOALERT which is a fraction of NOPERSQN.
4. The number of bombers or tankers which are on alert (NOALERT) for each squadron is calculated by specifying that NOALERT is equal to a user-specified fraction of NOINCOM.
5. The appropriate value of the attributes TYPE, VAL, and EFECTNES is established for each fighter interceptor unit based on the user-input parameter POSTURE. If POSTURE=1, these attributes are assigned the values of the attributes TYPE1, VAL1, and EFECNES1, respectively. If POSTURE=2, the values of the attributes TYPE 2, VAL2, and EFECNES2 are assigned.
6. The relative value (VAL) of urban/industrial targets is calculated as a function of either general industrial worth (IGIW) or population (POP).
7. If the TARDEF option is exercised, each target (opposing side) is processed and the level of local bomber defense available at the target is calculated.
8. If the ZONE option is exercised, items in ICLASS 4 and 5 (defensive command and control sites and interceptor bases, respectively) are processed to determine the air defense zone in which the item is located.
9. The value of the attribute IREG is determined based on the target designator code DESIG assigned to the item.
10. Each Blue (SIDE=BLUE) installation is assigned a value for the attribute FLAG. The assigned value (numeric code 1 through 8 established based on ICLASS) is subsequently used in program ALOC to impose user-restrictions on the allocation of weapons (see Program ALOC, User-Input Parameters, FLAGREST Function -- Restriction of Weapons Using FLAG Attribute in chapter 3 of User's Manual Volume II).
11. Targets may be deleted from the game base on the basis of TASK or MINIGIW (user-specified parameter which establishes the minimum index of general industrial worth to be considered).
12. The appropriate value of the attribute DBL (probability of destruction before launch) is chosen.

Subroutine DBMOD is illustrated in figure 101.

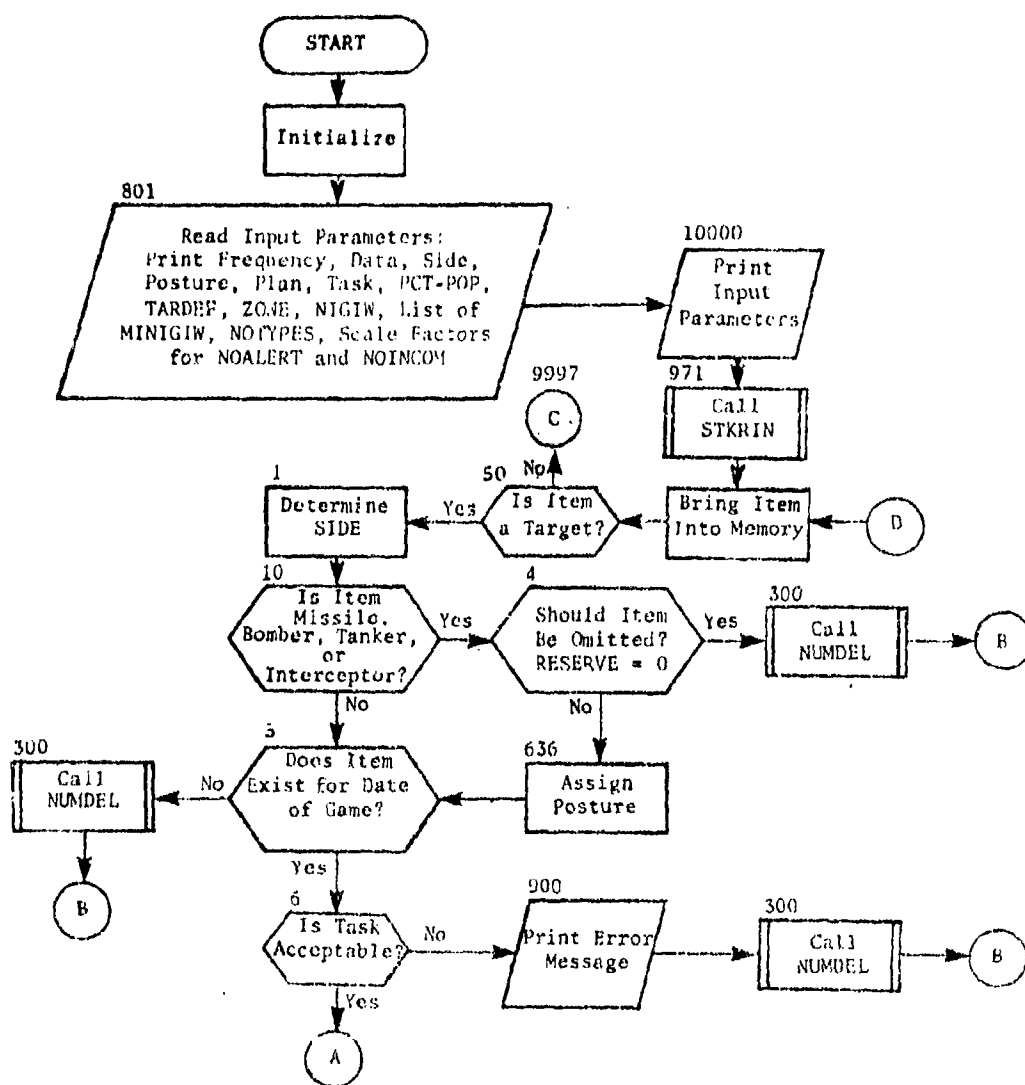


Fig. 101. Subroutine DBMOD
(Sheet 1 of 2)

DISTRIBUTION

<u>Addressee</u>	<u>Copies</u>
NMCSSC Codes	
B121	3
B122 (atock)	6
B200	1
B210	2
B220	19
B600	1
DCA Codes	
260 (original document only, no subsequent changes)	1
920	1
950	1
OJCS	
Studies, Analysis and Gaming Agency, ATTN: SFD, Room ID957, Pentagon, Washington, D.C. 20301	5
Commander in Chief, North American Air Defense Command, ATTN: NPPG, Ent Air Force Base, Colorado 80912	2
Commander, U.S. Air Force Weapon Laboratory (AFSC), ATTN: AWL, Kirtland Air Force Base, New Mexico 87117	2
Director, Strategic Target Planning, Offutt Air Force Base, Nebraska 68113	2
Chief of Naval Operations, ATTN: OP963G, Room 5E531, Pentagon, Washington, D.C. 20350	2
Defense Documentation Center, Cameron Station, Alexandria, Virginia 22314	12
	<u>60</u>

UNCLASSIFIED

Security Classification

DOCUMENT CONTROL DATA - R & D		
(Security Classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)		
1. ORIGINATING ACTIVITY (Corporate author) National Military Command System Support Center (NMCSSC) Defense Communications Agency (DCA) The Pentagon Washington, DC 20301		2a. REPORT SECURITY CLASSIFICATION
		2b. GROUP
3. REPORT TITLE The NMCSSC Quick-Reacting General War Gaming System (QUICK) Programming Specifications Manual, Volume I, Data Input Subsystem		
4. DESCRIPTIVE NOTES (Type of report and inclusive dates) N/A		
5. AUTHOR(S) (First name, middle initial, last name) NMCSSC: Yvonne Mapily Donald F. Webb		Lambda Corp: Betty J. Ellis Jack A. Sasseen
6. REPORT DATE 29 February 1972	7a. TOTAL NO. OF PAGES 486	7b. NO. OF REFS 4
8a. CONTRACT OR GRANT NO. DCA 100-70-C-0065	9a. ORIGINATOR'S REPORT NUMBER(S) NMCSSC COMPUTER SYSTEM MANUAL CSM PSM 9A-67	
8b. PROJECT NO. NMCSSC Project 831	9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report) None	
10. DISTRIBUTION STATEMENT This document is approved for public release; its distribution is unlimited.		
11. SUPPLEMENTARY NOTES		12. SPONSORING MILITARY ACTIVITY National Military Command System Support Center/Defense Communications Agency The Pentagon, Washington, DC 20301
13. ABSTRACT This is one of three volumes describing the computer programming specifications for the Quick-Reacting General War Gaming System (QUICK). This volume addresses computer programs of the QUICK Data Input Subsystem. It is intended to serve as the basis for program maintenance activities. Accordingly, it describes the program functions and contains flow charts for each program and subprogram of the Data Input Subsystem. Based upon suitable data base and user control parameters, QUICK will generate individual bomber and missile plans suitable for war gaming, and simulate the planned events. The generated plans are of a form suitable for independent review and revision. Subsequently, the planned events are simulated; various statistical summaries are produced to reflect the results of the war game. A variety of force postures and strategies can be accommodated. QUICK is documented extensively in a set of Computer System Manuals (series 9-67) published by the National Military Command System Support Center (NMCSSC), Defense Communications Agency (DCA), The Pentagon, Washington, DC 20301.		

DD FORM 1473

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