ED 057 829 LI 003 326

AUTHOR Hurlburt, Charles E.; And Others
TITLE The Intrex Retrieval System Software.
INSTITUTION Massachusetts Inst. of Tech., Cambridge. Electronic Systems Lab.
SPONS AGENCY Council on Library Resources, Inc., Washington, D.C.; National Science Foundation, Washington, D.C.
REPORT NO ESL-R-458
PUB DATE 15 Sep 71
NOTE 633p.; (16 References)

EDRS PRICE MF-$0.65 HC-$23.03
DESCRIPTORS *Computer Programs; *Computers; Data Bases; *On Line Systems; *Programing; Programing Languages
IDENTIFIERS Computer Software; *Project Intrex

ABSTRACT

The report describes the general structure of the Intrex Retrieval Systems and each of the component subroutines. The report is not an introduction to Intrex. In addition to a general description, the report covers the following topics: (1) system architecture, (2) software details, (3) command control logic, (4) list manipulation logic, (5) Intrex utilities, (6) CTSS utilities, (7) data base generation, (8) supporting software and (9) Intrex beyond 1971. The appendices are: (A) Summary of Common References, (B) Data Base Formats, (C) The Intrex Environment, (D) Message Text, (E) Subroutine Linkages (called by), (F) Subroutine Linkages (calls), (G) Glossary of Intrex Terms, (H) Data Base Generation Procedure and (I) Index to Subroutines in Chapter III. (MM)

September 15, 1971

Report ESL-R-458

THE INTREX RETRIEVAL SYSTEM SOFTWARE

by

Charles E. Hurlburt
Michael K. Molnar
and
Charles W. Therrien

The research reported herein was made possible through the support extended the Massachusetts Institute of Technology, Project Intrex at the Electronic Systems Laboratory under a research grant from the Council on Library Resources, Inc., designated as M.I.T. DSR Project Number 27808 and through Grant GN 774 from the National Science Foundation as M.I.T. DSR Project Number 71144.

Electronic Systems Laboratory
Department of Electrical Engineering
Massachusetts Institute of Technology
Cambridge, Massachusetts, 02139

ABSTRACT

This report describes the software of the Intrex Retrieval System. The intent of the report is both to expose the general structure of the System and to describe in detail each of the component subroutines. The report is not intended to be an introduction to Intrex.

ACKNOWLEDGMENT

The authors wish to express their gratitude to Messrs. R.S. Marcus and P. Kugel for their critical review and contributions. Thanks are also due to the publications and drafting staff of the Electronic Systems Laboratory and especially to Miss Catherine Griffin who did the major part of the typing. Finally it should be noted that the design of the Intrex Retrieval System software is a product of many Intrex staff members, past and present, not necessarily reflected in the authorship of this report. Those who led the design effort were Messrs. Marcus and Kugel and Mr. R. L. Kusik, who in addition, was responsible for much of the program implementation before his departure from the project. Any documentation of the software would be incomplete without acknowledgment of their contributions.

TABLE OF CONTENTS

		Page
I.	INTRODUCTION	1
1.1	General Description	1
1.1.1	The Augmented Catalog	4
1.1.2	Free Vocabulary In-Depth Indexing	4
1.1.3	Stemmed Terms in an Inverted File	7
1.1.4	Instructional and Monitoring Facilities	7
1.2	Programming Techniques and Strategies	8
1.3	Contents of the Report	9
II.	SYSTEM ARCHITECTURE	11
2.1	Overview of Software	11
2.1.1	Initialization	12
2.1.2	Command Interpretation	15
2.1.3	Inverted File Search	18
2.1.4	Catalog Output	20
2.2	Examples of Command Processing	21
2.3	Common Structure and Variables	25
2.4	Data Base Structure	26
2.4.1	Catalog Files	28
2.4.2	Inverted Files	32
2.4.3	Reference List Pointers	34
2.4.4	Fiche Direct	34
2.5	Overlay System	35
III.	SOFTWARE DETAILS*	38
3.1	System Control Logic	40
3.1.1	Supervision	40
3.1.2	Fixed Parameter and Data Base Initialization	42
3.1.3	Session Initialization	65
3.1.4	Logging	71
3.1.5	Time Controls	78

*See also "Subroutines Described in Chapter III" following this table of contents.

			Page
	3.1.6	Monitor File Control	87
	3.1.7	Typing Controls	94
	3.1.8	Interrupt Controls	129
	3.1.9	Overlay Controls	137
	3.1.10	Error Controls	144
3.2	COMMAND CONTROL LOGIC		154
	3.2.1	General Command Processing	154
	3.2.2	Subject/Title Command Interpretation	165
	3.2.3	Author Command Interpretation	176
	3.2.4	Primary Search Control	186
	3.2.5	Inverted File Lookup	211
	3.2.6	Search Command Play-Back	223
	3.2.7	Output Command Interpretation	232
	3.2.8	Output Command Controls	237
	3.2.9	Miscellaneous Command Controls	265
3.3	LIST MANIPULATION LOGIC		277
	3.3.1	Document Selection	277
	3.3.2	Naming and Restoring	282
	3.3.3	Saving and Using	302
	3.3.4	Boolean Operations	320
3.4	INTREX UTILITIES		351
	3.4.1	Free Storage Controls	351
	3.4.2	Code Conversion	367
	3.4.3	File Manipulation	379
	3.4.4	Character-String Manipulation	381
	3.4.5	Miscellaneous Utilities	403
3.5	CTSS UTILITIES		420
	3.5.1	Disk I/O	420
	3.5.2	Buffered Disk I/O	436
	3.5.3	File Status	440
	3.5.4	Disk I/O Errors	445
	3.5.5	Console I/O	447
	3.5.6	Data Conversion	453
	3.5.7	Miscellaneous Utilities	459

3.6 AED Utilities Page 486
3.6.1 Miscellaneous Utilities 486

IV. DATA BASE GENERATION 492
4.1 DRYRUN, WETRUN 493
4.2 MASH 501
4.3 STEMER SAVED 504
4.4 SORT SAVED 508
4.5 IFGEN SAVED 509
4.6 IFTEST SAVED 519
4.7 IFLIST SAVED 523

V. SUPPORTING SOFTWARE 532
5.1 DIRGEN 532
5.2 PRINTM, PRNASC 537
5.3 UPDIR 539

VI. INTREX BEYOND 1971 541
6.1 Modifications to the Intrex System 541
6.1.1 Extended Primary Search Capability 541
6.1.2 Extended Output Features 543
6.1.3 Other Search Aids 544
6.1.4 Data Base Update 545
6.2 Exportation of Intrex 546
6.3 Future Generations of Intrex 546

APPENDICES
A. Summary of Common References 550
1. Parameter Option Table 550
2. System State Table 555
3. Command List 557
B. Data Base Formats 559
1. Catalog 559
2. Inverted Files 562
3. List Pointers 565
4. Fiche Direct 567

		Page
C.	The Intrex Environment	568
	1. The Time-Sharing System	568
	2. The AED Language	570
D.	Message Text	571
E.	Subroutine Linkages (called by)	582
F.	Subroutine Linkages (calls)	591
G.	Glossary of Intrex Terms	605
H.	Data Base Generation Procedure	610
I.	Index to Subroutines in Chapter III	612

7

SUBROUTINES DESCRIBED IN CHAPTER III

3.1 System Control Logic

3.1.1 Supervision

3.1.1.1 SUPER

3.1.2 Fixed Parameter and Data Base Initialization

3.1.2.1 SEGINT

3.1.2.2 INITDB

3.1.2.3 INIFIX

3.1.2.4 INIT2

3.1.2.5 PREP

3.1.2.6 TABLE

3.1.2.7 RPRIME

3.1.2.8 RNGNAM

3.1.2.9 LEGFLD

3.1.2.10 FLDNAM

3.1.2.11 STANDL

3.1.2.12 FIELDS

3.1.2.13 INIVAR

3.1.2.14 IFSINT

3.1.2.15 INIEND, REND, ENDTAB

3.1.2.16 GIVTAB

3.1.3 Session Initialization

3.1.3.1 DYNAMO

3.1.3.2 INXCON

3.1.3.3 OPFILE, CLFILE

3.1.3.4 KILFAP

3.1.4 Logging

3.1.4.1 GO

3.1.4.2 SIGNIN

3.1.4.3 EXIT

3.1.4.4 QUIT

3.1.5 Time Controls
3.1.5.1 MONINT
3.1.5.2 MONTIM
3.1.5.3 TIME
3.1.5.4 SUMOUT
3.1.5.5 TRANS
3.1.6 Monitor File Control
3.1.6.1 INIMON, INMON2
3.1.6.2 INIDSK
3.1.6.3 MONTOR
3.1.6.4 ASIDE, ASSET
3.1.7 Typing Controls
3.1.7.1 INITYP
3.1.7.2 TYPEIT
3.1.7.3 LOGMES
3.1.7.4 BCDASC
3.1.7.5 INDENT
3.1.7.6 INCHAR
3.1.7.7 TYPASH
3.1.7.8 TRASH, PUTOUT
3.1.7.9 PRT12
3.1.8 Interrupt Controls
3.1.8.1 ININT
3.1.8.2 INTONE
3.1.8.3 INTTWO
3.1.8.4 LISTEN
3.1.9 Overlay Controls
3.1.9.1 SYSGEN
3.1.9.2 CALLIT
3.1.9.3 SENTRY
3.1.9.4 MAINBD
3.1.9.5 LINKUP
3.1.10 Error Controls
3.1.10.1 ERRGO
3.1.10.2 SETRTN
3.1.10.3 INXSUB

3.2 Command Control Logic
3.2.1 General Command Processing
3.2.1.1 INICON
3.2.1.2 CLP
3.2.1.3 GETLIN
3.2.1.4 NEXITM
3.2.1.5 LOOKUP
3.2.2 Subject/Title Command Interpretation
3.2.2.1 INIS.T
3.2.2.2 SUBJ.
3.2.2.3 TITLE
3.2.2.4 S.T
3.2.2.5 STEM
3.2.3 Author Command Interpretation
3.2.3.1 INIAUT
3.2.3.2 AUTHOR
3.2.3.3 GATP
3.2.3.4 LN
3.2.3.5 AI
3.2.4 Primary Search Control
3.2.4.1 SEARCH
3.2.4.2 SSRCH, TSRCH, STRCH
3.2.4.3 ATSCRN
3.2.4.4 REORD, MEADIR
3.2.4.5 ASRCH
3.2.4.6 CLEANP, STCLN, ACLN
3.2.4.7 DELIST
3.2.4.8 DRPDTR
3.2.4.9 FCLEAN
3.2.5 Inverted File Lookup
3.2.5.1 IFSRCH
3.2.5.2 LOCSEC
3.2.5.3 NAM5
3.2.5.4 MATAFF

3.2.6 Search Command Play-Back

3.2.6.1 INIEVL
3.2.6.2 EVAL
3.2.6.3 GETEND

3.2.7 Output Command Interpretation

3.2.7.1 INIOUT
3.2.7.2 OUT.
3.2.7.3 IN.

3.2.8 Output Command Controls

3.2.8.1 FSO
3.2.8.2 NEWPT
3.2.8.3 INIFLD
3.2.8.4 GETINT
3.2.8.5 FSOCLN
3.2.8.6 GETFLD
3.2.8.7 GETTAB
3.2.8.8 TRETRI
3.2.8.9 SPCTRN
3.2.8.10 TABLK
3.2.8.11 STBL
3.2.8.12 OTBL
3.2.8.13 WRHGH
3.2.8.14 INIRNG
3.2.8.15 RANGE
3.2.8.16 ATLCLN

3.2.9 Miscellaneous Command Controls

3.2.9.1 INIVRB
3.2.9.2 COMENT
3.2.9.3 LIBRY
3.2.9.4 WRT
3.2.9.5 LONG
3.2.9.6 SHORT
3.2.9.7 INFO
3.2.9.8 SEEMAT

11

3.3 List Manipulation Logic

3.3.1 Document Selection

3.3.1.1 NUMBER

3.3.2 Naming and Restoring

3.3.2.1 INIRES

3.3.2.2 NAME

3.3.2.3 TABENT

3.3.2.4 CHKNAM

3.3.2.5 RESTOR

3.3.2.6 LIST

3.3.2.7 DROP

3.3.2.8 CONNAM

3.3.3 Saving and Using

3.3.3.1 SAVE

3.3.3.2 CHKSAV

3.3.3.3 USE

3.3.3.4 MOVEIT

3.3.3.5 CONDIR

3.3.3.6 LISTSL

3.3.3.7 LISFIL

3.3.4 Boolean Operations

3.3.4.1 AND.

3.3.4.2 WITH.

3.3.4.3 OR.

3.3.4.4 NOT.

3.3.4.5 ANDER, GETBUF

3.3.4.6 GETLIS

3.3.4.7 BUFSCN

3.4 Intrex Utilities

3.4.1 Free Storage Controls

3.4.1.1 FREE

3.4.1.2 FRET

3.4.1.3 FREZ

3.4.1.4 CNTLOC

3.4.1.5 SIZE

3.4.1.6 FRER

3.4.1.7 FSIZE

3.4.1.8 FRESET

3.4.1.9 FTRACE

3.4.1.10 FRALG

3.4.2 Code Conversion

3.4.2.1 .C.ASC
3.4.2.2 INI.C.
3.4.2.3 INTASC
3.4.2.4 OCTASC
3.4.2.5 TSSASC
3.4.2.6 ASCTSS
3.4.2.7 ASCIT6
3.4.2.8 CTSIT6
3.4.2.9 ASCITC
3.4.2.10 ASCINT
3.4.2.11 CHKNUM

3.4.3 File Manipulation

3.4.3.1 FILCNT

3.4.4 Character-String Manipulation

3.4.4.1 GET
3.4.4.2 INC
3.4.4.3 PUT
3.4.4.4 GET6
3.4.4.5 INC6
3.4.4.6 PUT6
3.4.4.7 GET12
3.4.4.8 INC12
3.4.4.9 COPY
3.4.4.10 DIST
3.4.4.11 FIND
3.4.4.12 COMPAR
3.4.4.13 COMPUL
3.4.4.14 MATCH
3.4.4.15 GETSET
3.4.4.16 GETINC
3.4.4.17 PUTINC
3.4.4.18 INC1
3.4.4.19 DEC1

3.4.5 Miscellaneous Utilities

3.4.5.1 DNSORT
3.4.5.2 FAPDBG
3.4.5.3 NAP
3.4.5.4 TBSRCH
3.4.5.5 VSRCH
3.4.5.6 SHIFT
3.4.5.7 STRACC, PRSTRA
3.4.5.8 TESTMO
3.4.5.9 TSPOT, TRAC
3.4.5.10 TIMEIN, TOUT, TOTTIM
3.4.5.11 WHEN

3.5 CTSS Utilities

3.5.1 Disk I/O

3.5.1.1 OPEN
3.5.1.2 CLOSE
3.5.1.3 BUFFER
3.5.1.4 RDFILE
3.5.1.5 RDWAIT
3.5.1.6 WRFILE
3.5.1.7 WRWAIT
3.5.1.8 TRFILE
3.5.1.9 FWAIT

3.5.2 Buffered Disk I/O

3.5.2.1 BFOPEN
3.5.2.2 BFREAD
3.5.2.3 BFWRIT
3.5.2.4 BFCLOS

3.5.3 File Status

3.5.3.1 CHFILE
3.5.3.2 DELFIL
3.5.3.3 FSTATE

3.5.4 Disk I/O Errors

3.5.4.1 FERRTN
3.5.4.2 IODIOG

3.5.5 Console I/O

3.5.5.1 RDFLXA
3.5.5.2 RDFLX
3.5.5.3 WRFLXA
3.5.5.4 WRFLX
3.5.5.5 SETFUL
3.5.5.6 SETBCD

3.5.6 Data Conversion

3.5.6.1 BCDEC
3.5.6.2 DEFBC
3.5.6.3 DERBC
3.5.6.4 OCABC
3.5.6.5 RJUST
3.5.6.6 BZEL

3.5.7 Miscellaneous Utilities

3.5.7.1 DORMNT
3.5.7.2 SLEEP
3.5.7.3 WAIT
3.5.7.4 GETBRK
3.5.7.5 SETBRK
3.5.7.6 SAVBRK
3.5.7.7 GETMEM
3.5.7.8 SETMEM
3.5.7.9 WHOAMI
3.5.7.10 SETWRD
3.5.7.11 GETWRD
3.5.7.12 SETBLP
3.5.7.13 CHNCOM
3.5.7.14 GETCOM
3.5.7.15 COMARG
3.5.7.16 LDOPT
3.5.7.17 SETSYS
3.5.7.18 GETSYS
3.5.7.19 GNAM
3.5.7.20 RSCLCK
3.5.7.21 JOBTM
3.5.7.22 GETIME
3.5.7.23 GETTM
3.5.7.24 SCLS
3.5.7.25 NCOM

3.6 AED Utilities

3.6.1 Miscellaneous Utilities

3.6.1.1 WFLX
3.6.1.2 WFLXA
3.6.1.3 ISARGV
3.6.1.4 ISARG
3.6.1.5 GETP
3.6.1.6 OCTTOI

LIST OF FIGURES

		Page
1.1	Basic System Diagram for Intrex	3
1.2	Information Fields in the Catalog	5
1.3	GA Sample Catalog Record	6
2.1	The Initialization Module	13
2.2	Tasks of Initialization	14
2.3	The Command Module	17
2.4	The Search Module	18
2.5	The Output Module	21
2.6	Sequence of Subroutine Calls for the Command "s ion particle"	22
2.7	Sequence of Subroutine Calls for the Command "output title fiche"	23
2.8	Sequence of Subroutine Calls for a Combined Command	24
2.9	Common File Structure	28
2.10	Data Base Structure	29
2.11	Catalog Record Segments	30
2.12	Catalog Record Format	31
2.13	Format of I.F. Directories	33
2.14	Partitioning of Intrex Core	36
2.15	Flow Chart of System Generation Process	37
3.1	Subject/Title Search Structure	172
3.2	Author Search Structure	178
3.3	Diagram of Memory (Free Storage)	352
4.1	DRYRUN/WETRUN (Main Flow of Control)	498
4.2	Loading Sequence	500
4.3	Sample Page of I.F. Listing	524
B.1	Inverted File Format	563
B.2	Augmented List Pointer Format	566

I. INTRODUCTION

This report describes the computer programs or software for the Intrex Retrieval System. The purpose of the documentation is twofold. First, it is intended to expose the general structure of the system so that it may serve as a model or prototype upon which other similar, but more operational, systems can be based. Secondly, it is intended to describe the Intrex software to a level of detail sufficient for a reader to understand the purpose and logic of each subroutine. This detailed description is felt to be especially important to the analyst or system designer who may want to appreciate the level of complexity and hence the computational load imposed on a computer system by the sophisticated mode of operation of Intrex. It is also important for those who must understand and/or maintain Intrex in its present implementation. In order to accomplish these purposes, everything short of the actual program listings has been included. It should be recognized at the outset that this report is not meant to be an introduction to Intrex. Several other publications[1-5] are available which serve that purpose. In order to arrive at a common starting point, however, the present chapter does begin with some material of an introductory nature. An attempt has been made there to highlight certain aspects which bear particular relevance to the software of the Intrex Retrieval System.

1.1 General Description

Intrex is an experimental, pilot-model, machine-oriented bibliographic storage and retrieval system. The system includes a computer-stored catalog of about fifteen thousand journal articles in selected fields of Materials Science and Engineering and the full text of these documents stored on microfiche. The heart of the system is the Intrex retrieval programs which provide for searching through the data base to gather lists of documents on topics requested by users.

The Intrex programs operate in an on-line interactive mode with users in a dialog with the machine. Users control the actions of Intrex by issuing typed English <u>commands</u> within the Intrex language. Such commands permit the user to initiate searches of the catalog by subject, title, or author, to combine lists of documents resulting for subsequent searches, to save combined lists, and to request output of information pertinent to documents on the lists. Intrex in turn responds to these commands with the lists of documents found or the other requested information. In this way the user of Intrex designs his own search strategy from the basic functions available to him. The immediate feedback given the user insures that each such design will be unique and so can be better suited to the particular search.

A system diagram for Intrex is shown in Figure 1.1. The Intrex programs run on the MIT-developed CTSS time sharing system (see Appendix C). CTSS provides for sharing the resources of an IBM 7094 computer among thirty or so online users. Each user who is exercising Intrex has his own copy of the Intrex programs so his actions are completely independent of the other users. The time sharing system provides for allocating central processor time to the users, and swapping their programs in and out of core in such a way that each user appears to be receiving the full attention of the large computer. Intrex terminals also tie into the separate text-access subsystem, which consists of the document collection on microfiche and the machinery for selecting the microfiche corresponding to a document, scanning the fiche, and sending the image to the appropriate terminal. The text access subsystem also provides for microfilm copy of the full text. The microfilm is produced within 90 seconds after a request has been initiated at the user's terminal and so can be picked up by the user immediately following his terminal session.

Several different types of terminals[6] are supported by Intrex ranging from an IBM 2741 typewriter to a specially designed remote buffered CRT device with expanded character set and subscripting and superscripting capability. The terminals of this type are interfaced to CTSS and the text access subsystem through a Varian 620i minicomputer the software for which is to be described in a separate report. For full-text availability, any type of terminal must include a storage tube display. In the case of the Intrex "combined" terminal, this display is used both for interaction with the catalog and for the display of full text. Users whose terminals do not include the special storage tube display may interact via the Intrex programs with the catalog alone. This type of operation is more typical at distant locations since the text access subsystem can currrently transmit only over relatively short distances.

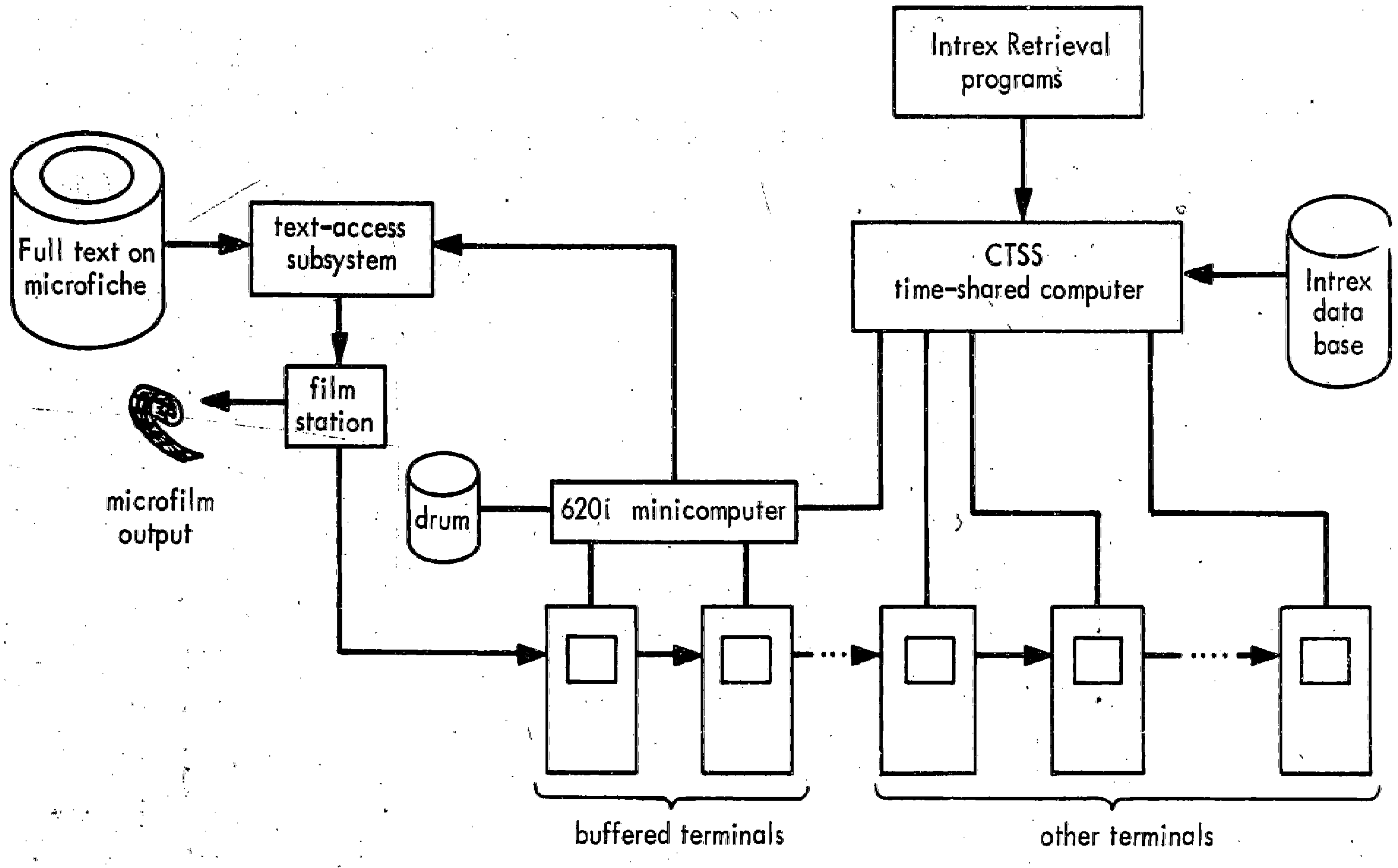


Fig. 1.1 Basic System Diagram for Intrex

Intrex, being an experimental retrieval system, has several novel features that distinguish it from more traditional operational retrieval systems. One of these features is the availability of full text at terminals. Others relate more directly to the retrieval programs and so are discussed separately below.

1.1.1 The Augmented Catalog

The Intrex data base consists of an augmented library catalog with a catalog record for each document in the system.[4] The catalog is stored on the disk of the CTSS time sharing system. The catalog records contain the fifty fields shown in Figure 1.2. These include, in addition to the usual catalog entries, the abstract or excerpts, table of contents, and several other items of information. Not all of these fields would necessarily be required in a fully operational system — but it is part of the Intrex experiment to determine which fields are of most utility to users and which could safely be eliminated. The fields starred in Figure 1.2 are used for journal records; the remaining fields pertaining to books, conference papers, and other types of documents. A typical catalog record is shown in Figure 1.3. The numbers at the left between slashes are the field numbers.

The catalog records, which comprise the major part of the data base and average about two thousand characters in length are compacted in order to conserve disk space. The character strings are stored in a special nine-bit code where one code word represents wherever possible a pair of characters or digram.[7] When compared to standard ASCII coding of individual characters, this scheme reduces the storage required by 31 to 45 percent.*

1.1.2 Free Vocabulary In-Depth Indexing

Intrex uses a free vocabulary indexing for documents, i.e., any words or symbolic notation (such as H_2O) may be included in the index terms. Note in Figure 1.3 (Field 73) that the subject index terms are generally not just words but entire phrases. Note also the number of such index terms, a result of the deep indexing. The inverted file (see following Section) to the catalog is constructed from the stemmed words or symbols found in the phrases.

*The percent reduction depends on whether the ASCII code is packed into a nine-bit byte or into a seven-bit byte as is possible with the 36-bit computer word.

I. CATALOG CONTROL FIELDS

*1. Document Number
*2. Document Selection
*3. Input Control
*4. On-Line Date
*5. Microfiche Location

II. PHYSICAL DOCUMENT CONTROL FIELDS

10. L. C. Card Number
11. Library Location
12. Serial Holdings

III. DESCRIPTIVE CATALOGING FIELDS

*20. Main Entry Pointer
*21. Personal Names
*22. Personal Affiliations
*23. Corporate Names
*24. Title
25. Coden Title
26. Edition Statement
27. Publisher
28. Place of Publication
29. Dates of Publication
*30. Medium
*31. Format
32. Pagination
*33. Illustrations
34. Dimensions
35. Serial Frequency
*36. Language of Document
*37. Language of Abstract
*38. Series Statement
*39. Report/Patent Numbers
*40. Contract Statement
41. Supplement Referral
42. Errata
*43. Thesis
44. Variants
45. Titles of Variants
46. Article Receipt Date
*47. Analytical Citation
48. Abstract Services
49. Cost-Text Access
50. Commercial Cost

IV. SUBJECT CONTENT FIELDS

*65. Author's Purpose
*66. Level of Approach
*67. Table of Contents
*68. Special Features
*69. Bibliography
*70. Excerpts
*71. Abstracts
72. Reviews
*73. Subject Indexing

V. ARTICLE CITATION FIELD

80. References/Citations

VI. USER FEEDBACK FIELD

85. User Comments

Fig. 1.2 Information Fields in the Catalog

```
//1/  3644

//2/  A24

//5/  175-A1-B5

//20/  1

//30/  1

//47/  PHRVA; v.161,no.2,091067, pp.350-366.

//21/  Hempstead, Robert D. (TA);
       Lax, Melvin (JA)

//22/  Bell Telephone Laboratories, `Murray Hill`, `N.J.`/
       University of `Illinois`, `Urbana`; Dept. of Physics;
       Bell Telephone Laboratories, `Murray Hill`, `N.J.`

//23/  M.I.T., `Cambridge`, `Mass.` (BT);
       #American Physical Society Meeting, `New York`, 1966

//24/  Classical noise, part 4; noise in self-sustained oscillators
       near threshold

//43/  2,090065. M.S. (Electrical Engineering)

//31/  bb

//46/  032867

//36/  e

//37/  e

//33/  illus.

//69/  lf(38)

//67/   1. Introduction (p.350)
        2. The Langevin and Fokker-Planck equations (p.352)
        3. Transformation to polar coordinates (p.354)
        4. Integration over the phase variable (p.354)
        5. Steady-state amplitude probability distribution (p.354)
        6. Calculation of the power spectra (p.356)
        7. Eigenfunction expansion (p.358)
        8. Accuracy of computations (p.362)
        9. Summary (p.362)
       10. Appendix A.  The laser model (p.363)
       11. Appendix B.  The circuit model (p.364)
       12. Appendix C.  The boundary condition for
           G(r, r*sub 0*; *lambda*, *omega*) (p.366)

//68/  Contains tables of fluctuation data and graphs of power
                spectra and fluctuation potentials

//65/  t

//66/  13

//71/  Because of the relative narrowness of the threshold region, a
       general model for spectrally pure self-sustained oscillators
       (both classical and quantum, including gas lasers) can be
       reduced, in the threshold region, to a rotating-wave Van der
       Pol (RWVP) oscillator... [Body of abstract omitted from this
       illustration because of its length]... Thus the intensity
       fluctuation spectrum is Lorentzian below and well above
       threshold, but more complex in the threshold region. (author)

//73/  mathematical development of classical noise in self-sustained
                 oscillators near oscillation threshold (1);
       Lax-Louisell model for self-sustained oscillator (4);
       Lax-Louisell study of laser noise (3);
       normalized rotating-wave Van der Pol oscillator (2);
       gas laser (4);
       laser noise (0);
       phase, intensity, and amplitude fluctuations in
                 self-sustained oscillators near threshold (2);
       nearly Lorentzian nature of power spectra of noise in
                 self-sustained oscillators near threshold (2);
       exact calculation of power spectra in the normalized RWVP
                 oscillator near threshold by numerical
                 Fokker-Planck methods (2);
       scaled Langevin equation (4);
       Fokker-Planck, Green's function, and eigenfunction
                 methods of calculating power spectra of noise in
                 self-sustained oscillators near threshold (2);
       white-noise sources in a self-sustained oscillator (3);
       steady-state amplitude probability distribution (3);
       one-sided Fourier transform of the spectrum and intensity
                 spectrum of gas lasers (3);
       power spectra boundary conditions (3);
       equation of motion of a self-sustained oscillator (4);
       effect of power output on power spectra of a self-sustained
                 oscillator (3);
       effect of net pump rate on operation of a rotating-wave
                 Van der Pol oscillator (2);
       sinusoidal power spectrum of a rotating-wave Van der Pol
                 oscillator (2);
       linearization methods of calculating power spectra of noise
                 in a self-sustained oscillator outside the
                 threshold region (2);
       nonlinear techniques for calculating power spectra of noise
                 in a normalized rotating-wave oscillator (2);

//3/   10/2,     012968,          1:30-2:25;
       1/7,      012968,          2:40-2:56;
       10/1,     021468,          1:40-1:50;
       5/4,      040168,          11:35-11:50,1:20-1:38;
       -.-
```

Fig. 1.3   Sample Catalog Record

Due to the free vocabulary indexing, the number of postings in the inverted file grows at a certain fraction of the rate of growth of the data base. At our current level of approximately 15,000 documents that growth rate is about four new postings for every five new catalog entries.

Documents for the data base are indexed by trained personnel who scan the document for words and phrases indicative of the subject matter. The extracted phrases and subject terms are classified into five groups or ranges which relate to their coverage of the document. Range 0 terms refer to a general topic under which the document falls. Range 1 terms relate to the main subject of the document, range 2 terms to topics of secondary importance, and range 3 terms to topics of minor importance. Range 4 terms relate to tools or techniques which are used or referred to in the document. Precision of retrieval can be controlled by the user limiting searches to terms within particular ranges.

1.1.3 Stemmed Terms in an Inverted File

In order to avoid an exhaustive search through all the 15,000 catalog records for matching terms each time a search request is made, Intrex uses an Inverted File structure. That is; the actual compilation of lists of documents corresponding to a given search term is done in advance of any searching and only once—during the so-called generation process. In compiling these lists, the words in the search terms are first stemmed; e.g. magnetic, magnetism, magnetize, and magnetizability are all reduced to the root magnet.[8] The document in which a given search term appears is added to the list of documents associated with the stemmed term. Thus when a search for documents is later requested by a user, Intrex needs only to find the stemmed search term in the Inverted File. From there, it directly obtains the list of documents. From that list, it can obtain, upon further request, the actual catalog records. The structure of the Intrex Inverted File is described in greater detail in Chapter II and Appendix B.

1.1.4 Instructional and Monitoring Facilities †

Because the Intrex system was intended for persons with little or no advanced training in its use, as well as for highly trained search specialists, a great deal of effort was expended in providing special techniques of system

† ...tributed by R. S. Marcus

instruction. The instruction aids include a guide to system use that is available both in printed form and online in a form that is displayable in sections under program control. In addition the system provides special instructional messages to guide new users and diagnostic comments to help users detect and correct errors.

To enable the staff to review and analyze system use, facilities for monitoring and recording all interactions with the system were devised. The full dialog of each user is automatically recorded on a disk file and printed out daily. In addition, a user can be monitored from a remote terminal in real time. When in this mode the person at the monitoring console can also communicate with the Intrex user and aid in or even control his use of the computer.

1.2 Programming Techniques and Strategies †

The foregoing features of the Intrex Retrieval System influenced the programming in several ways. The combination of certain features caused complexities in various routines. For example, the requirements for a variety of instructional messages that could easily be changed, an elaborate monitoring facility, the ability of the user to interrupt at any point in the program, and the necessity to adapt to a variety of terminals with different codes and line lengths led to a message-handling routine, TYFEIT (See Section 3.1.7.2) of considerable complexity. The importance of experimentation led to a desire to make all programs flexible and modular so that different features could be easily appended and experimented with. This in turn led to the extensive use of such devices as table-driven subroutines and a data-structure that contains the results of command language parsing and drives the search command modules. Because of our desire to get experiments started as soon as possible, many routines were, initially written in a fairly simple, straightforward way in the AED source language. Later, when analysis revealed that certain routines were resulting in system inefficiencies, some of these were rewritten in machine language and reassembled in larger units.

Some features that were originially planned to be tested have not yet been fully implemented as part of the Intrex system. Thus there are certain

† Contributed by R.S. Marcus

structures within the system which have no current use. For example, the reference words include parameters specifying the word position within a phrase and the word ending. These factors could be used in the search match algorithm. It might be added that, in this case, it was possible to study the utility of these additional features sufficiently well by simulation that it was not necessary to implement them directly. These features would still be desirable however, to increase retrieval system flexibility.

The architecture of the CTSS time-sharing system affected the programming in major ways. For example, the design of the CTSS file system and the consequent inability to directly address physical storage locations, and limited ability to overlap I/O and computation, were critical in our choice of file and directory organization.* The size of the programs and the limitation or core memory to 32K words led directly to the design of the overlay capability described in Sections 2.5 and 3.19. The 36-bit word on the 7094 led to our choice of nine-bit ASCII and later digram codes. The various I/O codes required by CTSS forced certain decisions regarding the storage of data internally and the conversion processes necessary at output. In addition, items not directly tied to CTSS but available on that system influenced our programming structure. Most notably the AED source language (see Appendix C) suggested certain particularly convenient data structures and some generally efficient procedures for use of core memory.

1.3 Contents of the Report

The remainder of this report is divided into five chapters. Chapter II is intended to be an overview of the Intrex programs. The purpose there is to provide a semi-detailed description of the programs on a modular level. A description of the data base is included since an understanding of that is essential to an understanding of the processes involved in searching. Chapter III is an in-depth description of each subroutine of the Intrex retrieval programs. The intent of that chapter is to describe the software at a level that would permit an applications programmer, analyst, or similar person, to follow the actual program listings and understand the logic of each subroutine. Chapter IV deals with auxiliary programs used for generation and updating of the data base. Chapter V

*For a thorough discussion of this point, including design of a more nearly optimum storage configuration on a more modern system, see Kusik[9] and also Goldschmidt.[10]

deals with programs that are used in support of the retrieval programs. The level of detail for Chapters IV and V is intermediate between the level of Chapters II and III. Chapter VI is a discussion of future plans for the Intrex Retrieval System, as they relate both to the current programs and to future generations of Intrex.

II. SYSTEM ARCHITECTURE

The present chapter describes the Intrex retrieval programs on a modular level. Such a description is necessary to establish a framework into which a more detailed description of the modules can fit.

2.1 Overview of Software

The retrieval program consists of a 32,768-word* core image and four approximately 3500-word overlay segments. The current 15,000 document data base consists of an author Inverted File of 120,000 words, a subject Inverted File of 1,400,000 words, a catalog file of 4,200,000 words, and a microfiche directory containing 1500 addresses to the fiche locations of documents.

Approximately 300 different subroutines, containing a total of about 30,000 instructions, make up the retrieval program. About 150 of these subroutines can be described as general purpose utility procedures. They perform such basic functions as console and disk I/O, core storage management, code conversion, and string manipulation. The other 150 subroutines are special purpose subroutines which, by interacting with the utility procedures and with each other, carry out the retrieval functions of INTREX. These subroutines can be clustered functionally into four groups: the initialization, command, search, and output modules. The initialization module starts up the system and logs in the user. The command module accepts commands from the user and either executes them directly or generates a data structure which is later interrogated by the search or output modules. The search module searches the Inverted Files. The output module displays information from the augmented catalog, the fiche directory, or the text access mechanism. The flow of control among these modules is directed by the supervisory subroutine SUPER. Each of these modules is described in more detail below.

*Word size for the IBM 7094 is 36 bits

2.1.1 Initialization

Initialization takes place in three phases. The first phase initializes the subroutines in overlay segments. The second phase completes the setting of parameters which are independent of a particular retrieval session. The third phase prepares the system for a session.

During the first phase of initialization, four loads are performed, each of which consists of the core-resident portion of the system and one of the four overlays. The hierarchical relation of the various subroutines of the initialization module is depicted in Fig. 2.1. Figure 2.2 shows the sequence of functions involved in initialization and the routines performing these functions. When a subroutine has completed its task, it returns to the subroutine on the level just above it. After each load, control is transferred to the main subroutine, SUPER, which in turn transfers control to the overlay generation subroutine, SYSGEN. SYSGEN calls SEGINT, which in turn calls initialization routines associated with subroutines in the overlay section (see Fig. 2.1). SYSGEN then writes out the overlay as a separate disk file and halts. Each of the four segments is generated in the same way. After the fourth segment is generated, the core image is saved. This core image, together with the four overlay files, makes up the retrieval system.

The second phase of initialization performs the remaining initialization tasks that are independent of a particular retrieval session. This phase starts with the execution of the core image saved at the end of the first phase. SYSGEN calls INITDB, which defines areas for the INTREX data structure (see Section 2.3). One word is allocated for the System State Table (SST), which is used as an array of control bits. Fifty words are allocated for the Parameter Option Table (POT), which is used to store important data elements, such as the names of system files. Nine words are allocated for the Command List (CL), which is the beginning of a data structure which will expand or shrink depending on the user's commands. SUPER then calls, INIFIX, which allocates seven 432 word blocks of storage to be used as I/O buffers and calls INIT2, PREP, TABLE, MONINT, and INICON (see Fig. 2.2). SUPER then calls INIVAR which transmits to the console a request for the second names of the Catalog Files and the Inverted Files. After these names have been entered, INIVAR calls IFSINT, which reads into memory the directories for the Inverted Files. IFSINT also calls INIEND, which initializes the ending table

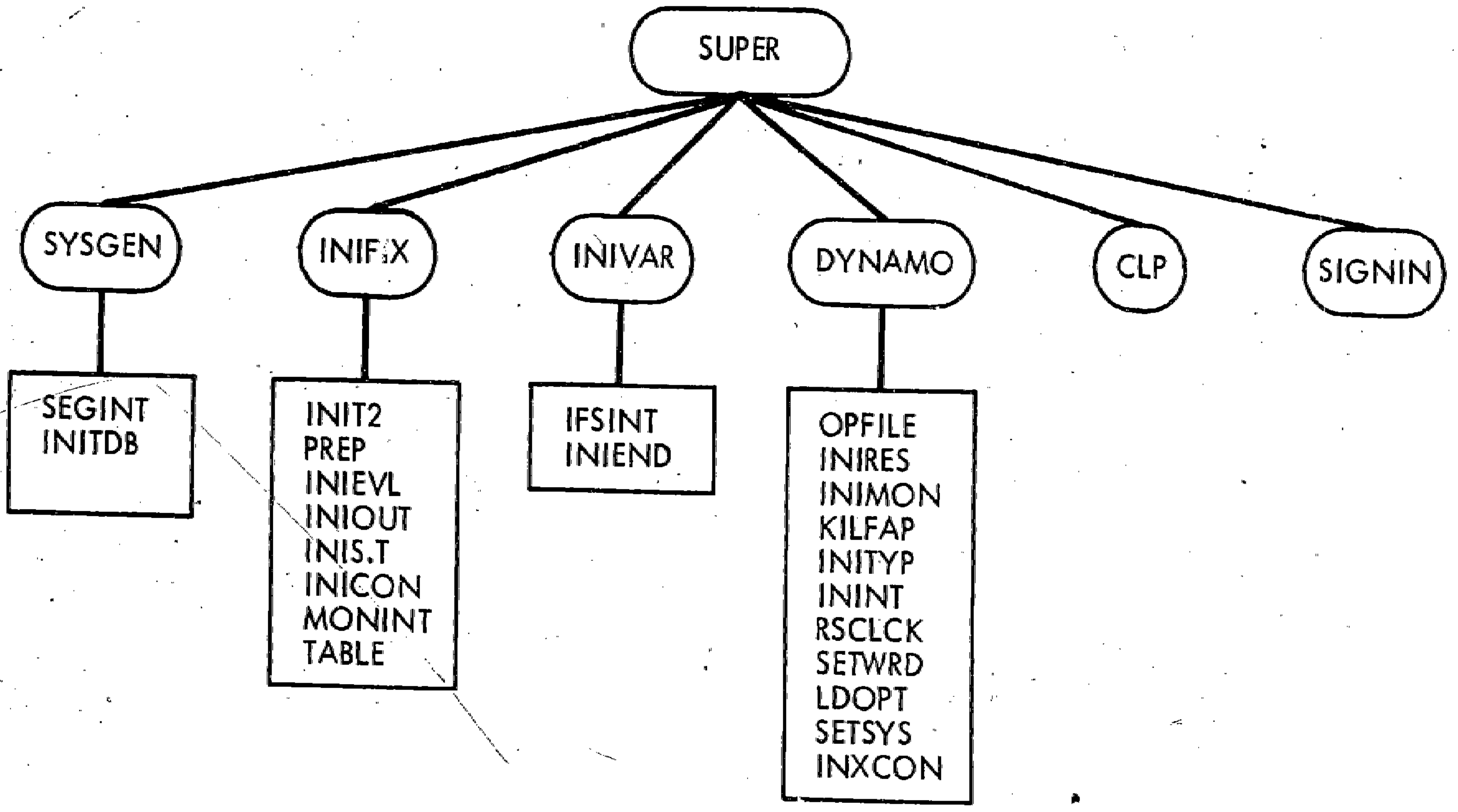


Fig. 2.1 The Initialization Module

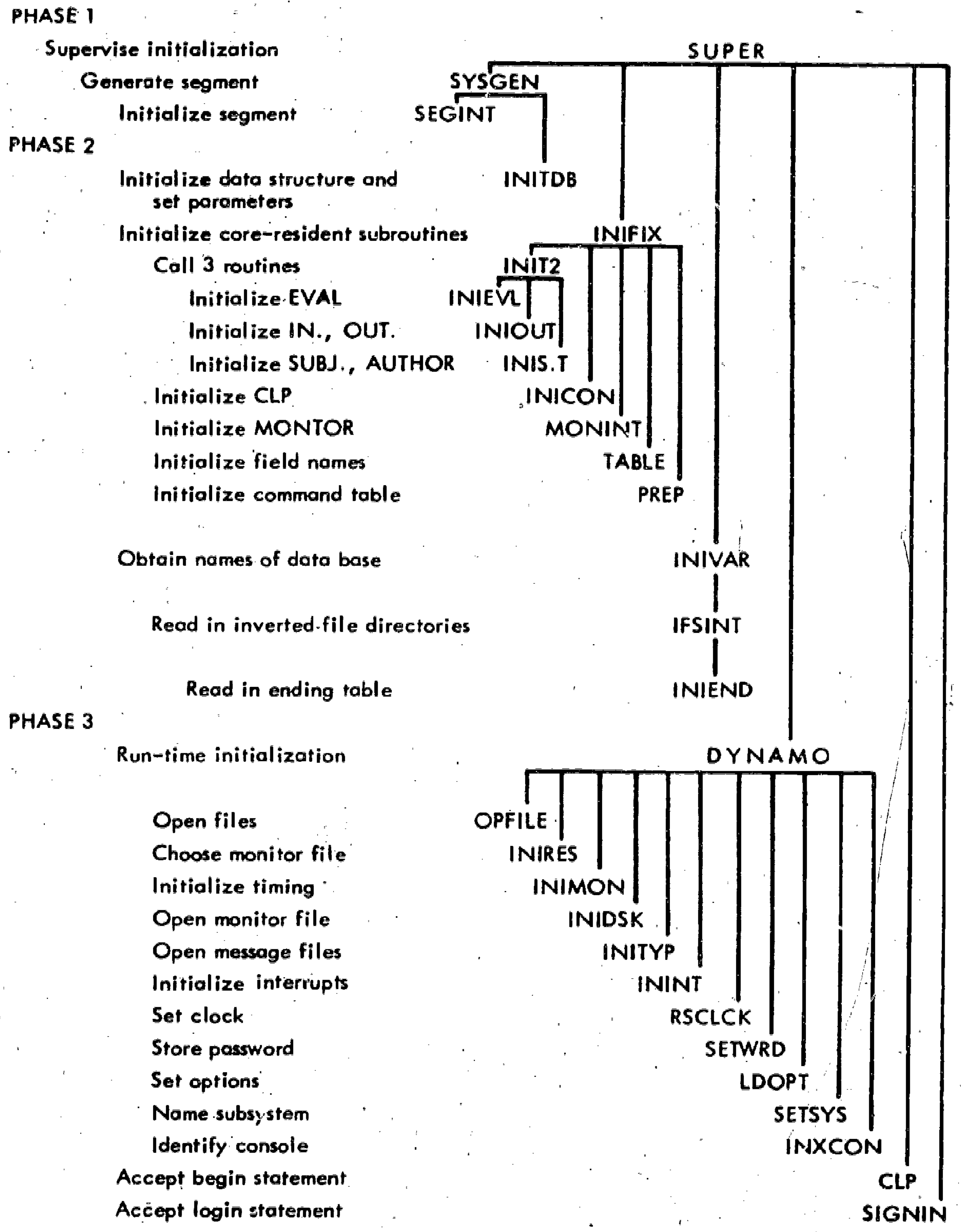


Fig. 2.2 Tasks Performed During Initialization

which is used for stemming search terms. INIVAR returns to SUPER which ends the second phase of initialization by returning control to CTSS. Once again the core image is saved.

The third phase of initialization handles those functions that are related to a particular retrieval session. This phase has one part that sets up the retrieval system for the session and a second part that logs the user into the system. The first part is handled by DYNAMO, which calls a variety of subroutines to complete the system initialization (see Fig. 2.2). OPFILE opens basic system files, including the Catalog Directory and the overlay segments. INIRES initializes the SAVE-RESTORE package and assigns the system a Monitor File. INITYP reads either the long or the short message file directory into core. INIDSK opens the Monitor File. INIMON writes the header information in the Monitor File and ININT sets up the interrupt handling mechanism.

The log-in process begins with SUPER asking the user to type the word "begin". SUPER then calls CLP to await the user's response. If the user types "begin", SUPER calls SIGNIN. SIGNIN asks the user to log in. If the user then types the word "log" followed by a space and any non-blank character string, SIGNIN will accept it as a legitimate log-in. SUPER transfers control to CLP, which waits for a user command.

2.1.2 Command Interpretation

The command interpretation module of Intrex, often referred to as the "Command Language Processor", is based upon the controlling program CLP. The Intrex Supervisor (SUPER) calls CLP at the outset of any new user/system interaction. CLP, in turn, calls several other procedures which accept and interpret the user's input.

The first routine called by CLP is GETLIN, which requests a logical line of user input.* GETLIN informs the user that Intrex is prepared to receive a command by calling the small sub-procedure READY, which out-

*One logical line consists of all the characters typed by a user up to the first carriage return which is not immediately proceded by a hyphen.

puts the "R" or "Ready" message, and if the TIME mode is ON, calls MONTIM to output the CPU and real times used since the last call to READY.

GETLIN then calls the CTSS system subroutine RDFLXA which actually obtains the string of characters typed by the user up to and including the carriage return. GETLIN then processes the character string to delete all characters whose removal is implied by the character-delete (# or back space) and line-delete @. A pointer showing the location and length of the edited input line is returned to CLP.

CLP then extracts the first item or word from the line by calling NEXITM. This procedure looks for specified delimiters, such as space, hyphen, slash, or carriage return, and returns a pointer to the string of characters up to the next such delimiter found. The string pointed to by this call to NEXITM will be, in most cases, an Intrex command.

CLP calls the procedure LOOKUP to compare the first four characters in the string to the list of Intrex commands. If there is a match, then the name of the appropriate procedure to process the command is returned to CLP. CLP then calls CALLIT which uses the name as a key to the overlay segment and location within that segment where the desired processing routine may be found. Control is returned to CLP after execution of the command routine, and NEXITM is recalled to fetch the next command, if any, from the command line.

If the first word in the character string is not a command, it is assumed that this word is an argument to the implied RESTOR command, i.e. the name of a list to be placed in active status. This is indicated by the return from LOOKUP of an error code to CLP, which then calls the procedure RESTOR. If RESTOR indicates that the name does not correspond to any existing list, then CLP informs the user that the word is "not a legal command". When this occurs, CLP returns immediately to the supervisor without processing the rest of the command line.

The subroutines called by CALLIT to perform the actions requested by the user through the various Intrex commands are shown in Fig. 2.3. Their names generally correspond to the name of the command which they implement. The subroutines have been grouped in Fig. 2.3 according to

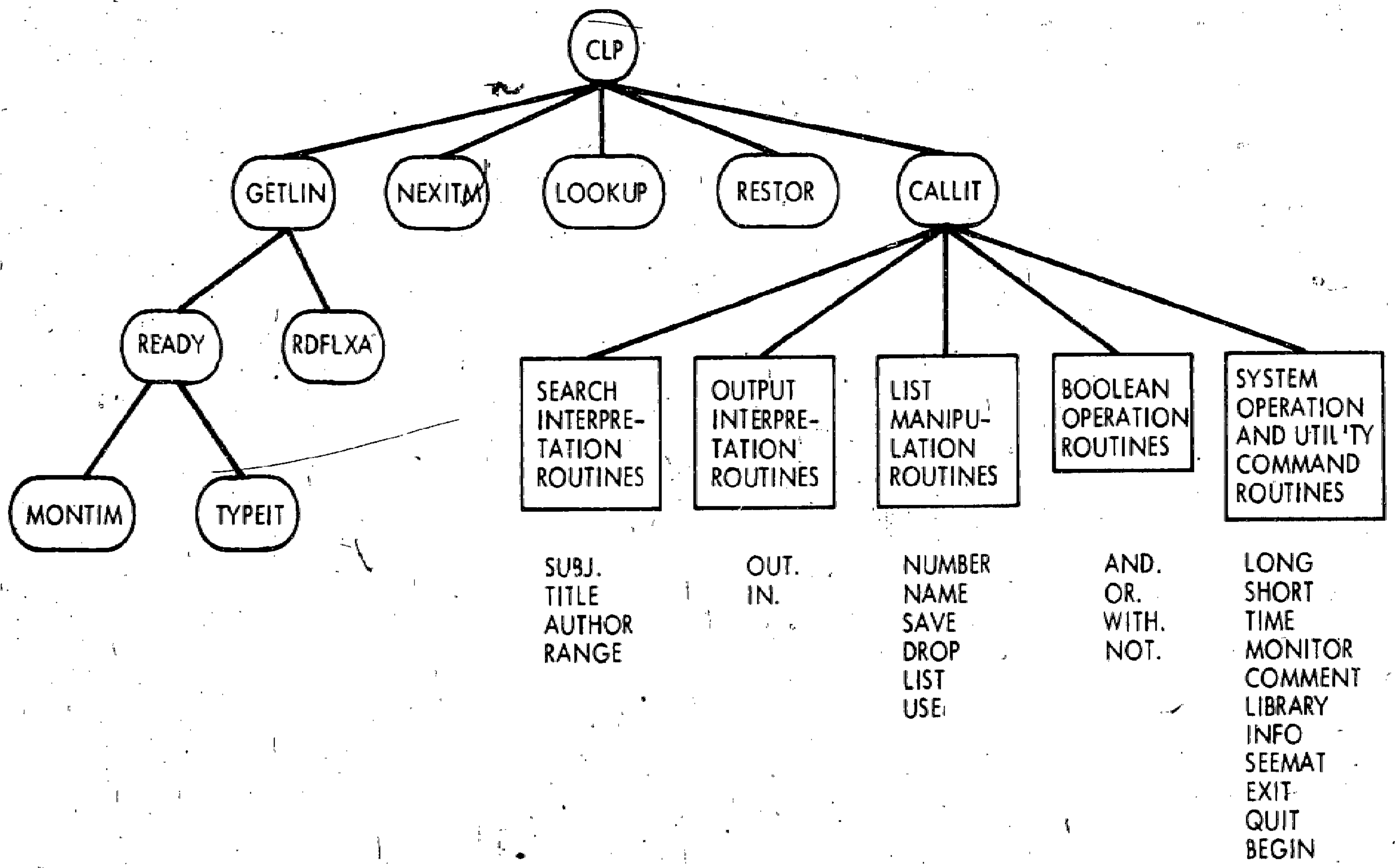

CLP
GETLIN
NEXITM
LOOKUP
RESTOR
CALLIT
READY
RDFLXA
MONTIM
TYPEIT
SEARCH INTERPRE-TATION ROUTINES
OUTPUT INTERPRE-TATION ROUTINES
LIST MANIPU-LATION ROUTINES
BOOLEAN OPERATION ROUTINES
SYSTEM OPERATION AND UTIL'TY COMMAND ROUTINES
SUBJ. TITLE AUTHOR RANGE
OUT. IN.
NUMBER NAME SAVE DROP LIST USE
AND. OR. WITH. NOT.
LONG SHORT TIME MONITOR COMMENT LIBRARY INFO SEEMAT EXIT QUIT BEGIN

Fig. 2.3 The Command Module

functions that they ~~perform~~ for ease of reference; this grouping implies no particular sequence of control.

2.1.3 Inverted File Search

The search module depicted in Fig. 2.4 is responsible for processing the search data structures set up by the interpretation of a search command issued by the user (described in Section 2.1.2). This interpretation takes place in the "Command Processing" module which executes the routines associated with "subject", "title", or "author" commands.

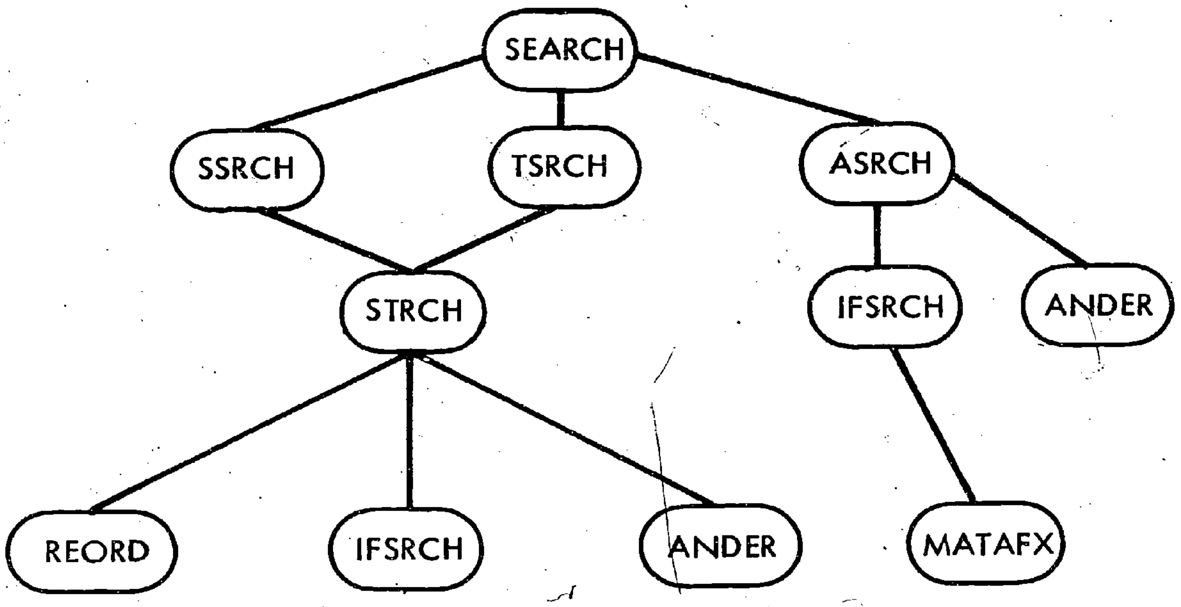


Fig. 2.4 The Search Module

A "System State Table" (SST) flag alerts SUPER that a search has been set up, but not yet executed. SUPER then calls the main search procedure SEARCH, which examines the three main search structure pointers in the Command List (CL). These pointers, if they exist, point to the data structures of one or more of the three types of searches available to the user; subject, title, and author (see Section 2.3).

The presence of a subject-search-form causes SEARCH to call a control routine named SSRCH. Similarly, a title-search-form would prompt SEARCH to call another control routine named TSRCH. These two procedures merely control the mode of the more important routine STRCH which actually processes either a subject term (phrase) or a title term.

The presence of an author-search-form causes SEARCH to call ASRCH, a procedure analogous in function to STRCH.

ASRCH is considerably less complex than STRCH since the former must only process the author's name while the latter must process each word of a subject or title term.

The subject- (or title) search-form constructed by the command processing module contains a pointer to a list of pointers called the "simple search list". Each of these in turn points to an "Inverted File search form" containing data relevant to one word of the search term in the user's request. These search word data structures are in the same order as the words that were typed by the user. STRCH employs a routine named REORD to reorder the pointers to the words to place a word with a short (preferably less than one disk record) list of references at the top to initiate the process described below. This usually reduces the number of reference comparisons necessary during the ensuing list intersections.

The reordered pointers are fed, one-by-one, to the lookup procedure IFSRCH, which is the heart of the search mechanism. With the aid of its associated directories, IFSRCH selects the segment of the Inverted File where it is most likely to find a list name matching the search word in question. An exhaustive list-by-list search of this area is then made until either a matching list name is found or it is determined that no such list is in the file. Either a pointer to the reference list of the matching entry or a "search-failed" code is returned to STRCH. If a search fails on any word, no further searching is done on that request since all words of a search term must be matched for the search to be successful.

The first list pointer returned from a successful call to IFSRCH becomes the "current list" by being placed in one of the components of the Command List, RRL.(CL.).* If a one-word search is being processed, this list will remain the current list and the search is finished. If other search words are part of the user's search request, the list pointers returned by the subsequent calls to IFSRCH will be intersected with the "current list" by the ANDER procedure. This routine produces an output list containing only those references whose document and subject term numbers are found in the references of both the lists in question. If ANDER produces a zero

*For a description of the format of list pointers, see Section 2.4.

length output list (no common document and term number), the entire search is considered a failure and is terminated. If the output list produced by ANDER is of non-zero length, it becomes the new "current list" and the process continues until all search words have been processed.

If an author-search is requested, ASRCH calls IFSRCH which will process the "author-search-form". In this case, IFSRCH will access the Author Inverted Files, which are separate and much smaller than the Subject/Title Inverted Files. If the author-search-form contains a pointer to a set of initials provided in the user's search request, then the procedure MATAFX is called by IFSRCH (if a matching last name is found) to check for a match on the authors initials.

The results of a subject search, title search, and author search are intersected only if these searches had been part of the same request, i.e., given by the user in a combined search command. Should this intersection be necessary, ANDER is called after successful creation of each of the separate title or author lists to effect the combination. The resulting reference list pointer is found at the end of any successful search in the Command List component RRL.(CL.). The number of different document numbers involved in this list will be found in the component DCNT(CL.).

2.1.4 Catalog Output

The main purpose of the output module is to transmit three kinds of disk-stored data to the user. These kinds of data are document numbers obtained from the lists in the Inverted Files, catalog information extracted from the Catalog Files, and fiche addresses pulled out of the Fiche Directory. In addition, the output module has the function of initiating a request for text from the text access subsystem.

As can be seen in Fig. 2.5, the subroutine FSO controls the output process. GETLIS is first called to prepare the reference list for use. FSO steps through the reference list and displays the information that the user has requested.

FSO calls GETINT to extract each document record from the Catalog Files and bring it into memory. FSO then calls GETFLD to extract each requested catalog field from the catalog record. If GETFLD finds the field, FSO uses the console I/O package, TYPEIT, to print the following information:

1. The position number of the document on the reference list.
2. The INTREX-assigned document number
3. The number and name of the requested field.
4. The contents of the field.

The character set used in the catalog is coded to represent the special symbols that are frequently encountered there.[11] For example, the Greek

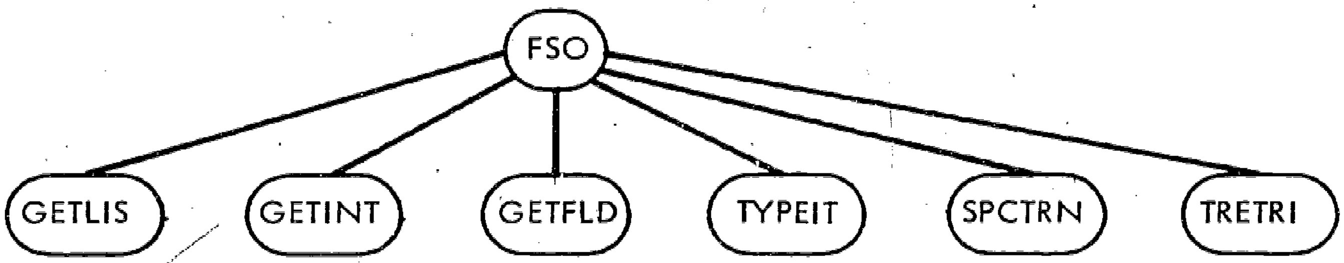


Fig. 2.5 The Output Module

letter α is represented in the INTREX catalog by the expression *alpha*. However, on a console with an extended character set, such as the INTREX console, special characters may be represented directly.[12] In this situation, FSO calls the subroutine SPCTRN instead of TYPEIT. SPCTRN translates any asterisk-coded expressions it encounters into the appropriate codes to display the symbols.

In addition to conveying catalog information to the user, FSO also responds to requests for full texts of documents. If a user asks for field 5 - the location of the document in the fiche collection - FSO calls TRETRI for this information. TRETRI looks up the address of the document in the fiche directory file and tells the user the fiche card number and the frame positions of the document. If the user asks to view the text immediately (field 90), TRETRI translates this information into a special coded sequence which activates the text access subsystem.

2.2 Examples of Command Processing

Executing a user command requires the interaction of many separate subroutines. The charts in Figures 2.6, 2.7, and 2.8 illustrate the flow of control

User command: Subject ion particle

System control

Command processing control

Receive command line
Parse command word
Look up command word
Search command control
Enter specifications into data structure
Parse specifications from command line
Look for more commands

Inverted File search control

Identify type of search
Search control

Reorder words by list size
Measure length of lists
Search for first specification word
Locate segment containing list
Search for second list
Locate segment containing list
Combine lists
Handle I/O for list merge
Check for title search request
Check for author search request

Report how many docs found
Type message to user

Go back for new command

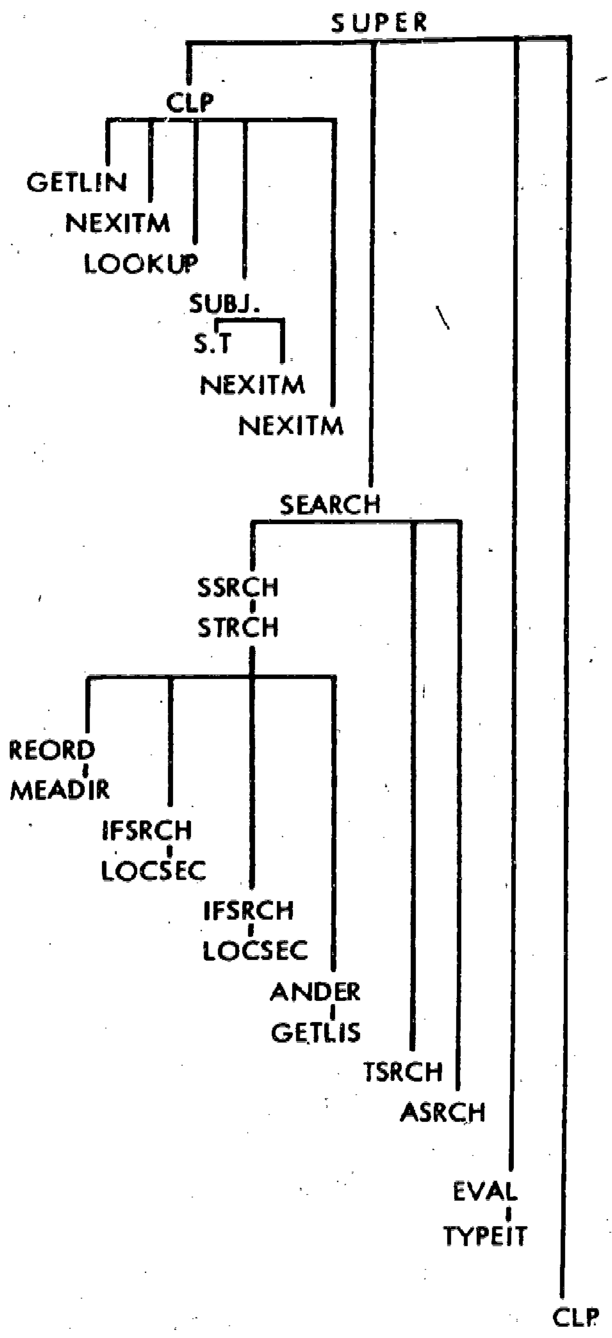


Fig. 2.6 Sequence of Subroutine Calls for the Command "s ion particle"

User command: Output title fiche

Coordinate command processing

Receive a line from the console
Type the "READY" message
Interrogate the console
Parse the command line
Find name of subroutine asked for
Interpret the "OUTPUT" command
Extract first argument
Check the argument
Extract second argument
Check second argument
Report that there are no more arguments

Produce requested output
Obtain list of documents
Extract the document
Location of a translation table
Read catalog directory
Read catalog record file segment
Extract field 24 from catalog record
Print field 24
Retrieve data on field 5
Read in fiche directory
Print fiche location

Receive a new command

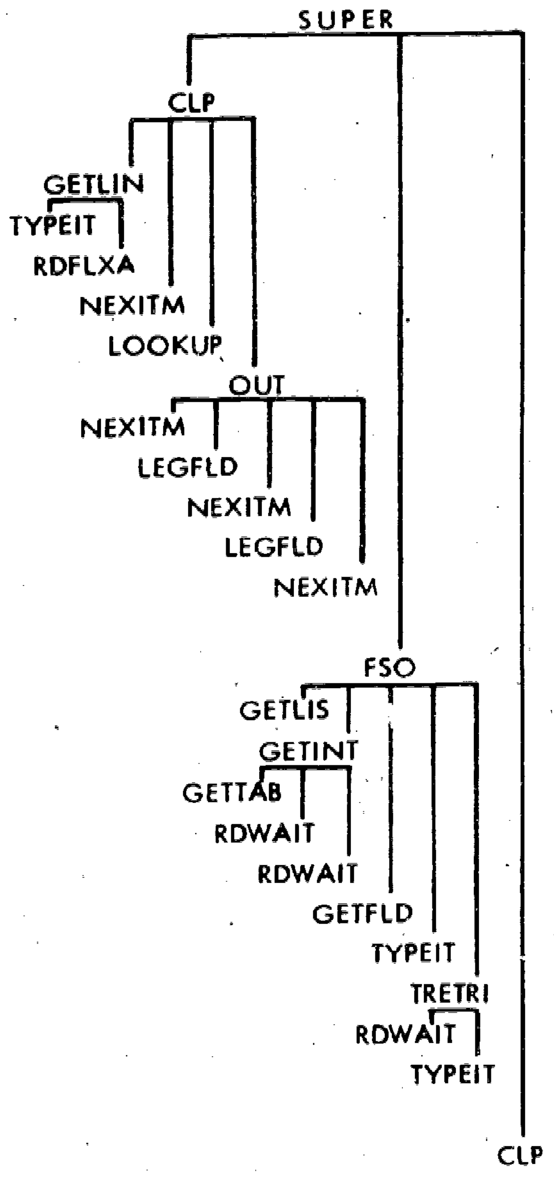


Fig. 2.7 Sequence of Subroutine Calls for the Command "output title fiche"

User command: LIST1 AND LIST2/NAME LIST3/SAVE file myfile/SAVE LIST3

Accept command line
Parse command line
Look for subroutine

Make list LIST1 active
Look up address in table
Delete current list

Combine current list and LIST2
Extract "LIST2" from command line
Look up address of LIST2
Combine LIST1 and LIST2
Initialize lists
Add combined list to table
Type results

Name current list
Extract "LIST3" from command line
See if "LIST3" is name of command
See if LIST3 already exists
Write LIST3 in NAM00x FILE

Create Save File
Identify argument "file"
Convert file name to BCD code
See if "myfile" exists as file name
Write directory section of myfile

Save LIST3 in file myfile
Find location of LIST3
Copy LIST3 into myfile

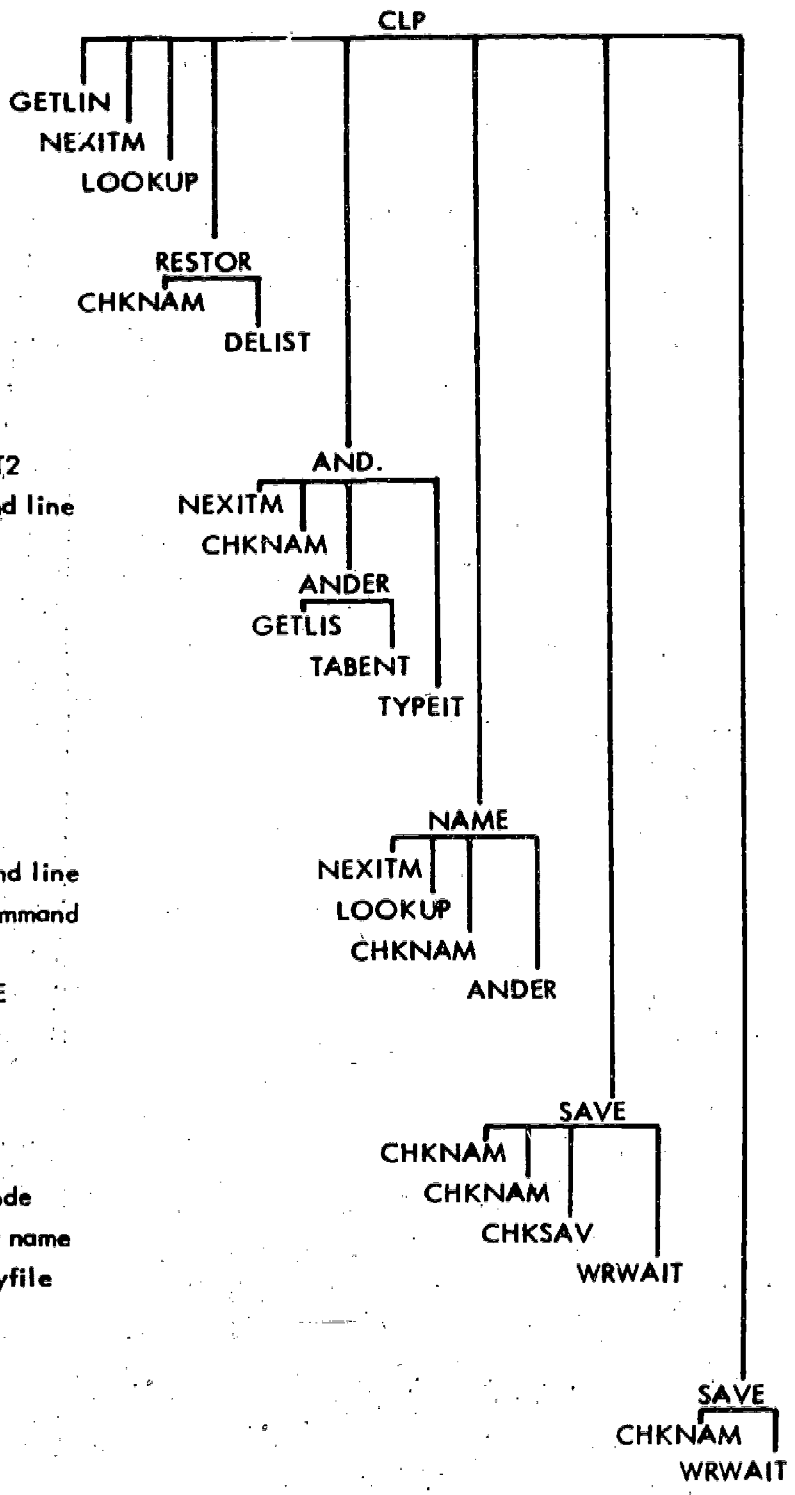


Fig. 2.8 Sequence of Subroutine Calls for a Combined Command

among the more important subroutines in response to three different commands.

Figure 2.6 shows what happens when a user issues the simple (subject) search command "subject ion particle" SUPER calls CLP, which accepts the command. SUBJ. is then called to enter the search words into the data structure. SEARCH then searches the Inverted Files for the reference lists associated with the subject search terms "ion" and "particle". Having found the lists, SEARCH calls ANDER to combine them into a single reference list. EVAL reports to the user how many documents were found. Finally, SUPER calls CLP once again to wait for further commands from the user.

Figure 2.7 shows the events in processing the command "output title fiche". SUPER calls CLP to accept the command and CLP in turn calls OUT., which enters the field specifications into the data structure. FSO transmits to the user the requested data for the documents on the currently active reference list. Control then returns to CLP.

Figure 2.8 shows the events involved in processing a string of four commands. The user enters the four commands as the combined sequence LIST1 AND LIST2/NAME LIST3/SAVE FILE MYFILE/SAVE LIST3. RESTOR is called to restore list "LIST1." AND. intersects it with LIST2. NAME assigns a name "LIST3" to the combined list. SAVE is called to create a file "MYFILE" for saved lists and SAVE is called again to transfer list "LIST3" to this file. As in the other examples, control is returned to SUPER, which returns control to CLP.

2.3 Common Structures and Variables

The approximately 150 Intrex-written routines share parameters, flags and data structures through the AED COMMON facility.[13] Variables in the source program may be declared to be located in the COMMON area assigned during compilation and thus become available to all subroutines which have the same COMMON declaration.

Intrex uses only three words of the COMMON area which serve as pointers to three essential parts of the Intrex data structure located in another part of core (see Fig. 2.9). This indirect use of COMMON allows expansion of the data structure without change in the size of the COMMON area. The three words contain the Command List pointer (CL.), the Parameter Option Table pointer (POT.), and the System State Table pointer (SST.). The corresponding areas of storage are defined as arrays in the source file OVNEW ALGOL.

The largest of these arrays, the Parameter Option Table (hereafter referred to as the POT) is partially filled during the second initialization phase of Intrex (see Section 2.1.1) with such data as system file names, line lengths, buffer addresses, and so on. Other parts of the POT are filled, as the system is executed, with parameters obtained from such sources as the arguments of the command line used to RESUME Intrex, the identification number of the console being used, and so on. A complete list of the POT parameters is given in Appendix A.

The second pointer in COMMON is the System State Table pointer (SST.). The System State Table is a group of Boolean flags, each of which indicates some state or condition of the Intrex system. These flag bits are set to TRUE(1) or reset to FALSE(0) during the execution of Intrex and help control the logical flow of processing during user/system interactions.

The third area, the Command List, is nine words long and during execution holds information and pointers to information pertaining to the user's search or output requests. These pointers are the beginning of a rather complex data structure consisting of arrays and other pointers at various levels within the data structure. This entire structure is explained briefly in Section 2.4 and in depth in Chapter III (Sections 3.2.2 and 3.2.3).

2.4 Data Base Structure

The Intrex data base consists of three main sets of files. First there are the catalog record files and a directory CATDIR to these files. The catalog record files contain catalog information for each document in the data

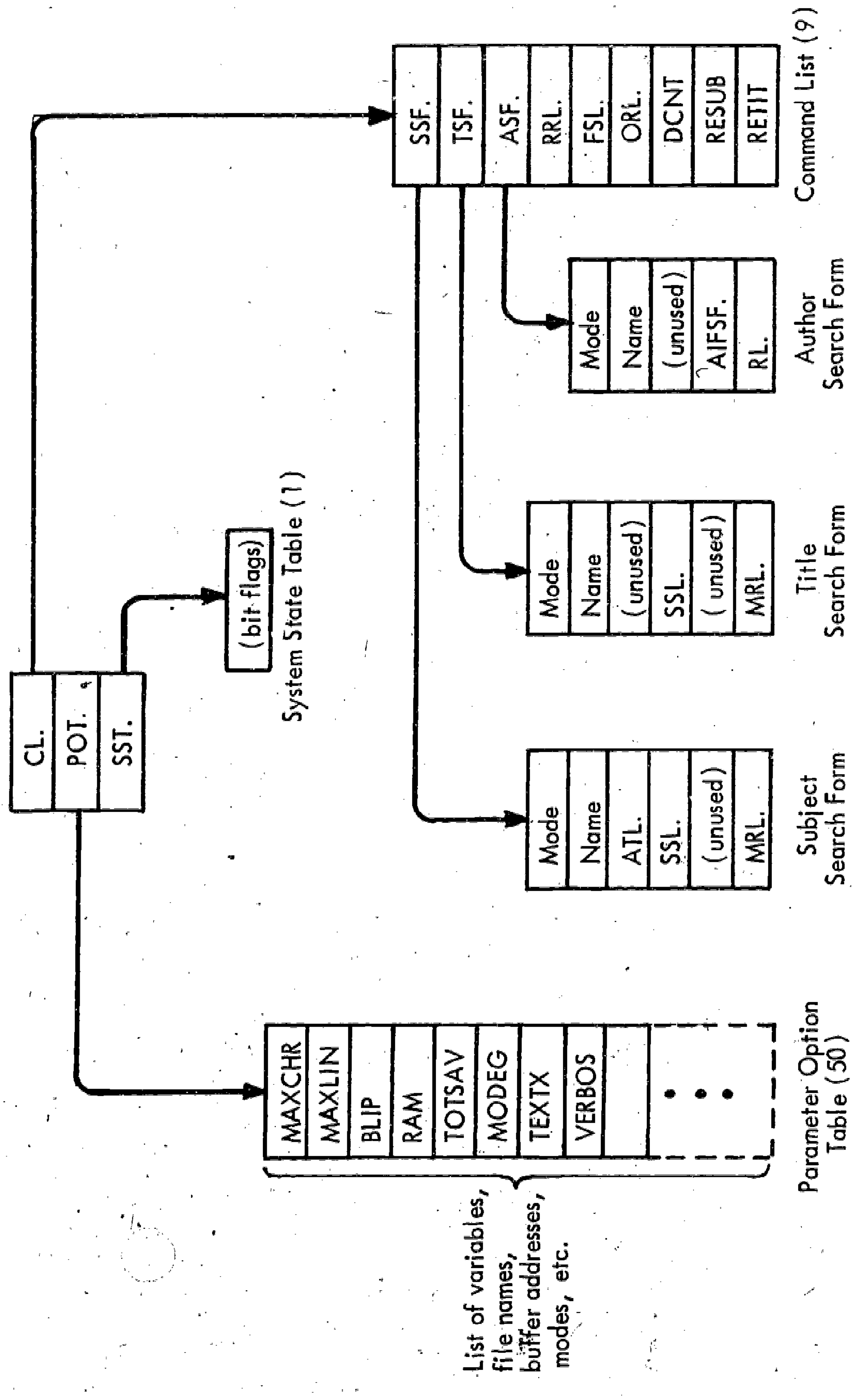


Fig. 2.9 Structure of Common Storage

base. The directory is a list of pointers to locations within the catalog record files where catalog information pertaining to a given numbered document can be found (see Fig. 2.10). The remaining two sets of files are identical to each other in structure but one is used only for subject or title searches while the other is used exclusively for author searches. Each set consists of the Inverted File and two levels of directories to the Inverted File contains an alphabetically ordered list of word stems (or author names) and a corresponding list of document numbers (pointers to CATDIR). The first directory to the Inverted File (IFDS for Subject/Title or IFDA for Author Inverted File) contains pointers to segments in the Inverted File. (This directory has been likened to the guide words on the pages of a dictionary.) The second Inverted File directory (IFTABS or IFTABA) has pointers to the beginning of each alphabetic group in the first directory and thus serves as an index to the larger directory. (This index directory can be likened to the thumb tabs in a dictionary). In order to accelerate the search process, both directories are brought into core during system initialization and maintained there. The logical relation of all of these files is depicted in Fig. 2.10. Documents are retrieved by starting with the index directory and proceeding to the catalog records corresponding to the given search term.

In addition to the files just cited which pertain to catalog retrieval, there is a directory file to the collection of microfiche which is used for full-text access. An understanding of the structure and format of all these files is essential to a complete understanding of the Intrex system. Some of the broader aspects of the files are discussed in the next three subsections. A more detailed description is contained in Appendex B and the reader is urged to consult that part of the report.

2.4.1 Catalog Files

The formatted catalog records are stored in a set of segments. The names of the segments are of the form CRnnn Name2 where nnn is a three-digit segment number and Name2 is a name common to all of the segments. The size of the segments is determined by a length threshold value M which

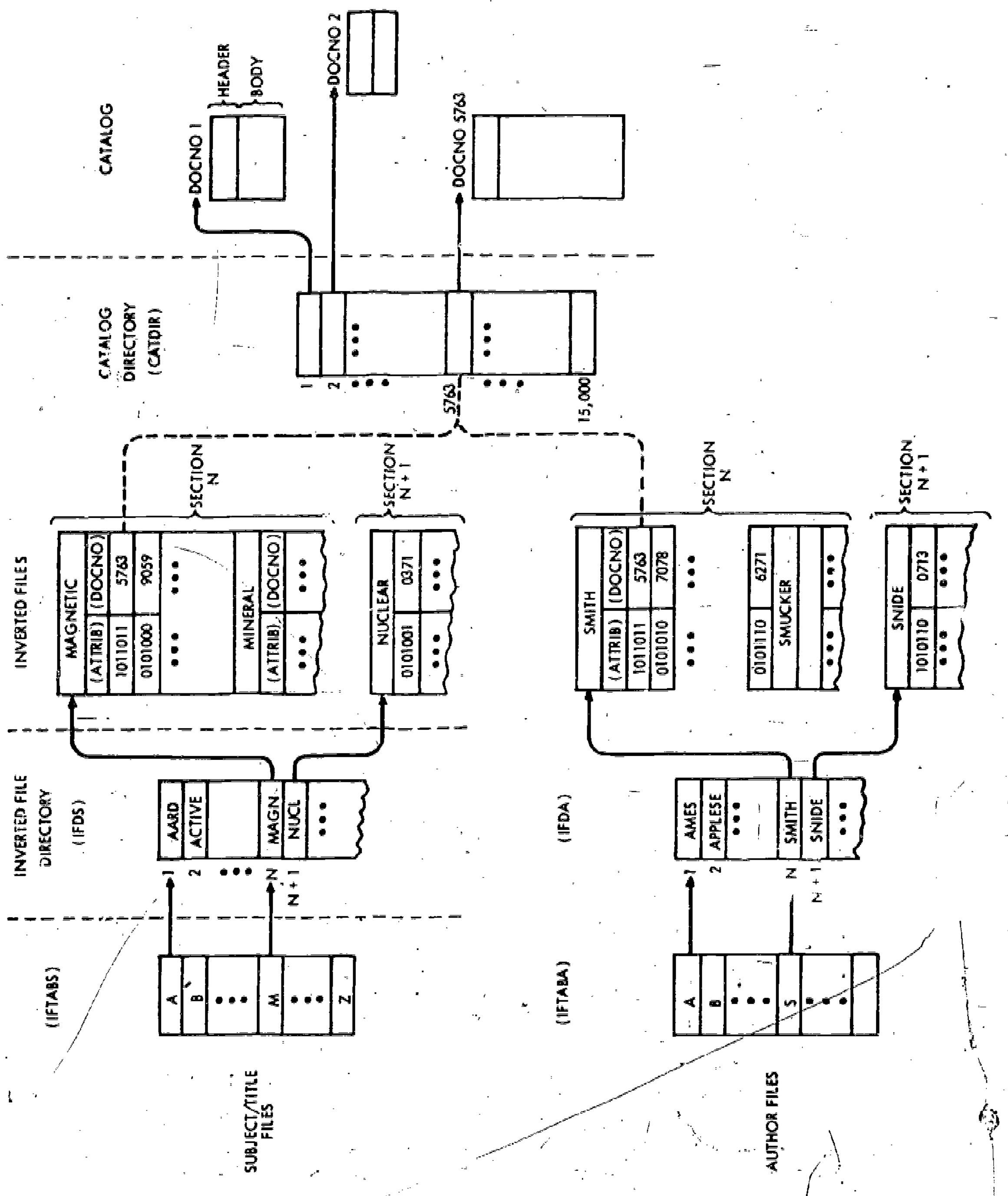


Fig. 2.10 Data Base Structure

is pre-stored in one of the ten reserved words of the catalog directory. Since formatted records are not split between segments, the length of a segment may exceed this threshold by an amount less than the length of its last record (see Fig. 2.11).

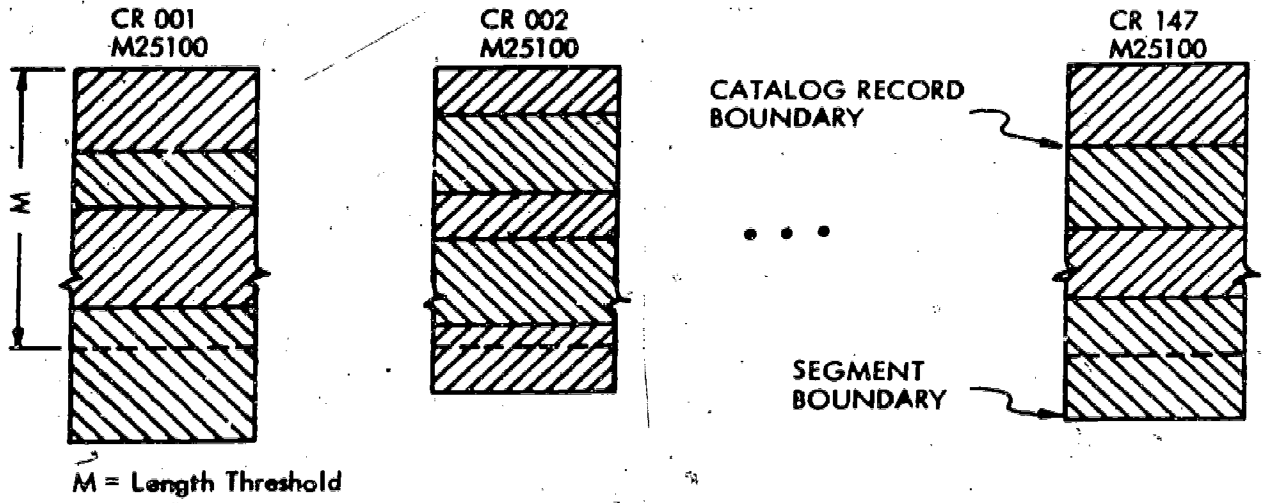


Fig. 2.11 Catalog Record Segments

Each catalog record consists of a header and a body (see Fig. 2.12). The catalog fields of fixed length (e.g. document number, on-line date, language) are encoded directly into the header. The other fields are contained in the body in the form of digram-coded ASCII strings packed one right after the other. Pointers in the header indicate the beginning of each field. Appendix B describes this format in greater detail.

The catalog record directory associated with a set of catalog record segments (i.e. CR001 Name2, CR002 Name2, ... CRnName2) is named CATDIR where Name2 is a unique second name common to each file in the set of segments and to the directory file. The CATDIR file contains an implicitly-keyed table of catalog record locations (segment number, record size, and offset in segment). In addition, the first ten words in the directory are reserved for storing statistical information about the catalog. Thus, the pointer for document no. x is located at directory position x+10. If the catalog record for document x is not yet in the catalog, its pointer will be empty (binary zero). The exact format for items in the catalog record directory (pointers and statistics) is described in Appendix B.

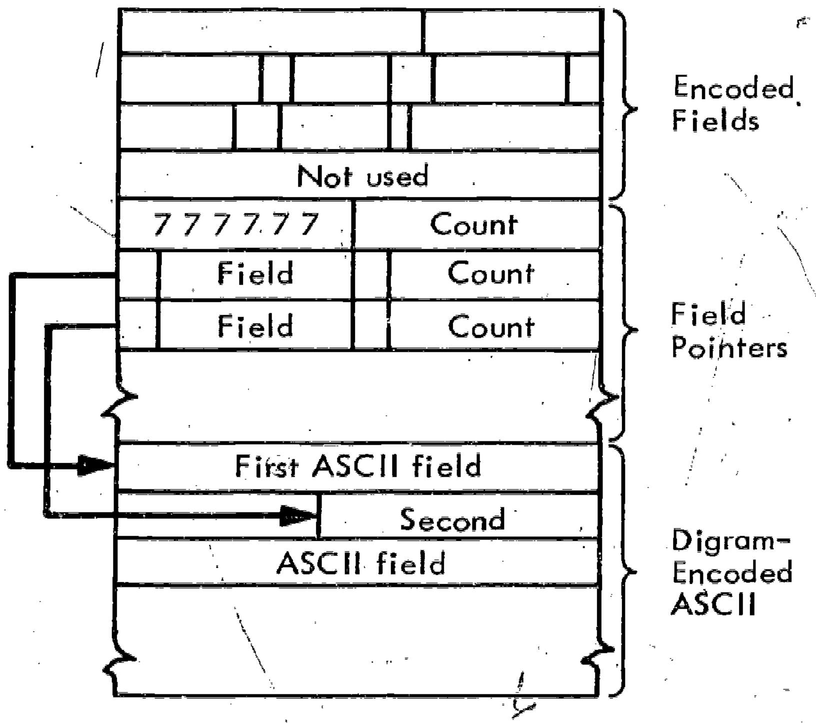


Fig. 2.12 Catalog Record Format

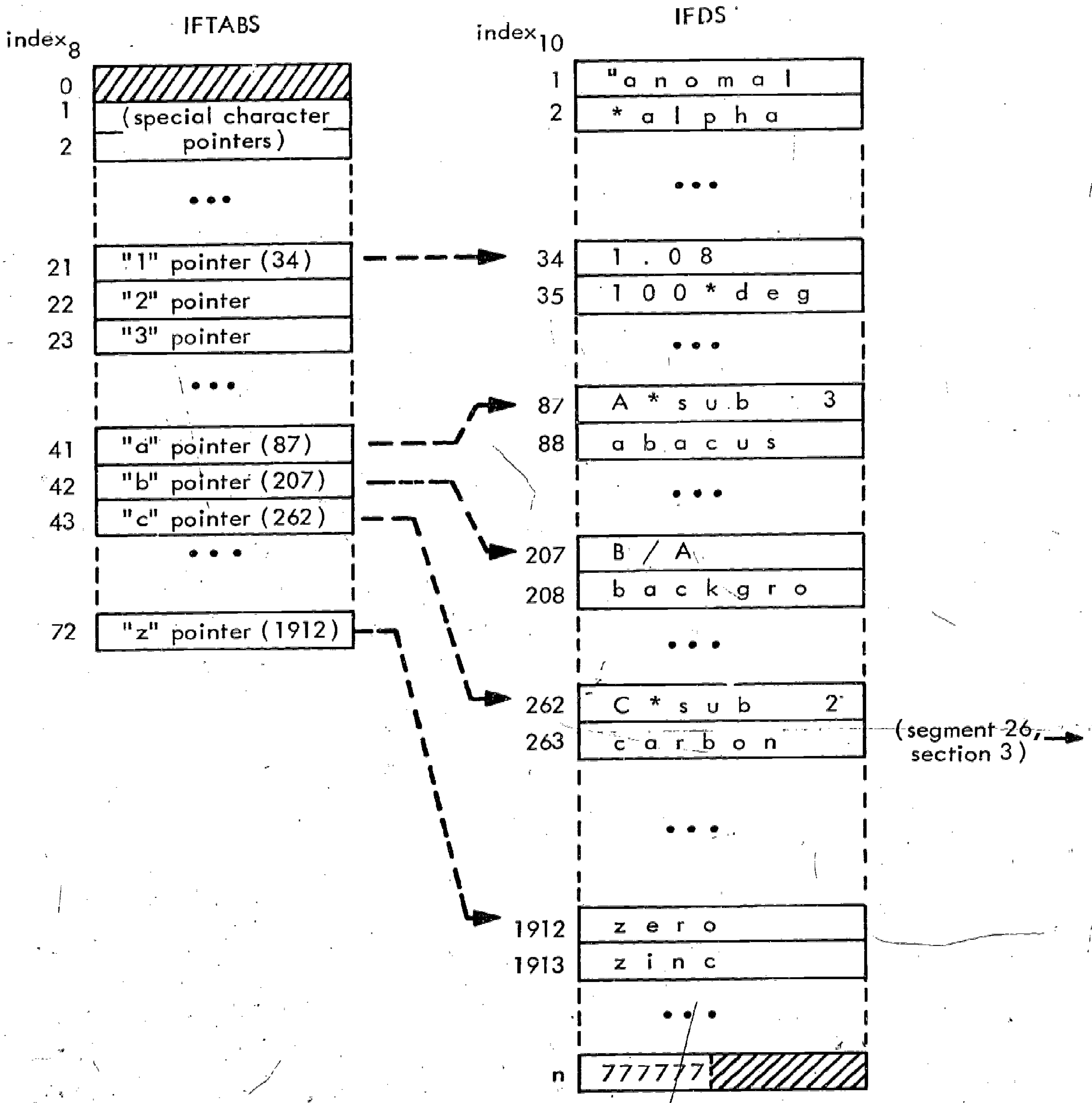
2.4.2 Inverted Files

The Inverted Files consist of a series of small disk files or segments which are numbered serially from 1 to n as part of the first name of each segment. Subject/Title segments are named SIddd date and Author segments are named AIddd date where ddd is the segment number and date is a uniform second name. Each segment is further subdivided into ten "sections". Each section may contain one or more lists of documents or only part of a list (the remainder being contained in adjacent sections). For a detailed diagram and description, see Appendix B.

The name of the first list in each section appears as an entry in the Inverted File directory (IFDS for subject/title Inverted File and IFDA for author Inverted File). Only the first seven characters of the name are used and these are represented in a 5-bit ASCII code. This allows each name to be packed into one computer word. The name is related to its file section by its position in the directory. Directory entries 1, 2, 3, etc. will contain the names of the first list in sections 1, 2, 3, etc. of segment 1. Directory entries 1, 11, 21, etc. will contain the names of the first list in segments 1, 2, 3, etc. since there are 10 sections per segment. As a further example, the name in position 54 of the directory would be the first list in Section 4, Segment 6. If no list starts in a given section, the name of the previous list (which is being continued in the given section) is placed in the corresponding directory position. In these cases, the sign bit of that directory word is a one. A fence of 6 octal 7's in the left half of the last computer word terminates the Inverted File directory.

As mentioned earlier, an index to the Inverted File directory exists to allow more direct searching of the IFDS or IFDA (which exceeds 2000 computer words in length). This index, IFTABS for subject/title files and IFTABA for author files, points to the position within IFDS or IFDA where each alphabetic group begins (see Fig. 2.13). That address is found in the IFTABS or IFTABA position which corresponds to the six-bit ASCII code of the letter.*

*The standard ASCII code requires seven bits. The six-bit code is formed by ignoring the highest order bit.



▨ Denotes Unused Portion of Computer Word

Fig. 2.13 Format of Inverted File Directories

For example, if the first name starting with the letter "b" is in position number 102 of the IFDS or IFDA directory, then the number 102 will appear in position $34 = 42_8$ (the six-bit ASCII code for b) of IFTABS or IFTABA.

The thirty-two words preceding the first alphabetic entry in the index directories are used to point to names in the Inverted File Directory beginning with numerals or other non-alphabetic characters.

2.4.3 Reference List Pointers

A list of reference words may be obtained, either through a search of the Inverted File data base or through use of the DOCUMENT command. Once obtained the list may be altered by other operations, such as RESTRICT, AND, OR, and NOT. Lists may also be saved temporarily (duration of current Intrex session) by the NAME command, or saved more permanently (on the disk) via the SAVE command. Lists may be completely stored in core (if they are small enough); they may be written into a "Dump File" if they result from a Boolean operation or an attribute screen; or they may be deposited into a "Name File", if assigned a name by the user via the NAME command. Because of this variety of list types and conditions, an elaborate three-part pointer was designed (called an "augmented pointer") to hold all the vital data about the list to which it is related. These pointers are retained in an in-core table after being constructed by the operating program segment.

Procedures which refer to reference lists use single-word pointers to one of these word augmented pointers. The most important of these is the pointer currently in the Resultant Reference List component of the Command List, RRL.(CL.), which points to the augmented pointer of the currently active list.

The format of the augmented pointers is shown and discussed in Appendix B.

2.4.4 Fiche Direct

Fiche Direct gives the locations within the text access subsystem, of the full texts of documents in the data base. The directory is ordered by document number: word n of the directory gives the location of document n. Each word contains four fields: fiche number, first frame position, last frame position and document number. If the full text of a document is not on microfiche, the fiche number field is zero and the first frame position has a special code (see Appendix B for a more detailed description).

2.5 Overlay System

An overlay system has been implemented so that the INTREX system will not be restricted by the size of core memory. The system uses a pre-assigned area of core, into which one of four segments of subroutines and data can be read as needed. The segments exist as separate disk files in loaded and initialized form: address adjustment has been performed and initialization procedures, such as INIAUT, have been executed.

Core is partitioned into three major areas: the resident area, the segment area, and the free storage area (see Fig. 2.14). The resident area contains the control subroutine, SUPER, utility subroutines such as FREE and TYPEIT, and the overlay generation and linkage mechanisms, SYSGEN and CALLIT. These programs remain in core at all times.* The segment area contains one of the four segments, each of which consists of a group of closely related procedures, together with the overlay generation and linkage mechanisms LINKUP, SENTRY, and SEGINT. The free storage area is available for use by subroutines in both the main body and the segments.

Two tables are central to the overlay mechanism. One table, which exists both as a part of SYSGEN and as the disk file, sysnam.TBLE. specifies how many segments there are in the system and in what segment or segments a particular procedure may be found. The other has four versions, one associated with the subroutine SENTRY in each of the four overlays. This table lists the entry points into the segment with which it is associated.

SYSGEN, LINKUP, SENTRY and SEGINT are used in generating the overlay system. (See also Section 2.1). SYSGEN reads sysnam .TBLE.,

*More precisely these programs remain in the core image of an Intrex user on CTSS. CTSS swaps core images in and out of physical core in the process of time-sharing.

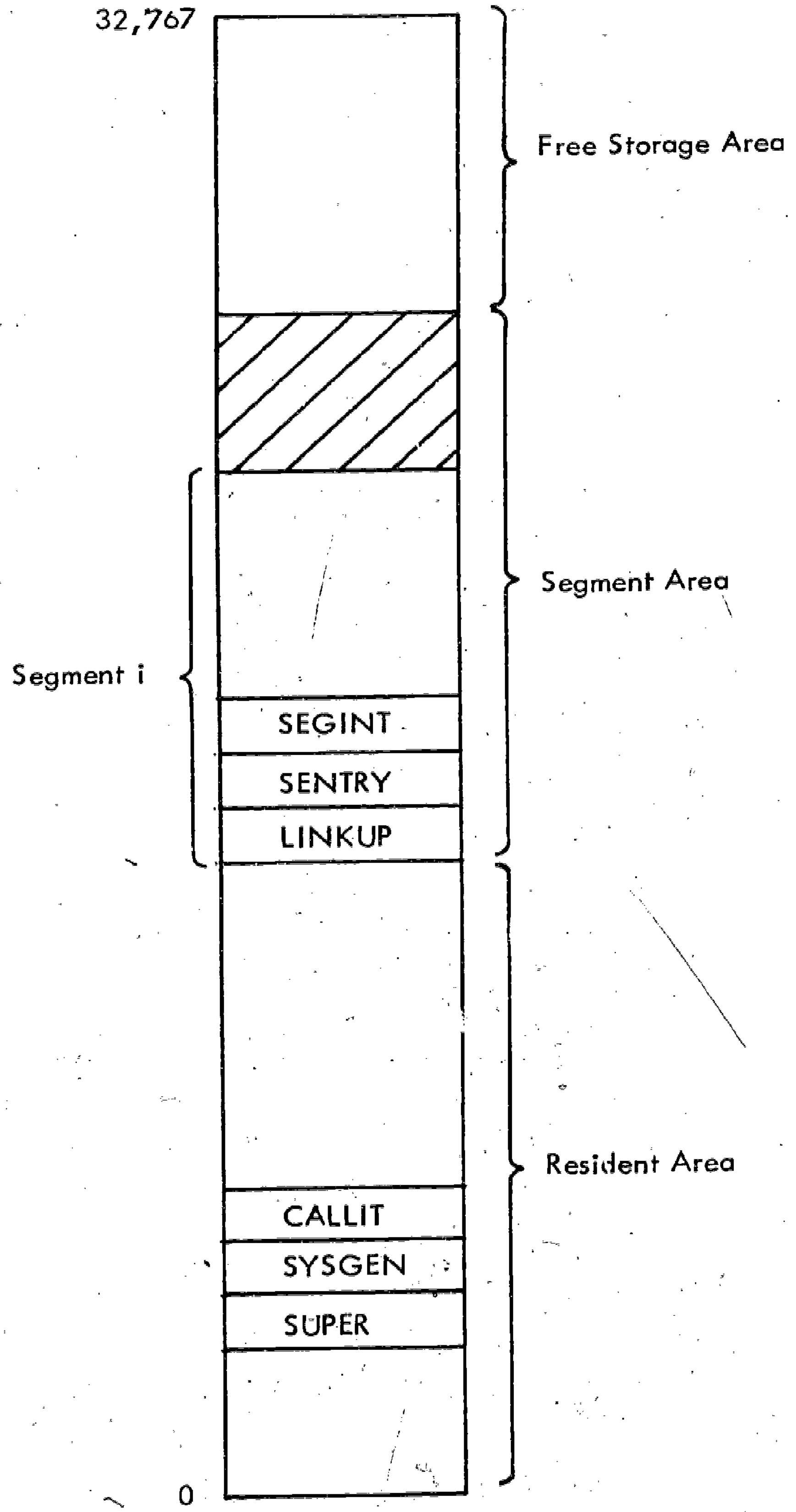


Fig. 2.14 Partitioning of Intrex Core Memory

and fills it in with information that it finds in the table contained in SENTRY. SYSGEN calls SEGINT, which in turn calls initialization procedures for subroutines within the segment. SYSGEN then writes out sysnam. TBLE. and the segment as a file sysnam SGMTnn, where nn is the number of the segment and sysnam is the name given to this version of Intrex (see Fig. 2.15). LINKUP merely provides linkage between SYSGEN and SENTRY.

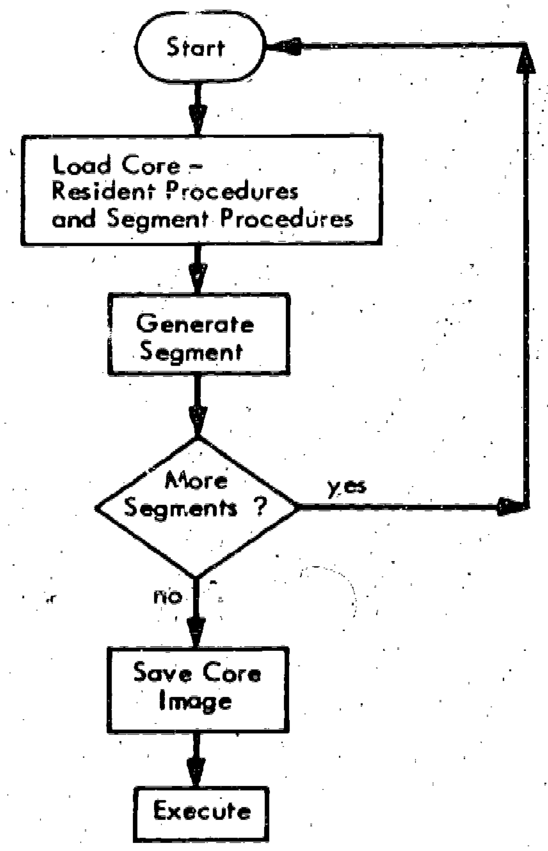


Fig. 2.15 Flow Chart of System Generation Process

CALLIT, LINKUP, and SENTRY are used during execution. A procedure in the core-resident area transfers to a procedure, PROC, in a segment by making a call of the form X = CALLIT (.bcd/PROC/, arg1, arg2), where arg1, arg2 are arguments of PROC and X is its value. CALLIT looks up PROC in the in-core copy of sysnam. TBLE. and reads in the needed segment, if it is not already in core. It then transfers to LINKUP, which transfers to SENTRY. SENTRY looks up PROC in its own table and transfers directly to it passing on arg1 and arg2.

III. SOFTWARE DETAILS

The objective of this chapter is to provide the reader with an understanding of the purpose and the operation of each subroutine or procedure of the Intrex retrieval programs. Included in this chapter is a statement of the fundamental strategies and goals of each system component. Also included is the detailed information that allows a reader to follow the program listings and understand the exact operation of each procedure of the Intrex retrieval programs.

The description of each routine o_ program has been broken down into eleven parts. Part A "Operation" gives a step-by-step account of the logical flow, explaining alternative branches as they occur. This part could be used in conjunction with the program listings by a reader who requires an in-depth understanding of the programs. The other parts, B through K, provide various other types of information pertinent to the subroutine. For example, Part K lists all files which could be created, deleted, or used by the routine in question. Part J cites the file name where the program can be found.

Some of the conventions employed in this chapter may need a few words of explanation.
In general the following conventions are observed with regard to names.

1. Names of precedures, components and specific files are expressed in all capital letters (e.g. SUPER,RRL.(CL.), SAVED DIRECT).
2. Optional (may or may not be used) arguments are represented in lower case characters with a hyphen on both sides of the word. (e.g. -mode-, -wordno-).
3. Variable file names or portions thereof which must be present are represented in lower case without hyphens. (e.g. file , SInnn, date).

The abbreviations chosen to represent the arguments of the procedure call in Part E and at the beginning of Part A are intended to be indicative of the role they play in the call but are not always the same as those used in the program listings.

All messages which can possibly originate from the procedure are listed in Part H and numbered. Parenthesized numbers in Part A (Operation) appearing after references to messages produced by the procedure refer to the position of the message in this list. Each message is enclosed in quotation marks in the list, although not in actuality, and followed by the label(s) or routine(s) which is (are) used to generate the text. Vari ble parts of the messages are represented by symbolic tags and either underlined or footnoted to indicate their meaning.

We have attempted, especially in Part A, to provide an extensive amount of cross-referencing to other procedures which relate to the one being described. Should additional cross-referencing be required, the reader may refer to the alphabetic index of procedures to find the location of the section describing the needed procedure.

In order to aid the reader in finding particular subroutine descriptions, each such description (subsection) begins on a new page. The subsection number is printed at the top of each page to facilitate locating the subsection.

3.1 System Control Logic

3.1.1 Supervision

3.1.1.1 SUPER

Purpose

To supervise Intrex logic flow

Description

SUPER is the original entry point into the retrieval system. It is also the routine to which control returns after the basic modules of the system have performed their tasks.

A. Operation: SUPER( )

1. SUPER calls routines to initialize the INTREX System.
   a. SYSGEN is the first routine called. After it does its work it calls DORMNT.
   b. When the system is resumed, SYSGEN returns control to SUPER, which calls the initialization routines INIFIX and INIVAR, and then calls DORMNT.
   c. When the system is resumed once again, SUPER calls DYNAMO.
2. SUPER asks that the user type begin and then calls SYSGEN which introduces the user to the system.
3. Depending on the settings of bits in the System State Table (SST.), SUPER will call the following routines:
   a. CLP is called to accept a command line from the user.
   b. SEARCH is called to carry out a search of the Monitor File
   c. EVAL is called to summarize the results of a search.
   d. FSO is called to print catalog information.
4. SUPER makes calls to the free storage monitor FSIZE and the timing monitor MONTIM. The values reported by these routines are written in the Monitor File by means of ASIDE.

B. Procedures Calling SUPER:

SUPER is the original entry point of the retrieval system.

C. Procedures Called by SUPER:

ASIDE	FSO	SIGNIN
CLP	INIFIX	SYSGEN
DORMNT	INIVAR	TYPEIT
DYNAMO	MONTIM	
EVAL	SEARCH	

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
STM(POT.)	System time monitor	x	
FSONX(SST.)	FSO not executed	x	x
GCE(SST.)	Go command exists	x	x
ISI(SST.)	In sign-in		x
RRLE(SST.)	Resultant reference list exists	x	
SNX(SST.)	Search not executed	x	x
IBEG (SST.)	In begin state	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Please type the word begin followed by a carriage return." (/begmes/)
2. "You may make a new search by dropping some of your search words or by chosing other search words, or you may make some other request of INTREX (see Part 1.)" (/op10.5/)
3. "You have no current active list for which to provide output. You must perform a new search or restore a saved (named) list." (/super/)

I. Length:

440_8 or 288_{10} words

J. Source:

SUPER ALGOL

K. Files Referenced:

None

3.1.2 Fixed-Parameter and Data Base Initialization

3.1.2.1 SEGINT

Purpose

To initialize an overlay segment

Description

A. Operation:

A different version of SEGINT is contained in each segment. SEGINT calls individual initialization routines for procedures in a particular overlay segment. SYSGEN calls SEGINT before writing out the segment.

B. Procedures calling SEGINT:
SYSGEN

C. Procedures called by SEGINT:
INIFLD, INIVRB, INIRNG, INIAUT, INMON2

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
10-20 words

J. Source:
nSENT ALGOL (where n is 1, 2, 3, or 4)

K. Files Referenced:
None

3.1.2.2 INITDB

Purpose

To initialize COMMON parameters

Description

A. Operation INITDB( )

INITDB is called by SYSGEN during fixed-parameter initialization. INITDB contains, as arrays, the areas used for the command list (CL), the parameter option table (POT.) and the system state table pointer (SST.) with the addresses of these areas. These three words are in COMMON storage.

INITDB sets certain POT parameters, such as the length of output lines, and sets up free storage arrays for the output request list (Section 3.2.7.2) and field search list (Section 3.2.7.3). The addresses of these arrays are stored in the appropriate component of the command list array.

B. Procedures Calling INITDB:

SYSGEN

C. Procedures Called by INITDB:

FREZ

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
ORL.(CL.)	Output Request List		x
FSL.(CL.)	Field Search List		x
STM.(POT.)	System Time Monitor		x
RTM.(POT.)	Ready Time Monitor		x
MAXCHR(POT.)	Max. chars. per output line		x
MAXLIN(POT.)	Max. lines per command		x
VERBOS(POT.)	Typeit message mode		x
RAM(POT.)	Residual author mode		x
DFSN1(POT.)	Short messg. file name 1		x
DFLN1(POT.)	Long messg. file name 1		x
MAXCIN(POT.)	Max. chars. per input line		x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

54_8 or 44_{10} words

J. Source:

OVNEW ALGOL

K. Files Referenced:

None

3.1.2.3 INIFIX

Purpose

To initialize fixed parameters

Description

INIFIX, like SYSGEN, is called by SUPER to carry out the fixed parameter phase of the initialization of the retrieval system. INIFIX generates relatively stable parameters for the system, such as character strings and table entries.

A. Operation

INIFIX calls for a large block of free storage and divides it among common buffers 1 through 6 in 432 word blocks. Common buffer 0 is assigned to the top of core. INIT2 is called next. INIT2 calls the following initialization routines:

INIEVL,	which initializes EVAL	(Section 3.2.6.1),
INIOUT,	which initializes INOUT	(Section 3.2.7.1),
INIS.T,	which initializes S.T	(Section 2.2.1).

INIFIX then calls:

PREP,	which reads in COMMAND TABLE	(Section 3.1.2.5),
TABLE,	which reads in FIELDS TABLE	(Section 3.1.2.6),
MONINT,	which initializes MONTIM	(Section 3.1.5.1),
INICON,	which initializes CLP	(Section 3.2.1.1).

In the course of calling these routines, INIFIX prints out the memory bound five times, as a way of monitoring the utilization of free storage. INIFIX calls FRALG to give its coding space over to free storage before returning to SUPER.

B. Procedures Calling INIFIX:

SUPER

C. Procedures Called by INIFIX:

FRET, INICON, INIT2, MONINT, PREP, SIZE, TABLE, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBFO(POT.)	Common buffer 0		x
COMBF1(POT.)	Common buffer 1		x
COMBF2(POT.)	Common buffer 2		x
COMBF3(POT.)	Common buffer 3		x
COMBF4(POT.)	Common buffer 4		x
COMBF5(POT.)	Common buffer 5		x
COMBF6(POT.)	Common buffer 6		x
COMTB.(POT.)	Command Table		x

E. Arguments·

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Top of Intrex is X" (LOCMES)
2. "After Init2 X" (LOCMES)
3. "After Prep X" (LOCMES)
4. "After Table X" (LOCMES)
5. "After Inifix X" (LOCMES)

I. Length:

153_8 or 107_{10} words

J. Source:

INITLY ALGOL

K. Files Referenced:

None

3.1.2.4 INIT2

Purpose

To call initializing procedures

Description

A. Operation INIT2( )

"INIT2 is called by INIFIX. INIT2 is used as a convenient place to put calls to initialization routines of procedures which are in the core-resident section of the system. INIT2 currently calls:

INIEVL,	which initializes EVAL	(Section 3.2.6.1),
INIOUT,	which initializes INOUT	(Section 3.2.7.1),
INIS.T,	which initializes S.T.	(Section 3.2.2.1).

B. Procedures Calling INIT2:

INIFIX

C. Procedures Called by INIT2:

INIEVL, INIOUT, INIS.T

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

6 words

J. Source:

OVNEW ALGOL

K. Files Referenced:

None

3.1.2.5 PREP

Purpose

To construct command table

Description

PREP constructs the INTREX command table. This table associates the INTREX commands with their interpretive subroutines. The table consists of two word entries: the first word contains the first four letters of an INTREX command in ASCII and the second word contains the associated subroutine name, written in 6-bit BCD, LOOKUP uses VSRCH to find the second element of an entry by searching the table for a match on the first element.

A. Operation: Ptr = PREP( )

PREP reads theline-mark file COMMAND TABLE into core memory. An area equal to 2/3 the length of this file is allocated for the table that PREP is to construct. Each line of the file contains the first four letters of a command, followed by a tab, followed by a procedure name. Using the tab as a delimiter, PREP extracts the four letter string, converts it to ASCII and stores it in word n of the table. The 6-bit procedure name is stored in word n + 1. When the table has been filled, PREP returns a pointer to the table, where the decrement contains the number of 2-word elements, rather than the number of words.

B. Procedures Calling PREP:

INIFIX (via CALLIT)

C. Procedures Called by PREP:

ASCITC	DORMNT	FRET	TYPEIT
BFCLOS	FILCNT	GET6	
BFOPEN	FRALG	LOCMES	
BFREAD	FREE	PUT6	

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMTB(POT.)	Command table Ptr.		x

E. Arguments

None

F. Values:

Ptr = Bits 3-18: number of 2-word elements
Bits 21-35: location of table

G. Error Codes:

None

H. Messages:

1. "File error in PREP" (LOCMES)
2. "PREP command over 4 characters . . . Fatal error." (LOCMES)

I. Length:

400_8 or 256_{10} words

J. Source:

PREP ALGOL

K. Files Referenced:

COMMAND TABLE

3.1.2.6 TABLE

Purpose

To generate field name and range name tables

Description

TABLE is used to create four tables:

(1) A table of pointers to the ASCII names of all the catalog fields. Each word of the table corresponds to a field number. If there is no field n, the nth word of the table is zero. This table is used in two ways: to determine if a field specified by a user is a legitimate field, and to associate the field number of a field with the name of the field.

(2) A table of pointers to the ASCII names of the RANGE attribute. This table is completely analogous to the field names table.

(3) A simple list of all the field numbers, one number per word. This list is used by FSO when the user requests "output all".

(4) A simple list of the field numbers of the standard fields. This is used by FSO when the user requests "o" or "o standard".

A. Operation: TABLE( )

TABLE creates these four tables by using the data it finds in the file FIELDS TABLE. This is an ASCII file and has on each line the catalog field number followed by a tab and the field name. A plus (+) sign following a field name signifies that this field is to be used in the standard field list. The RANGE attributes follow the catalog fields and have the same format.

INIFIX calls TABLE via CALLIT. TABLE allocates a 91-word array for the the field names table and a 5-word array for the range attributes. TABLE reads FIELDS TABLE into memory, counts the total number of fields and the subset of standard fields and allocates memory for the complete fields list and the standard fields list. TABLE proceeds to copy the ASCII field names into free storage, inserting a pointer in the field names table and adding the binary equivalent of the field number to the field list. After TABLE has gone through the same procedure for the range names, TABLE calls RPRIME. RPRIME is an initialization procedure for the subroutines RNGNAM, LEGFLD, FLDNAM, FIELDS and STANDL.

These subroutines supply system routines with information about the tables, information which they receive from TABLE via RPRIME.

B. Procedures Calling TABLE:

INIFIX

C. Procedures Called by TABLE:

ASCINT	FILCNT	LOCMES
BFCLOS	FRALG	NEXITM
BFOPEN	FREE	PUT
BFREAD	FREZ	RPRIME
COPY	INC	TYPEIT
DORMNT	INC 1	

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBF1(POT.)	Common buffer 1	x	
COMBF2(POT.)	Common buffer 2	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1 "Error in TABLE" (preset)

I. Length:

1006_8 or 518_{10} words

J. Source:

TABLE1 ALGOL

K. Files Referenced:

FIELDS TABLE

3.1.2.7 RPRIME

Purpose

To initialize pointers to tables

Description

A. Operation: RANGE (Range, Range.1, Name, All.field, All.len, Stand, Stand.len)

RPRIME initializes RNGNAM, LEGFLD, FLDNAM, FIELDS, and STANDL by transferring parameters to these routines from TABLE.

B. Procedures Calling RPRIME:

TABLE

C. Procedures Called by RPRIME:

None

D. COMM nces:

N

E. Arguments:

RANGE:	location of array of pointers to RANGE names.
RANGE.L:	length of RANGE array.
NAME:	location of array of pointers to names of catalog fields.
ALL.FIELD:	location of list of all of the fields.
ALL.LEN:	length of list of fields.
STAND:	location of list of standard fields.
STAND.LEN:	length of standard field list

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

30_8 or 24_{10} words

J. Source:

TABLE2 ALGOL

K. Files Referenced:

None

3.1.2.8 RNGNAM

Purpose

To return pointer to range names pointers

Description

A. Operation: Ptr = RNGNAM( )

RNGNAM returns a pointer to a list of the pointers to the ASCII names of the RANGE attributes. The document of the pointer contains the length of the list in words.

B. Procedures Calling RNGNAM:

RANGE, EVAL

C. Procedures called by

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

Ptr = word pointer to list of ASCII pointers

G. Error Codes:

None

H. Messages:

None

I. Length:

6 words

J. Source:

TABLE2 ALGOL

K. Files Referenced:

None

3.1.2.9 LEGFLD

Purpose

To check for legal field number

Description

A. Operation: Boolean = LEGFLD(Fldno)

LEGFLD uses the field number "field" to index the array of pointers to the names of catalog fields. If "field" is greater than zero and less than 91 and the word indexed by "field" contains a pointer, LEGFLD returns a value of true.

B. Procedures Calling LEGFLD:
IN., OUT.

C. Procedures Called by LEGFLD:
None

D. COMMON References:
None

E. Arguments:
Fldno: field number (binary)

F. Values:
Boolean = true or false

G. Error Codes:
None

H. Messages:
None

I. Length:
10_8 or 8_{10} words

J. Source:
TABLE2 ALGOL

K. Files Referenced:
None

3.1.2.10 FLDNAM

Purpose

To return pointer to field names pointers

Description

A. Operation: Ptr = FLDNAM( )

FLDNAM returns a word pointer to an array of pointers to the ASCII names of all of the fields. The pointer to the field name of field n is in word n of the array. If there is no field n, word n is zero.

B. Procedures Calling FLDNAM:

EVAL, FSO

C. Procedures Called by FLDNAM:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

Ptr = word pointer to array of pointers

G. Error Codes:

None

H. Messages:

None

I. Length:

6 words

J. Source:

TABLE2 ALGOL

K. Files Referenced:

None

3.1.2.11 STANDL

Purpose

To return pointer to list of field numbers

Description

A. Operation Ptr = STANDL( )

STANDL returns a word pointer to the list of standard fields. This list is 4 words long and contains the numbers 24, 21, 23, and 47.

B. Procedures Calling STANDL

FSO

C. Procedures Called by STANDL

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

Ptr = word pointer to array containing standard fields

G. Error Codes:

None

H. Messages:

None

I. Length:

10_8 or 8_{10} words

J. Source:

TABLE2 ALGOL

K. Files Referenced:

None

3.1.3.12 FIELDS

Purpose

To return a pointer to a list of field numbers

Description

A. Operation: Ptr = FIELDS( )

FIELDS returns a pointer to a 50-word array containing all of the field numbers. The length of the array is in the decrement of the pointer. Each word of the array contains one field number in binary.

B. Procedures Calling FIELDS:
FSO

C. Procedures Called by F :LDS:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
Ptr = word pointer

G. Error Codes:
None

H. Messages:
None

I. Length:
12_8 or 10_{10} words

J. Source:
TABLE2 ALGOL

K. Files Referenced:
None

3.1.2.13 INIVAR

Purpose

To link the retrieval system to a particular data base.

Description

A. Operation: INIVAR( )

SUPER calls INIVAR after INIFIX returns control to it. SUPER also calls INIVAR if DYNAMO indicates to SUPER that the user has typed 'RENEW' in his command line. INIVAR asks the user to type the last names of the catalog files* and the inverted files**. These names are stored in CATS2(POT.) and IFS2(POT.) respectively. Next, INIVAR calls IFSINT, which reads the inverted file directories and calls INIEND, which will initialize the ending table that is used by the stemming mechanism. INIVAR closes all files before returning to SUPER.

B. Procedures Calling INIVAR:

SUPER

C. Procedures Called by INIVAR

COMARG	RDFLXA	RET
TYPEM	IFSINT	CLOSE
LOCMES	FREE	

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
SYSNAM(POT.)	System name		x
CATS2(POT.)	Last name of catalog files		x
IFS2(POT.)	Last name of inverted files		x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Please enter last name of catalog files", (LOCMES)
2. "Please enter last name of inverted files", LOCMES)

Footnotes:

* 1. "INTREX"

** 2. the date of creation of the files: mmddyy.

I. Length:

75_8 or 61_{10}

J. Source:

INITLY ALGOL

K. File References:

None

3.1.2.14 IFSINT

Purpose

To initialize search module

Description

The procedure IFSINT must be called once prior to the first use of the Inverted File lookup routine IFSRCH. IFSINT is called from INIVAR during the data base initialization phase and handles the reading of the Inverted File directories and "ending table". Although IFSINT is closely related to IFSRCH, it is compiled in a separate source file and communicates with IFSRCH by means of the sub-procedure IFSET.

A. Operation: IFSINT( )

IFSINT first calls the procedure INIEND (see next section) which reads a file of common word endings into core and constructs a table of pointers to ending subsets (grouped by length) for use by the stemming procedure STEM during the processing of user search requests. INIEND returns to IFSINT with the core address of the ending pointer table. This address is then passed to the module containing STEM by calling the procedure GIVTAB (Section 3.1.2.16) with the address in the argument.

Next, the last name of the current Inverted File is extracted from IFS2(POT.) (where it was placed earlier by INIVAR). The size (number of computer words) of both the subject file directory and the author file directory is obtained and the directories are read into these two areas via RDWAIT.

If no directories with this last name are found on the disk, an error message (1) is typed and CHNCOM is called, terminating the initialization process.

B. Procedures Calling IFSINT

INIVAR

C. Procedures Called by IFSINT

INIEND, GIVTAB, FILCNT, FREE, FRET, OPEN, RDWAIT, CLOSE, TYPEIT, LOCMES, CHNCOM

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
IFS2(POT.)	Inverted File Name-Two	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "No subject inverted file.", (LOCMES)

I. Length:

504_8 or 324_{10} words

J. Source:

IFSINT ALGOL

K. File Reference:

IFTABA	-	date	-
IFTABS	-	date	-
IFDA	-	date	-
IFDS	-	date	-

3.1.2.15 INIEND, REND(E.F) ENDTAB

Purpose

To initialize the Ending Table

Description

A list of word-endings exists in a disk file name ENDING TEST2 which must be read and formatted into an ending-table by Intrex during initialization.

A. Operation: TAB = INIEND( ), REND(E.F), ENDTAB(S,T)

INIEND is called by IFSINT during "fixed" initialization and by GETEND when EVAL is reconstructing the user's subject or title search term (see description of GETEND in Section 3.2.6.3). INIEND's main function is to return the address of the ending pointer table to the calling program. If that address has already been inserted into the local variable named TAB, then INIEND merely returns with this address as its value.

If TAB is empty, INIEND calls a sub-procedure named REND which will read the ending file from the disk. REND accepts two arguments--the names of the ending file (currently ENDING TEST 2) - and calls FSTATE on this file to get its length. An array of free-storage of this length is obtained by calling FREZ. Two I/O buffers are also so obtained and the file is opened for buffered reading. An error in opening the file will result in an error message (1) and an abort via CHNCOM.

Then,the contents of the file are read into the allocated area, the file is closed and the I/O buffers are returned to free-storage. An error while reading the file will also produce an error message (2) and a call to CHNCOM.

At the top of the ending list is a group of "relative-location pointers" to the various ending-length subsets. This is converted to a table of absolute-location pointers by calling a subprocedure of REND named ENDTAB. This procedure accepts two arguments containing the address of the core-stored ending/pointer list and the address of an array set aside to hold the new pointers. ENDTAB then extracts each relative pointer, adds the address of the top of the ending list to the relative address, and stores the modified pointer into the corresponding slot of the new pointer array.

When all pointers have been constructed and stored, the address of this table is passed back to REND, from there to INIEND, and from INIEND to the original calling routine.

B. Procedures Calling INIEND:

IFSINT, GETEND

C. Procedures Called by INIEND:

(through REND) FILCNT, FREZ, BFOPEN, FBREAD, BFCLOS, FRET, TYPEIT, LOCMES, CHNCOM

D. COMMON References:

None

E. Arguments:

INIEND - None; REND - E.F: ending file; ENDTAB - S: endings address, T: table address

F. Values:

TAB = address of ending pointer table

G. Error Codes:

None

H. Messages:

1. "bf routine error", (LOCMES)
2. "error in reading endings" (LOCMES)

I. Length:

253_8 or 171_{10}

J. Source:

STEM2A ALGOL

K. File References:

ENDING TEST2

3.1.2.16 GIVTAB

Purpose

To pass Ending Table address

Description

A. Operation: GIVTAB(TABVAL)

GIVTAB's only task is to accept an argument containing the ending table address as passed from IFSINT and deposit it into the local variable so that it is available to STEM.

B. Procedures Calling GIVTAB:
IFSINT

C. Procedure Called by GIVTAB:
None

D. COMMON References:
None

E. Arguments:
TABVAL address pointer

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
10_8 or 8_{10}

J. Source:
STEMIA ALGOL

K. File References:
None

3.1.3 Session Initialization

3.1.3.1 DYNAMO

Purpose

To initialize system for a user session

Description

DYNAMO is the first procedure that SUPER calls when the INTREX retrieval system is invoked in a user session. DYNAMO processes any arguments that the user has typed on the command line and performs the kind of initialization (such as the opening of files) that must be done at the beginning of a retrieval session.

A. Operation: Code = DYNAMO ( )

DYNAMO looks for the following arguments in the CTSS command line.

(a) Sysnam: If the name of the RESUMED system is not "INTREX" or "INTNEW" or "INXTST" then TESTIT (SST.) is set to true. In the TESTIT mode, no monitor file is written.

(b) "SHORT" "LONG": VERBOS(POT.) is set to either 0 or 1. The default value is 1 (long mode).

(c) "BEG": If this argument is found, the system will not ask the user to type "BEGIN".

(d) "SKIP": This will cause the entire signin procedure to be skipped.

(e) "HOLD" password: The system will not return to CTSS level unless the password is typed along with the quit command.

(f) "RENEW": Dynamo returns to SUPER with a value of -1,SUPER responds to this value by calling INIVAR to re-initialize the data base.

After processing the command line, DYNAMO proceeds to set up the system for execution by calling the following initialization routines:

(a)	OPFILE:	The overlay segments are opened.
(b)	INIMON:	MONTOR is initialized
(d)	KILFAP:	if TESTIT(SST.) is false, FAPDBG is returned to free storage.
(e)	INITYP:	The typeit message directory is read into core
(f)	ININT:	The interrupt mechanism is initialized
(g)	RSCLCK:	The B-core CPU clock is initialized

(h) SETWRD: The password is stored in A-core.
(i) LDOPT: The A-core supervisor is told under what circumstances the INTREX subsystem should be invoked.
(j) SETSYS: The A-core supervisor is told the name of the INTREX subsystem.
(k) INXCON: The type of the terminal is determined.

DYNAMO returns to SUPER with a value of 0. SUPER goes on to log the user in unless in SKIP mode.

B. Procedures Calling DYNAMO:

SUPER

C. Procedures Called by DYNAMO:

CLOSE	INIRES	OPEN
COMARG		OPFILE
INIMON	INXCON	RSCLCK
INIDSK	KILFAP	SETSYS
ININT	LDOPT	SETWRD

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
ESCODE(POT.)	Escape code		x
CATS2(POT.)	Name 2 of catalog	x	
BYTEC(POT.)	Byte count		x
SYSNAM(POT.)	System name		x
MAXCHR(POT.)	Line length for output		x
DFSN1(POT.)	Short message file	x	
DFLN1(POT.)	Long message file	x	
PFN1(POT.)	Password file	x	
VERBOS(POT.)	Long/short mode		x
CLAMP(SST.)	Hold mode		x
IBEG(SST.)	In begin stage		x
SPEC1(SST.)	Special switch #1		x
SKIPS(SST.)	Skip signin		x
TESTIT(SST.)	Testing mode	x	x
CAT11(SST.)	Catalog off-line		x

E. Arguments:

None

F. Values:

Code = 0 normal return
- 1 renew option requested

G. Error Codes:

None

H. Messages:

None

I. Length:

561_8 or 369_{10} words

J. Source:

INITLY ALGOL

K. Files Referenced:

CATDIR	INTREX
(Sysnam)	SGMT 01
(Sysnam)	SGMT 02
(Sysnam)	SGMT 03
(Sysnam)	SGMT 04

3.1.3.2 INXCON

Purpose

To identify type of console

Description

A. Operation: Bool = INXCON( )

INXCON sets the value of BLIP(POT.) and returns a value of true or false depending on the type of console the system is interacting with. This determination is made by noting the page length supplied by the procedure GETP (Section 3.6.1.5). The values are set according to the following rules:

Type of Console	Value of INXCON	Value of blip(pot.)
2741	false	space-backspace
ARDS	false	0
INTREX Console	true	0

B. Procedure Calling INXCON:

DYNAMO

C. Procedures Called By INXCON:

GETP, WHOAMI

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
BLIP(POT.)	blip characters		x

E. Arguments:

None

F. Values:

Bool = TRUE if INTREX console, FALSE otherwise

G. Error Codes:

None

H. Messages:

None

I. Length:

45_8 or 37_{10} words

J. Source:

INITLY ALGOL

K. Files Referenced:

None

3.1.3.3 OPFILE, CLFILE

Purpose

To open and close overlay segments

Description

A. Operation: OPFILE( ), CLFILE( )

These routines open the overlay segments and close them. OPFILE calls SETWRD with an argument of "STOP" so that an I/O error on opening a segment will not put the system into a loop. After all 4 segments have been successfully opened, OPFILE calls SETWRD with an argument of ESCODE(POT.).

B. Procedures Calling OPFILE and CLFILE:

DYNAMO, GETLIN

C. Procedures Called by OPFILE and CLFILE:

OPEN, CLOSE, SETWRD

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
ESCODE(POT.)	Escape Code	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length 113_8 or 75_{10} words

J. Source:

INIT2 ALGOL

K. Files Referenced:

(Sysnam)	SGMT 01
(Sysnam)	SGMT 02
(Sysnam)	SGMT 03
(Sysnam)	SGMT 04

3.1.3.4 KILFAP

Purpose

To return FAPDBG to free storage

Description

A. Operation KILFAP( )

KILFAP returns FAPDBG to free storage. It finds the beginning of FAPDBG by means of a transfer vector and returns to free storage a block of 12520_8 words beginning at this point. This routine is only used in versions of INTREX which actually have FAPBG. Those versions which do not have FAPDBG use a dummy version of KILFAP.

B. Procedures Calling KILFAP:
DYNAMO

C. Procedures Called by KILFAP:
FRET

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
14_8 or 12_{10} words

J. Source:
SYSGEN FAP

K. Files Referenced:
None

3.1.4 Logging

3.1.4.1 GO

Purpose

To start retrieval session

Description

A. Operation: Code = GO ( )

CLP transfers to GO when it detects the command "BEGIN" in the user's command line. If the user is in the begin stage of INTREX (i.e., if IBEG(SST.) is true), GO will set IBEG(SST.) to false, thereby letting the user out of the begin stage. If the user is not in the begin stage, but is operating in HOLD mode, (i.e. CLAMP(SST.) is true), GO will call QUIT to end the session and force the system to recycle. If the user is neither in the begin stage nor in HOLD mode, GO will return with a value of -1.

B. Procedures Calling GO:

CLP (via CALLIT)

C. Procedures Called by GO:

QUIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
IBEG(SST.)	In begin stage	x	x
CLAMP(SST.)	In HOLD mode	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

Code = -1: BEGIN command inappropriate

H. Messages:

None

I. Length:

37_8 or 31_{10} words

J. Source:

VERBOS ALGOL

K. Files Referenced:

None

3.1.4.2 SIGNIN

Purpose

To log in an INTREX user

Description

A. Operation: SIGNIN( )

SUPER calls SIGNIN to welcome the user to INTREX. If the argument SKIP had been given when INTREX was resumed, SIGNIN will bounce back to SUPER without doing anything. Otherwise, SIGNIN will print a message on the console asking the user to log in and will then call GETLIN to receive the user's response. If the user types something other than LOG, LOGIN, or QUIT, SIGNIN will repeat its request. If the user types QUIT, SIGNIN will call the subroutine QUIT. If the user correctly types LOG or LOGIN, but did not follow this word with his name, SIGNIN will complain and then repeat its request. When SIGNIN receives the sequence LOG(IN)-space-character.string, it will capitalize the first letter of the character string and use it as the user's name in its final message. SIGNIN calls FRALG (Section 3.4.1.10) to give up the SIGNIN area to free storage before returning to SUPER.

B. Procedures Calling SIGNIN:

SUPER(via CALLIT)

C. Procedures Called by SIGNIN:

.CTSIT6, FRALG, GET, NEXITM, PUT, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
ISI(SST.)	In sign-in		x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Intrex could not understand your log statement" ($BEGER1$)

2. "Please log in by typing the word LOG followed by a space and your name and address as in the following example, log smith, rj; mit13-5251; ext7234.
   Note that your log statement should end with a carriage return." ("Please log in by typing the word LOG followed by your name and address.") ($beger2$)

3. "Intrex could not find your name in your log statement." ($beger3$)

4. "Greetings! This is Intrex. Please log in by typing the word LOG followed by a space and your name and address as in the following example:
   log smith, rj; mit 13-5251; ext 7234
   Note that your log in statement should end with a carriage return." ("Please log in.") ($sin1$)

5. "Welcome to Intrex M. xxxxx. If your already know how to use Intrex, you may go ahead and type in commands. (Remember, each command ends in a carriage return.) Otherwise, for information on how to make simple searches of the catalog, type info 2
   or, to see the table of Contents (Part 1) of Intrex Guide which will direct you to other parts of the Guide explaining how to make more detailed searches, type
   info 1" ("Welcome M. xxxx") (/sin2a/, /sin2/)

I. Length:

150_8 or 104_{10} words

J. Source:

SQUIRE ALGOL

K. Files Referenced:

None

3.1.4.3 EXIT

Purpose

To tell the user how to exit from INTREX

Description

A. Operation: Code = EXIT( )

EXIT prints a message to the user suggesting that he make some comments about the system before he terminates his retrieval session.

B. Procedures Calling EXIT:

CLP (via CALLIT)

C. Procedures Called by EXIT:

TYPEIT

D. COMMON References:

None

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

1. "We would appreciate your comments on the Intrex system. For information on how to make comments, see Part 13 of Guide or type.

   info 13

   You may also make additional service requests of the Intrex consultant. If you do not wish to make any other comments or requests, type

   quit." ("Please comment or quit.") (/exmes/)

I. Length:

23_8 or 19_{10} words

J. Source:

SQUIRE ALGOL

K. Files Referenced:

None

3.1.4.4 QUIT

Purpose

To exit from INTREX

Description

QUIT terminates an INTREX session and returns the user to CTSS command level. If the INTREX system is in HOLD mode (i.e., CLAMP (SST.) = TRUE), the subsystem INXSUB will take control and return the user to INTREX.

A. Operation QUIT(Ptr)

Quit is called by three different subroutines: CLP, SIGNIN, and GO. When a user terminates a retrieval session by typing QUIT, CLP calls the procedure QUIT. If a user responds to SIGNIN's request to log in by typing QUIT, SIGNIN will call QUIT. Finally, a user can terminate a retrieval session by typing BEGIN, which will cause CLP to transfer control to GO, which will transfer control to QUIT. The inner workings of QUIT are as follows:

1. If the argument Ptr of QUIT is a pointer, the string that it points to is compared with ESCODE(POT.). If they are equal, the HOLD mode is turned off by setting CLAMP(SST.) to zero. In either case, the message "Password received . . . . (PASSWORD)" is printed.
2. The Dump File DUM00x FILE, is truncated to zero.
3. If the monitor is on, a timing summary is added to the Monitor File and to the file TIMING SUMARY by means of a call to SUMOUT.
4. CATDIR is closed.
5. If control has not passed to QUIT from GO (the argument of QUIT will be a 1 if it has) the message "Thank you for using INTREX" will be typed.
6. If the system is not in HOLD mode, the A-core option register is set to zero and DORMNT is called.
7. If the system is in HOLD mode, the Password File is extended by one word. If QUIT was called by GO, CHNCOM is called; otherwise, DORMNT is called.

QUIT communicates information to the INTREX subsystem INXSUB, (Section 3.1.10.3) in two different ways:

1. QUIT lengthens the password file to 3 words. This indicates to

INXSUB that control was deliberately returned by means of a QUIT or BEGIN command, rather than by means of a system error.

2. QUIT returns to CTSS via CHNCOM if the user has typed QUIT, or via DORMNT if the user typed BEGIN. INXSUB can distinguish between the two by examining the bits in the subsystem condition code.

B. Procedures Called By QUIT:

BCDASC, BUFFER, BZEL, CHNCOM, CLOSE, CTSIT6, DORMNT, FRET, LDOPT, NEXITM, RJUST, SETSYS, SETWRD, SUMOUT, TRFILE, TYPEIT, WRWAIT

C. Procedures Calling QUIT:

CLP (via CALLIT), SIGNIN, GO

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
PFN1(POT.)	Name1 of password file	x	
COMBFO(POT.)	Common buffer 0	x	
CATS2(POT.)	Name2 of catalog	x	
MFUN1(POT.)	Name1 of monitor file	x	
DFN1(POT.)	Name1 of dump file	x	
ESCODE(POT.)	Escape code	x	
CLAMP(SST.)	Hold mode	x	x

E. Arguments:

Ptr: ASCII pointer if called by CLP or SIGNIN, binary 1 if called by GO

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Error in writing password file. No automatic resumption of Intrex." ($passer$)

2. "Password received--<u>pass</u>." ($passok$)

3. "Thank you for using Intrex!" ($outmes$)

I. Length:

277_8 or 191 words

J. Source:

SQUIRE ALGOL

K. Files Referenced:

CATDIR	INTREX
DUM00x	FILE
PAS00x	FILE

3.1.5 Time Controls

3.1.5.1 MONINT

Purpose

To initialize MONTIM, TRANS, SUMOUT

Description

A. Operation: MONINT( )

MONINT is called by INIFIX to initialize MONTIM, TRANS and SUMOUT. MONINT generates a number of short ASCII strings by calls to .C.ASC. Three eleven-word arrays are set up to hold timing data. Their locations are deposited in the common words MODS(POT.), CPUS(POT.) and REAS(POT.). MODS(POT.) points to an array containing counts of the frequency of calls to various modules. The array pointed to by CPUS(POT.) contains central processor times and REAS(POT.) refers to real times. MONINT calls FRALG before returning to INIFIX.

B. Procedures Calling MONINT:

INIFIX

C. Procedures Called by MONINT:

FRALG, FREZ, .C.ASC

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
MODS(POT.)	Module call count	x	
CPUS(POT.)	CPU Times	x	
REAS(POT.)	Real Times	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

110_8 or 72_{10} words

J. Source:

MONTIM ALGOL

K. Files Referenced:

None

3.1.5.2 MONTIM

Purpose

To monitor computer and user times

Description

A. Operation: MONTIM(Mode, T.array, Label)

MONTIM updates the timing array pointed to by T.array. Depending on the value of Mode, it will also update the summation arrays, write a message on the console, or write a message in the Monitor File.

The second argument of MONTIM, T.array, is a pointer to a four word timing array. If the first word of this array is zero, MONTIM is being called for the first time for that array. In this case, MONTIM calls JOBTM to place the total CPU time in the first and third words of the array. The current time of day, obtained from GETIME, is placed in the second and fourth words, and MONTIM returns to its calling program.

If the array has already been initialized, MONTIM uses the array to compute the following:

1. The total elapsed CPU Time
2. The total elapsed Real Time
3. The elapsed CPU time since the last update of the array
4. The elapsed Real Time since the last update of the array

The third word of the array is replaced with the current CPU Time and the fourth word with the current Real Time.

If Mode has a value of one, timing data will be printed on the console in the following format:

```
Time Used 3.18/105.65
```

The first number is the elapsed CPU time since MONTIM last referenced T.array, and 105.65 is the total CPU time since the timing process began.

If Mode has a value other than one, the following line is written in the Monitor File.

TIME (label) 0.18/10.80 2.10/208.32

The string represented by (label) is given to MONTIM by its second argument. The second pair of numbers represents the incremental and the total Real Time.

If Mode is 2, the summary arrays are updated. The last 6 bits of label are used to index the arrays.

B. Procedures Calling MONTIM:

CALLIT, GETLIN, INIMON, MONTOR, SUPER, TYPEIT

C. Procedures Called by MONTIM:

ASIDE, GETIME, JOBTM, TRANS, TYPEIT

D. COMMON References

Name	Meaning	Interrogated?	Changed?
MFUN1(POT.)	Monitor File Name 1	x	
MFUN2(POT.)	Monitor File Name 2	x	

E. Arguments:

Mode = 0:	write timing message in monitor file
= 1:	write timing message on console.
= 2:	write timing message in monitor file and update summation arrays.
T.array:	location of four-word array
Label:	TYPEIT message label - last six bits are used as index for timing summation arrays.

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "TIME USED 3.18*/106.65**

I. Length:

506_8 or 326_{10} words

* Elapsed CPU time since timing array was last referenced

** Total CPU time since first reference

J. Source:

MONTIM ALGOL

K. Files Referenced:

None

3.1.5.3 TIME

Purpose

To carry out user's time request

Description

A. Operation: Code = TIME (Ascptr)

The subroutine TIME is called by CLP in response to the user's command "time". If the command is followed by the argument "on" or no argument at all, TIME will set TIMES(SST.) to TRUE. The the command is followed by "off", TIMES(SST.) is set to FALSE. If the command is followed by some other character string, an error message is printed.

B. Procedures Calling TIME:

CLP (via CALLIT)

C. Procedures Called by TIME:

COMPUL, LOCMES, NEXITM, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
TIMES(SST.)	Timing mode		x

E. Arguments:

Ascptr: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

1. "Improper time request" (LOCMES)

I. Length:

310_8 or 200_{10} words

J. Source:

MONTIM ALGOL

K. Files Referenced:

None

3.1.5.4 SUMOUT

Purpose

To write timing summation

Description

A. Operation: SUMOUT ( )

SUMOUT is used to write out a summary of timing data into the Monitor File at the end of a retrieval session. It is called by MONTOR or QUIT in response to the user commands "monitor off" or "quit".

SUMOUT performs the following steps:

1. The Monitor File is closed by the call INIDSK(0).
2. A temporary file "NEW SUMARY" is opened via INIDSK
3. The following information is written (via ASIDE) into "NEW SUMARY"

```
Summary File for Intrex system--012571 monitor file--mon001 file
                 total cpu time is -- 109.73 secs
                 total real time is-- 927.16 secs
```

module name	no calls	total cpu	ave cpu	per cpu	tot real	ave real	per real
signin	19	5.87	0.30	5	88.74	4.65	9
sign2	19	7.03	0.36	6	7.41	0.38	0
clp	16	29.42	1.83	26	516.11	32.24	55
fso	4	47.91	11.97	43	172.94	43.23	18
eval	7	6.49	0.91	5	21.40	3.05	2
search	8	11.74	1.45	10	12.80	1.60	1
int1	1	1.25	1.25	1	107.69	107.69	11
int2	1	8.61	8.61	6	36.28	36.28	0

This table is an example of the type which is constructed using the data that has accumulated in the arrays MODS(POT.), CPUS(POT.) and REAS(POT.)

4. The file NEW SUMARY is appended to the end of the Monitor File and TIMING SUMARY.
5. NEW SUMARY is deleted.

B. Procedures Calling SUMOUT:

MONTOR, QUIT

C. Procedures Called by SUMOUT:

ASIDE, BCDASC, BFWRIT, BUFFER, CLOSE, DELFIL, INIDSK, LOCMES, OPEN, RDWAIT, TYPEIT, WRWAIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBF1(POT.)	Common buffer	x	
MFUN1(POT.)	Monitor Filename 1	x	
MFUN2(POT.)	Monitor File name 2	x	
MFN1(POT.)	Monitor File name 1	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Error in writing summary file. Error code x" (LOCMES)
2. "Error in adding timing summary to Monitor File" (LOCMES)

I. Length:

702_8 or 450_{10} words

J. Source:

MONTIM ALGOL

K. Files Referenced:

TIMING SUMARY
MONnnn FILE
NEW SUMARY

3.1.5.5 TRANS

Purpose

To convert binary times to ASCII minutes

Description

A. Operation: Ptr = TRANS(Time)

TRANS is used by MONTIM and SUMOUT to convert binary-coded decimal representation. TRANS returns two values: a pointer to the number of seconds (or minutes) represented by Time and a pointer to the number of hundredths of a second (or minute) beyond the first value that are represented by Time. The first pointer is expressed as the value of TRANS; the second pointer is a variable (Rptr), held in common with TRANS, MONTIM, and SUMOUT.

B. Procedures Calling TRANS:
MONTIM, SUMOUT

C. Procedures Called by TRANS:
COPY, DEC1, INC, INC1, INTASC, PUT

D. COMMON References:
None

E. Arguments:
Time: time expressed in 60th's of a second (binary value).

F. Values:
Ptr = pointer to ASCII representation

G. Error Codes:
None

H. Messages:
None

I. Length:

217_8 or 143_{10} words

J. Source:
MONTIM ALGOL

K. Files Referenced:
None

3.1.6 Monitor File Control

3.1.6.1 INMON2, INIMON

Purpose

To initialize MONTOR and TIME

Description

A. Operation: INMON2( ), INIMON( )

INMON2 and INIMON initialize the routines MONTOR and TIME. INMON2 is called during the overlay initialization phase and is called by SEGINT. INIMON starts up the timing and monitoring process. It is called by DYNAMO during the session initialization phase.

B. Procedures Calling INMON2, INIMON:

SEGINT (INMON2)
DYNAMO (INIMON).

C. Procedures Called by INMON2, INIMON:

.C.ASC (INMON2)
GETTM, MONTIM, MONTOR (INIMON)

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
STM.(POT.)	System time monitor	x	
RTM.(POT.)	Ready message time monitor	x	
TESTIT(SST.)	Test mode	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length

140_8 or 96_{10} words

J. Source:

MONPAK ALGOL

K. Files Referenced:

None

3.1.6.2 INIDSK

Purpose

To direct output stream onto disk or console

Description

A. Operation: INIDSK(Name1, Name2, Mode)

INIDSK can be called with one, two or three arguments. If INIDSK has one argument with a value of 0, the Monitor File is closed. If the argument is not 0, an error message is printed. If INIDSK has two arguments, the console is turned ON. If the arguments are non-zero, they are used as the names of a Monitor File, which is opened. If INIDSK has three arguments, the first two are used as the name of a Monitor File to be opened. If arguments one and two are zero, the Monitor File is closed. The third argument will turn off the data flow to the console if it is set to zero and will turn the console on if it is set to one.

B. Procedures Calling INIDSK:

DYNAMO, FSO, GETLIN, MONTOR, SUMOUT

C. Procedures Called by INIDSK:

BFCLOS, BFOPEN, FSTATE, LOCMES, SETWRD TYPASH

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
MONBF1(POT.)	Monitor buffer 1	x	
MONBF2(POT.)	Monitor buffer 2	x	

E. Arguments:

Name 1:	1st name of Monitor File
Name 2:	2nd name of Monitor File
Mode:	0- console off, 1-console on

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Error in opening disk output file." (LOCMES)
2. "Error in closing disk output file." (LOCMES)
3. "Improper Inidsk arguments." (LOCMES)

I. Length:

217_8 or 143_{10} words

J. Source:

TYPINT ALGOL

K. Files Referenced:

MONnnn FILE
CATII OUTPUT

3.1.6.3 MONTOR

Purpose

To open or close Monitor File

Description

A. Operation: Code = MONTOR (Ascptr)

MONTOR is called by CLP in response to the user command "monitor". If there is no argument following the command, or if the command is followed by "ON", the Monitor File is opened and a header is written. If the command is followed by "off", it is closed. If some other character string follows the command, an error message is printed.

MONTOR opens the Monitor File by calling INIDSK (Section 3.1.6.2) with the arguments "MFN1(POT.), FILE, 1". If the Monitor File is already open, INIDSK will do nothing. MONTIM is called with a mode of 0, which causes a message like the following to be written in the Monitor File:

```
TIME monom 0.13/0.13   0.00/00
```

This line is followed by a form feed and a message of the following type:

```
"Today's date is 021671
 32 past 14 TO289 6162   MIT8B7  OVL900008
 No holding password"
```

This message can be explained as follows:

T0289 6162:	problem number and programmer number
MIT8B7:	name of the CTSS operating system currently in use
OVL:	Name of the current INTREX system
900008:	Address of console
No holding Password:	Printed if user has not typed a password, x. If he has, the message "Password is x" appears.

B. Procedures Calling MONTOR:

CLP (via CALLIT)

C. Procedures Called by MONTOR:

ASCIT6, ASIDE, BCDASC, COMPUL, INIDSK, MONTIM, NEXITM, SUMOUT, TYPEIT, WHEN, WHOAMI

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
MFUN1(POT.)	Monitor File Name 1		x
MFUN2(POT.)	Monitor File Name 2		x
STM.(POT.)	System Time Monitor	x	
MFN1(POT.)	Monitor File Name 1	x	
SYSNAM(POT.)	System name	x	
CLAMP(SST.)	Hold mode	x	

E. Arguments:

ASCPTR: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

1. "Improper monitor request" (LOCMES)

I. Length:

324_8 or 212_{10} words

J. Source:

MONPAK ALGOL

K. Files Referenced:

None

3.1.6.4 ASIDE, ASSET

Purpose

To write data in Monitor File

Description

A. Operation: ASIDE (Argl, . . ., Argn), ASSET( )

ASIDE is a means of writing data into the Monitor File without also writing it on the console. A procedure callsASIDE with the same sort of arguments a call to TYPEIT would have. ASIDE saves the contents of index register 4, which contains the location of the call to ASIDE. ASIDE then calls ASSET, which is associated with TYPEIT. ASSET turns on a flag, which puts TYPEIT into the "aside" mode. After regaining control from ASSET, ASIDE restores the contents of index register four and transfers control to the entry point of TYPEIT. As far as TYPEIT can tell, it is being called directly from the subroutine which called ASIDE. TYPEIT transmits the message specified by the argument list, turns off the "aside mode" flag, and transfers control back to the program which called aside.

B. Procedures Calling ASIDE, ASSET

ASIDE: GETLIN, LISTEN, MONTIM, MONTOR, PUTS, SEARCH, SUMOUT, SUPER, TYPEIT, WRT

C. Procedures Called by ASIDE, ASSET

ASIDE: ASSET, TYPEIT
ASSET: None

D. COMMON References:

None

E. Arguments:

Argl... Argn: see description of TYPEIT (Section 3.1.7.2)

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

10 words (5 each)

J. Source:

TYPINT ALGOL (ASSET), ASIDE FAB (ASIDE)

K. Files Referenced:

None

3.1.7 Typing Controls

3.1.7.1 INITYP

Purpose

To initialize TYPEIT

Description

Before the main typing procedure TYPEIT can be used in the mode that accepts a (core-or-disk-stored) message label as an argument and outputs the corresponding message, a Message File and Directory must be generated by DIRGEN (see Section 5.1). DIRGEN creates a Message Directory File and Message or "text"File from a disk-stored ASCII file containing messages (or message segments). These files are activated by INITYP.

A. Operation: INITYP(NAME1)

INITYP reads the Message Directory into core into an array obtained from free-storage. If a subsequent call to INITYP is made to change Message Files, the old storage area is returned before a new one is obtained. This is accomplished by using local variables to store the address and length of the current directory and setting a flag (named IN) to indicate that a directory is stored. When the presence of this flag is detected on entering INITYP, the indicated area is returned to free storage via FRET and the current text file is closed by the CTSS procedure CLOSE.

Before attempting to open the new files, the special word in the CTSS supervisor (which Intrex uses to control the error-recovery subsystem) is set to the "stop" code to prevent resumption of Intrex if an I/O error occurs. First the CTSS procedure, GETWRD, is called to extract and save the current contents of the A-core word. Then the word is set to "stop" by the routine SETWRD.

Now the length of the message directory, whose first name is supplied in the argument of the call to INITYP and whose last name must be DIRTAB, is obtained by calling FSTATE. Unless the data from FSTATE shows that the file is already open, the directory file and the text file (whose first name is the same and whose last name is TEXT) are both opened for reading via the CTSS procedure, OPEN.

Next, the storage area for the directory is obtained by a call to FREE, with the directory length as an argument. IF FREE returns an error code instead of an address (indicating that not enough storage is available), the address parameter is set to the top of core to force a protection violation to occur when RDWAIT attempts to read the directory into core at that address. This causes the I/O error mechanism to take the standard exit through EREXIT in INIRES (see Sections 3.1.10 and 3.3.2.1).

If no such error has occurred, the directory is read into core via RDWAIT starting at the address provided by FREE. If an I/O error occurs, the system will automatically transfer to EREXIT as predetermined by calling the CTSS procedure FERRTN during the initialization phase of Intrex (Section 3.1.10.2). If an end-of-file marker is encountered, then a discrepancy exists between the directory length reported by FSTATE and the number of records attempted to be read by RDWAIT. Transfer, in this case, is automatically made to a local error exit where an explanatory message (1) is printed before calling DORMNT (which terminates the program).

When the reading of the directory and core-stored message text is properly completed, SETWRD is again called to return the supervisor word to its original value. The Message Directory is then closed (but not the Message File) and the flag IN is set which indicates that a current directory exists in core. The name of this file is saved in a local variable for possible future calls to CLOSE from INITYP.

Finally, another flag, TYPE, is set which will be used by TYPEIT to determine if the message it is constructing is to be output at the console (as opposed to written onto the disk).

B. Procedures Calling INITYP:

DYNAMO, GETLIN, INFO, LONG, SHORT, TYPEIT

C. Procedures Called by INITYP:

FRET, CLOSE, GETWRD, SETWRD, FSTATE, OPEN, RDWAIT, FREE.

D. COMMON References:

None

E. Arguments:

NAME 1: BCD file name

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Premature end-of-file reading message directory. Computer words read = N ," (LOCMES, integer)

NOTES:

(Note: The actual number of computer words read is given by N.)

I. Length:

120_8 or 80_{10} words

J. Source:

TYPINT ALGOL

3.1.7.2 TYPEIT

Purpose

To output messages, both on-line and off-line

Description

TYPEIT is a basic routine supporting the system/user dialog. It prints messages to users in upper and lower case and is called with a sequence of arguments that represent the message to be typed. These arguments can include the following:

1. Message fragments which may be disk or core stored identified by symbolic names.
2. Locally declared or constructed ASCII messages pointed to by standard ASCII pointers.
3. Integers given by their binary values.
4. BCD variables specified by a single computer word BCD string declared in the calling program.
5. Mode codes that govern general features of the output.

The following comments correspond to the numbers and types of arguments listed above.

(1) Message fragments may be pre-stored in core or on the disk and symbolically referenced by (.BCD. declared) labels in the calling program. This allows changes in the message text without re-compiling the calling program.

The .BCD. integers used as message labels must be left-justified.

The labels may consist of one to six characters, the first of which must be alphabetic.

(2) An ASCII pointer to a locally declared .C. character string may be created by using the sub-procedure LOCMES (see Section 3.1.7.3) which converts the .C. string to ASCII and returns a pointer that may be used as an argument to TYPEIT.

(3) Binary numbers (within the range $\pm 777777_8 = \pm 262143_{10}$) are converted automatically by TYPEIT to ASCII codes and made part of the composite message string.

The number -0 may not be typed since it has a special meaning. (See special modes.)

(4) .BCD. variables which are not message labels (such as file names) may be printed as ASCII strings by using the subprocedure BCDASC (See Section 3.1.7.4).

(5) The special modes control the format of the output produced by TYPEIT. Before listing these modes and their descriptions, we first present information about the output formats.

(OUTPUT FORMATS)

In the standard mode, the composite message string which TYPEIT generates from its arguments is printed on the CTSS console according to the following formatting rules:

1. Carriage return codes in the message string are changed to space codes, except when 2 or more appear in sequence. When n consecutive carriage returns appear, TYPEIT will retain n-1 of them.

2. Carriage returns are inserted after the last word before a limit of n characters is exceeded on a line.

   n = 75 for 2741 or ARDS
   n = 55 for Intrex console

   (These two rules allow text formatted for any line length to be output at a different line length.)

3. A space is added after each message element.

4. A carriage return is inserted at the end of the complete message.

5. Wherever conversion to ASCII is performed (by LOCMES or BCDASC) all letters will become lower case except those preceded by a $.

(SPECIAL MODES)

1. ASIS: If a pointer or label argument (type 1 or 2) is preceded by a -0, that message element will not be formatted in the ways described by 1 and 2 above, but will be printed "as is".

2. EDGE: If a TYPEIT argument is preceded by an argument consisting of the .BCD. variable "EDGE", then rules 1 and 3 above will not apply but rule 2 will apply when needed. Thus, the EDGE mode combines the carriage return controls of both the normal and the ASIS modes and is used to output the tubular format of such catalog fields as authors or subjects.

3. SPACE OMISSION: If one or more TYPEIT arguments are preceded by an argument consisting of the .BCD. variable "SMON", then rule 3 will not apply until either an argument of .BCD. "SMOFF" is seen or until all the arguments of that call are processed.

4. CONTINUED CALLS: If a call to TYPEIT contains the .BCD. variable "CONT" anywhere in the string of arguments, the characters-per-line counter is saved to be used by the next call to TYPEIT. The saved counter is incremented further by the new call. If a CONT argument is contained in the new call, the count is again passed along to the next call. Each CONTINUED call causes the text produced by it to be printed terminated by a space unless the maximum line length is reached. Thus, format modification 4 (ending c.r) is bypassed.

   Note: If the space omission mode described in 3 above is in use in conjunction with the CONT mode, the space mode will not automatically be turned off at the end of the call to TYPEIT but only by a SMOFF argument.

(MONITOR FILE CREATION)

TYPEIT allows the writing or output onto a disk file (in ASCII) for later or off-line printing. Such output may or may not be printed simultaneously on the console. The writing and/or printing is controlled by calling a sub-procedure named INIDSK. This procedure is described in Section 3.1.6.2 which covers "Monitor File Control".

A. Operation: TYPEIT(ARG1, ARG2, ---ARG_n)

As described in Section 3.1.8 (Interrupt Controls), the occurrence of an interrupt, caused by the user pressing the ATTN button to halt the printing (or display) of output or message text, is not detected by Intrex until the next call to TYPEIT. If the interrupt occurs during the processing of a TYPEIT call, it will probably be detected during the execution of that call. This possibility is discussed later in this section (see PUTS).

TYPEIT's first task is to check the indicator set (see INTONE, Section 3.1.8.2) when an interrupt occurs at level one, that is, outside of TYPEIT. If this flag, INT1(SST.), is set, the TYPEIT call is not processed. Instead, transfer is made to a section of TYPEIT which will record the interrupt address and time in the Monitor File and reset the interrupt flag. A CTSS procedure named GETBRK (Section 3.5.7.4) is used to obtain the core location where the program was operating when the interrupt took place. ASIDE (Section 3.1.6.4) is then used to write the interrupt message (1) into the Monitor File.

Before MONTIM (Section 3. 1. 5. 2) can be called to record the time of the interrupt, TYPEIT must ascertain that the Message File, not the Guide File, is currently active. It does this by comparing the name last saved by INTYP to the first name of the Guide File. If the names match, VERBOS(POT.) determines whether the long or the short Message File and Directory should be re-activated by another call to INITYP. If this switch of directories is done here in TYPEIT, the flag set by the INFO command (see Section 3. .9.7), INFOX(SST.), is reset to [illegible]nt an unnecessary repeat of this action later in the system.

Now MONTIM is called with one of the arguments containing a message label whose text is "INT1" to tag this timing-message in the Monitor File.

Transfer back to the Intrex supervisor is made by calling the procedure LISTEN, which is compiled with the "main routine" of Intrex in the source file, SUPER ALGOL. The function of LISTEN and its connection to the supervisor is described in Section 3. 1. 8. 4.

If the interrupt flag is not found to be set upon entering TYPEIT, then another indicator, ASIDEM, is examined to determine if this call to TYPEIT is from the procedure ASIDE (see Section 3. 1. 6. 4). ASIDE is designed for entering messages into the Monitor File without allowing them to be typed on the console. ASIDE precedes its call to TYPEIT by a call to ASSET, a small procedure within TYPEIT which sets the above-mentioned mode indicator, ASIDEM. When TYPEIT sees that ASIDEM is set, it resets the typing flag, TYPE. This will later prevent TYPASH from being called (see PUTS below), thus inhibiting console output.

If the Monitor File has been "turned off" either by the user through a MONITOR OFF command, or by resuming Intrex in a "test session" mode (see DYNAMO in Section 3. 1. 3. 1), then the WRITE flag will be off. If both WRITE and TYPE are off, then TYPEIT returns immediately to the calling program.

If TYPEIT passes the above tests, the interrupt level is raised to 2 by calling the CTSS procedure SETBRK (Section 3. 5. 7. 5) with the address of the procedure INTTWO passed as an argument. SETBRK allows the programmer to specify where he wants control transferred when the ATTN key is pressed. Level 2 will then mean "interrupted from within TYPEIT".

TYPEIT uses two counting parameters, CHARCT and OLDCNT, to keep track of the number of characters placed in the message buffer—an array within TYPEIT which is used to hold the constructed ASCII message string for outputing. OLDCNT, used primarily in the "CONT" mode where lines may be constructed from multiple calls to TYPEIT, is set now to contain the old contents of CHARCT (from the previous TYPEIT call) before CHARCT is reset to zero to process the present call.

Next, the maximum line length is extracted from the POT and deposited into the local parameter MAXLEN.

TYPEIT now proceeds to store the arguments passed to it by the calling program into a fixed length array called MESLIS. Up to fifteen arguments may be given to TYPEIT on any single call. After argument one is placed in the first location of MESLIS, the AED procedure ISARGV (Section 3.6.1.3) is called repetitively to obtain the next argument and deposit it into the next location of MESLIS until the argument-terminator, a word containing all octal 7's, is found. The number of arguments found and stored is saved in a local parameter named N. The argument counter, I, used in counting and indexing MESLIS storage, is then reset to zero in preparation for counting the arguments as they are selected from MESLIS and processed.

Setup*

An array named MESSTG is used to hold the assembled characters of the message. Its starting address is stored in two pointers, MESSPT which will point to the next byte of MESSTG to be filled or examined and PNTPTR which will determine the number of characters from MESSTG to be typed on each line of output.

* This and subsequent headings relate to program labels and are given to provide the reader with reference points in the lengthy description.

The next task for TYPEIT is to determine if indentation is necessary and, if so, supply the specified number of spaces. The sub-procedure INDENT (Section 3.1.7.5) sets a local parameter named TAB to the number of spaces desired for indentation as supplied to INDENT in its argument. It also sets a flag named DENT to inform TYPEIT that indentation is required. If DENT is TRUE and CHARCT is still zero, TYPEIT will insert the number of spaces indicated by TAB into MESSTG via use of the string-utility procedure, PUT (Section 3.4.4.3). The pointer, MESSPT, is moved up after each PUT by another utility procedure called INC1 (Section 3.4.4.18) and the character count, CHARCT, is also incremented by one.

Next

The next argument to be selected from MESLIS (as indexed by I) is examined to determine its type. First, it is tested to see if it is a negative zero by calling an Intrex utility procedure named TESTMO (Section 3.4.5.8). An argument of -0 means that the ASIS mode is to be used. Therefore, the ASIS indicator is set and the message buffer is emptied to start the new message on a new line. If the buffer is already empty (CHARCT = 0), then the character count is raised to one and the message pointer is moved up one nine-bit byte to allow for the insertion of a carriage return. TYPEIT then transfers ahead to PREND where the buffer contents are printed (in case previous arguments had stored some text). This is usually the area of TYPEIT which terminates processing of the procedure, but in this case (ASIS set) TYPEIT will come back to SETUP.

If the argument being processed is not a -0, a second test is made to see if it is an integer. The left-most eighteen bits of the word are checked, and if they are not filled, the argument is assumed to be an integer, which is then converted from binary to ASCII by a call to INTASC (Section 3.4.2.3), which returns a pointer. The internal sub-procedure MOVEIN is then called to copy those codes into the message buffer. TYPEIT then transfers to "More", where the argument count is incremented and tested against N to see if any more remain to be processed.

If the argument fails the integer test, it is tested to see if it is an ASCII pointer to an in-core string of characters. Two tests must be made to distinguish ASCII pointers from message labels. A message label must begin with an alphabetic character to ensure that the left-most octal digit will be non-zero. This means that, if the argument is a message label, the sign bit and/ or bits 1 and 2 will be a one. Bits 1 and 2 are tested by masking, but the sign bit must be tested for a negative condition to determine if the bit is on. If none of these bits are on, the argument is assumed to be an ASCII pointer. This pointer is then passed to the sub-procedure, MOVEIN which copies the characters pointed to into the message buffer. Having done this, TYPEIT transfers to "More".

When any of the bits 0-2 are one, the argument is assumed to be a message label or a special, BCD mode code. A series of tests is made next to determine if it is a mode.

If the argument is the word CONT, the indicator RUNON is set and TYPEIT transfers to "More".

If the argument is the word EDGE, the indicator DUAL is set and TYPEIT transfers to "More".

If the argument is the word SMON, the indicator NOSPAC is set and TYPEIT transfers to the statement just beyond "More", skipping the logic that resets the ASIS indicator.

If the argument is the word SMOFF, the indicator NOSPAC is reset and TYPEIT transfers to after "More", as with SMON.

If the argument is none of these things, it is assumed to be a message label and is passed to GETLAB for lookup in the Message Directory.

<u>Getlab</u>

GETLAB compares the argument passed to it by TYPEIT with the BCD-coded labels in the Message Directory. The message Directory contains pairs of words consisting of a [illegible] and a pointer to the corresponding message text. When a matching label is found in the directory, the pointer in the next word is extracted, disected, and tested to see whether it is core or disk-stored. If it is a core-stored message, the text is at the end of the directory and it is distinguished by the presence of a one in bit 2 of the pointer. The address portion of the pointer contains the relative location or "offset" of the text, either within the Message File or within the core-stored directory. If it is core-stored, this relative address is deposited into a parameter common to TYPEIT(REL) and the top of the directory storage area is used as the base address of the message text. The byte address and length are contained in the pointer which, minus the bit which flagged it as "core-stored", is returned as a value to TYPEIT.

If the pointer indicates a disk-stored message (zero in bit 2), common buffer three is used as the base address of the message text, and the offset or relative address (REL) is set to 0. One full record (432 words) of the Message File is read into this buffer, starting at the file address found in the address of the pointer. Since no message or Guide Section should contain more text than can be contained in one record, this single read should be sufficient to put the entire message in core. If the length (number of characters) in the decrement shows that there is more than one record-full, then the length is truncated. This truncation is indicated to the calling program (most likely, INFO) by placing a 1 in the first argument of the call. The text pointer is then returned as a value from GETLAB to TYPEIT.

If the label passed to GETLAB is not found in the directory, an error message (4) is printed via the procedures LOCMES and TYPASH. An exceptional case is made when the call to TYPEIT is from the procedure INFO, indicated by the fact that the first name of the Message File will be GUIDE. In this case, no message is typed but the first argument of the call to TYPEIT is changed to zero. In both cases, TYPEIT transfers ahead to the final clean-up area, "Frit", which terminates this call.

Upon successful return from GETLAB, the base address and offset are added and inserted into the address portion of the pointer provided by GETLAB. This completes the text pointer which is now passed to the sub-procedure MOVEIN, which copies the text into the message buffer at the point dictated by MESSPT. TYPEIT then transfers to "More".

Movein

This sub-procedure is called by TYPEIT whenever a string of ASCII characters is ready to be copied into the message buffer. A pointer to the string is passed as an argument to MOVEIN, which saves it in a local variable named DATAPT before beginning to process the codes. The decrement of the pointer, containing the number of codes to be transferred, is also extracted and saved in parameters MT and NCHARS. These two counts will control the deposit of characters into the message buffer to prevent its overflow. If more characters are about to be added to the buffer than it can hold, then MT is reduced to a number within the bounds of the buffer size. MT is then used as the terminating value of a rather large loop which stores MT characters into the message buffer. When this group of characters has been deposited (and possibly emptied via CHKEND), NCHARS is reduced by MT and the new value of NCHARS is deposited back into MT. If NCHARS is not yet reduced to zero, MOVEIN transfers back to the test above, which determines if NCHARS, plus the current CHARCT, is greater than the capacity of the message buffer. This loop is repeated until NCHARS is reduced to zero, at which time MOVEIN returns to TYPEIT.

This is a broad overview of how MOVEIN cycles through the characters to be added to the message buffer and inserts them. Room to insert the next batch is ensured by the calls to CHKEND which are made at least at the end of each batch-insertion, and sometimes more often as explained below. CHKEND, described later in this section, checks to see if more characters have been stored in the message buffer than the length of one output line (as determined by MAXLEN). If this is the case, a carriage return is inserted at an appropriate word break and that amount of text is output. The remaining characters of the message buffer are then shifted down to the start of the buffer, making room for more. A detailed description of the characte -storing loop now follows:

First, the utility procedure GET (Section 3.4.4.1), is called to take the next character from the string being deposited, as pointed to by DATAPT, and to leave that character in a local word CODE. If CODE is a carriage return, it is changed to a space unless the special mode ASIS or EDGE is in use or the multiple carriage return flag, CARMUL, is set. If the carriage return is changed to a space, CARMUL is set in case more carriage returns follow. If the code is not a carriage return, CARMUL is [illegible]set.

Th[illegible] character in CODE is now inserted at the message buffer by the utility procedure PUT (Section 3.4.4.3) at the byte pointed to by MESSPT. The character count, CHARCT, is incremented by one.

If the code is not a carriage return, the two pointers MESSPT and DATAPT are both updated to the next byte by the utility procedure INC1 (Section 3.4.4.1[illegible]), and one cycle through the loop is completed. If, on the other hand, the code is a carriage return, then one of the special modes may be assumed to be on. If the mode is EDGE, indicated by the DUAL indicator being set, or if CARMUL is set, then the message buffer checking procedure, CHKEND, is called to see if any previous carriage returns need to be inserted earlier in the buffer because of extension beyond the maximum line length. If neither of the above modes are in use, then the ASIS mode must be. This mode is unconcerned about line lengths or about inserting additional carriage returns. The current number of stored characters is placed in the decrement of PNTPTR and the output sub-procedure, PUTS, is called to print and/or write into the Monitor File the contents of the message buffer up that point. PUTS (described below after CHKEND) will empty the buffer. Therefore, upon return to MOVEIN, the message buffer pointer MESSPT is reset to the first byte of the buffer and the character count is reset to zero. If the indentation flag DENT is set, TAB-2 spaces are deposited into the buffer via PUT. (The first line of output in any TYPEIT call is indented two spaces less than are subsequent lines).

Finally, DATAPT is updated to the next character in the input string and the cycle through the storage loop is complete.

When all the characters are stored, the procedure CHKEND is called in case the buffer has been filled to the point where one or more output lines should be flushed.

Chkend

The main task of CHKEND is to test for the possibility of the message buffer containing more output characters than are allowed on one printed line. This check is made by comparing CHARCT with MAXLEN. When CHARCT is greater and the ASIS flag is not set, then CHKEND must find a place to terminate the line and output it. This is not done when the ASIS mode is in use because no new carriage returns are allowed in that mode.

If the character count has not yet reached a line length, a space is inserted, via PUT, into the next byte of the buffer, to separate message components. This necessitates updating the buffer pointer MESSPT and incrementing the character count CHARCT. This space is not added if the NOSPAC, DUAL, or CARMUL flag is set. In all cases where CHARCT is less than MAXLEN, an early exit is made from CHKEND.

When the line length has been exceeded (CHARCT > MAXLEN) the function of CHKEND is less trivial. It must scan back over the message buffer looking for a space code (or hyphen) which might provide an appropriate place to terminate the line. The distance it will look back depends on whether or not CONT (continued line) mode is being used. In CONT mode, the scan may go all the way back through the string of characters inserted by the current call to TYPEIT, if necessary. If not in CONT mode, the scan is ended about one-fifth of a line from the beginning of the buffer. As soon as a word delimiter is found, the number of characters, K, scanned thus far is subtracted from the number in the buffer, and the difference is tested to see if it is still greater than MAXLEN. If so, the scan continues back toward the beginning of the buffer, looking for another space. If the allowed scanning distance is covered without finding a space, then a carriage return must be inserted in a less appropriate place. In the CONT mode, where the scan has moved the message pointer all the way back to the front of the buffer, the problem is handled rather simply. PNTPTR, which controls the output, is temporarily changed to point to a single carriage return. PUTS is then called to output the carriage return, PNTPTR is reset to point to the beginning of the message buffer, and both CHARCT and MESSPT are reset to reflect only the current contents of the buffer.

In other than CONT mode, a line break must be made in the midst of the unusually long string of characters containing no word delimiters. The decrement of PNTPTR is set to put out one full line of text and PUTS is called to output it. Then the carriage return is put out as described above for the CONT mode. The adjustment of the pointers and counters this time is much more complicated. First, the message pointer, which had been moved back almost to the front of the buffer, is moved up again to the next non-output character. Then K, which reflects the number of characters scanned in the search for a place to put the carriage return, is adjusted to the number of non-output codes left in the buffer. It is then used to set the decrement of MESSPT which, in turn, controls a call to COPY to move the remaining characters down to the front of the buffer.

This same push-down technique is used when a word delimiter is found during the scan back. Of course, if the delimiter is located at a position in the buffer still beyond a maximum line, then the scan is continued. When a good position is found, the delimiter is replaced by a carriage return via PUT. The decrement of PNTPTR is then set to the number of characters from the front of the buffer to this point, and PUTS is called to output this amount. Here, as mentioned above, K is used to set up MESSPT for the transfer of the remaining text to the front by the procedure COPY (Section 3.4.4.9). At this point, the message buffer pointers are reset to the front of the buffer. MESSPT is then advanced K bytes to the next available byte for incoming characters and CHARCT is adjusted to reflect the length of the current string.

Upon completion of any of the above described output-push-down operations, CHKEND transfers back to its opening test the current length of the buffer contents. It is possible that several such cycles will be necessary before the remaining text is less than one output line in length.

One detail not mentioned in the above description is the possibility of indentation after each line is output. Whenever a carriage return is put forth, the DENT flag must be tested. If the flag is set, the proper indentation, as prescribed by TAB, is output by pointing PNTPTR at a string of preset spaces and calling PUTS. This makes it necessary to include TAB in any computations involving the character counts.

<u>Puts</u>

The basic function of PUTS is to use the ASCII pointer in PNTPTR to output the specified number of characters on the consol (if TYPE is set) and onto the disk (if WRITE is set). Since PUTS is the trigger for output, it is a most appropriate procedure to test for interrupts. An interrupt by the Intrex user while processing is being carried on within TYPEIT will cause control to be sent to the procedure INTTWO (Section 3.1.8.3), where the System State Table flag, INT2, will be set. PUTS first tests this flag before attempting to produce any output. If the flag is set, it is reset and PUTS proceeds to reset the parameters of TYPEIT, such as the character counters and mode indicators. Any free storage used by the special conversion routines associated with TYPEIT, viz.,LOCMES and BCDASC, is returned by calling the sub-procedure FRETIT (described below). The address of the interrupt point is now obtained by calling the CTSS procedure GETBRK (Section 3.5.7.4). This address becomes part of the message (2) written into the Monitor File by a call to ASIDE (Section 3.1.6.4). The procedure MONTIM (Section 3.1.5.2) is then called with an argument containing a label of "INT2" to tag the timing message in the Monitor File. As in the case of interrupts at level one, transfer is made to the LISTEN procedure of the supervisor.

If no interrupt has set INT2 (sic.), PUTS checks to see if it should write the output text onto the disk by examining the WRITE flag (set and reset by INIDSK, described in Section 3.1.6.2). If WRITE is set, the CTSS procedure FSTATE (Section 3.5.3.3) is called to determine if the Monitor File, whose name is found in the POT, is open for writing. If it is not, then the file is opened by a call to BFOPEN (Section 3.5.2.1). The message buffer word currently pointed to by MESSPT is saved in a local storage address SAVEND, and the pointer itself is saved in another address named NULLPT. This prepares for the following process of zeroing out the bytes of the last buffer-word to be written which are not included in the output. These bytes must be blanked because only whole words can be written onto the disk via the CTSS procedure BFWRIT (Section 3.5.2.3). The tag of MESSPT is used to determine how many non-output bytes remain in the last word. PUT is called to deposit a null code in each of these bytes as pointed to by NULLPT. The word will be restored to its original contents at the end of PUTS' output tasks.

The number of computer words to be written is computed by dividing the number of characters in the decrement of PNTPTR by four. If the call to BFWRIT results in an I/O error, transfer is made to an error exit within TYPEIT. Here, an error message (3) is printed (via LOCMES and TYPASH). the A-core subsystem control word is set to STOP by a call to SETWRD, and processing is terminated by calling DORMNT.

If writing onto the disk is successful (or not called for) the TYPE flag is examined. If this flag is set, the console typing procedure TYPASH is called with an argument containing PNTPTR. This procedure, described in the following section, will convert the ASCII codes in the message buffer to 12-bit BCD codes and cause them to be printed on the typewriter console or displayed on the CRT.

Before PUTS returns, the last buffer word to be written is restored so that no characters will be omitted from the next group to be output.

More

As each argument to TYPEIT is processed, a check must be made to see if any unprocessed ones remain in the storage area, MESLIS. Most completed arguments will cause a transfer directly to "More", where the first step is to reset the ASIS-mode flag (because ASIS mode can apply to only one argument at a time). Mode arguments, which produce no actual text, are finished by transferring past this resetting of ASIS directly to the incrementation of the argument counter I. After I is increased by one, it is compared to the argument total in N. If I is still less than N, TYPEIT transfers back to "Next" to fetch the next argument.

Prend

The details of pointer and counter adjustments, mode tests, and the resulting switche[illegible] branches of logic in this area of TYPEIT have grown so complex as T[illegible] evolved into an almost all-purpose output device, that any attempt at a detailed explanation would be very involved. A better approach might be to present a more general explanation of the operation.. First, the message buffer pointer MESSPT is adjusted back one byte or not, depending on whether or not a space was added after the last argument was processed and the pointer was moved up to the next availab[illegible] byte.

In some modes, such as NOSPAC mode, this is not done.

Second, a carriage return is added to the current message buffer to complete the output from this use of TYPEIT, unless the RUNON flag indicates that CONT mode is being used. In any case, the current contents of the buffer are output by setting PNTPTR to the length dictated by CHARCT (minus any previous continuation characters counted by OLDCNT) and calling PUTS.

Next, unless in CONT mode, the NOSPAC flag is reset to FALSE, and the character counters are reset to zero. In CONT mode, these parameters are not reset but RUNON is reset to FALSE.

Now the ASIS flag is tested and, if found to be set, argument counter I is incremented and compared to N. If I is still less than N, TYPEIT transfers back to "Setup" to prepare the next argument for processing.

If ASIS is FALSE or all arguments have been processed, then TYPEIT is ready to terminate its task. Before wrapping up, however, it examines the level-two interrupt flag again to see if an interrupt has occurred since the last call to PUTS. If INT2(SST.) is set, transfer is made to the top of PUTS where the action previously described is taken. Otherwise, TYPEIT finishes by (1) resetting the ASIS and DUAL (EDGE mode) flags; (2) checking ASIDEM and, if set, resetting it and setting the TYPE flag which it had previously made FALSE; (3) calling the subprocedure FRETIT to return any free storage used by the conversion routines LOCMES and BCDASC (see below); and (4) calling the CTSS procedure SAVBRK (Section 3.5.7.6) to lower the interrupt level to 1.

Fretit

The BCD-to-ASCII conversion procedure, ASCITC, used by both LOCMES and BCDASC, uses free-storage to hold the ASCII characters it produces. The pointers to these areas are saved by those routines in an array called FREPTS (declared within TYPEIT). FRETIT then examines the index of this array, P, to see if any pointers have been saved since the last call to TYPEIT. If P is non-zero, the pointers are extracted from FREPTS, one-by-one, and the area pointed to is returned to free-storage by calling FRET (Section 3.4.1.2). When all areas have been returned, P is reset to zero and FRETIT returns to TYPEIT.

B. Procedures Calling TYPEIT:

ANDER, AND., AUTHOR, CHKNUM, CLP, CONNAM, DROP, ERRGO, EVAL, EXIT, FSO, GETFLD, GETLIN, IFSINT, IFSRCH, INFO, INIFIX, INIVAR, IN., LISFIL, LIST, LISTSL, MONTIM, MONTOR, NAME, NUMBER, OUT., PREP, QUIT, RANGE, REND, SEARCH, SEEMAT, SHORT, SIGNIN, SIGN2, SPCTRN, SUBJ., SUMOUT, SUPER, S.T, TABENT, TABLE, TIME, TITLE, USE.

C. Procedures Called by TYPEIT:

ASIDE, BCDASC, BFOPEN, BFWRIT, COPY, DEC1, DORMNT, FRET, FSTATE, GET, GETBRK, INC, INC1, INITYP, INTASC, ISARGV, LISTEN, LOCMES, MONTIM, OCTASC, PUT, PUTS, RDWAIT, SAVBRK, SETBRK, SETWRD, TESTMO, TYPASH.

D. Common References:

Name	Meaning	Inter-rogated?	Changed?
INT1(SST.)	inter. level-one flag	x	x
MAXCHR(POT.)	max. line length	x	
VERBOS(POT.)	dialog mode indicator	x	
DFSN1(POT.)	Message File (short) name one	x	
DFLN1(POT.)	Message File (long) name one	x	
INFOX(SST.)	INFO command flag		x
STM(POT.)	system time monitor array		x
COMBF3(POT.)	common buffer three	x	
INT2(SST.)	inter. level-two flag	x	x

E. Arguments: (any of the following types)

ARG: integer (treated as decimal integer), ASCII pointer, BCD-codes label or mode code

F. Values:

None

H.
1. "Interrupt at level 1 at location . . !"(LOCMES)
2. "Interrupt at level 2 at location . . !"(LOCMES)
3. "Error in writing disk output file . . !"(LOCMES)
4. " Label in call to Typeit not found in directory . . . . !"(LOCMES)

I. Length:

1742_8 or 994_{10}

J. Source:

TYPINT ALGOL

K. Files Referenced:

MON00n FILE ($1 \leq n \leq 10$)

3.1.7.3 LOCMES

Purpose

To generate a "local" ASCII message string

Description

When no disk-stored or core-stored message exists to provide the desired message text, LOCMES provides a convenient means of generating an in-core ASCII string and providing a pointer to this string which can be passed on to TYPEIT for outputting.

A. Operation: PTR = LOCMES (.C. /STRING/)

The AED language provides a means of declaring variable-length strings of BCD-coded characters. This facility, known as the .C. convention, is activated by the presence in the source code of the characters, .C.. The next character must be a space and the character following the space will be considered by AED to be the string delimiter. All characters between this delimiter and the next appearance of the same delimiter will be set up by AED as packed (six codes to a computer word) BCD character codes. The number of such codes is stored in the decrement component of the address immediately before the string. A pointer to the address of the first character is generated by AED and, in this case, used as the argument in the call to LOCMES.

LOCMES extracts the character count from the word before the string and inserts it into the decrement of a local pointer, BCDPTR. The address of the string is copied from the argument to the address of BCDPTR, which is then passed to the utility conversion procedure ASCITC (Section 3.4.2.9). This procedure returns a pointer to the converted ASCII string. The converted codes will be all in lower case unless preceded by a $ in the .C. string. (One $ makes one ASCII character upper case.)

Since the ASCII codes are stored by ASCITC in an area obtained from free-storage, the pointer to this string is saved in an array local to both TYPEIT and LOCMES named FREPTS. The index of this array is another parameter, P, shared by TYPEIT. It is incremented by LOCMES at each call and reset to zero at the end of a TYPEIT execution. This allows multiple uses of LOCMES as arguments to TYPEIT and prevents the waste of free-storage by enabling TYPEIT to continually return these areas after they have been used.

B. Procedures Calling LOCMES:

ANDER, IFSINT, IFSRCH, INIDSK, INIFIX, INIVAR, LISTEN, MONTOR, POT., PREP, REND, SHORT, STRCH, SUMOUT, TABLE, TYPEIT.

C. Procedures Called by LOCMES:

ASCITC

D. COMMON References:

None

E. Arguments:

.C. /STRING/: .C. Pointer

F. Values:

PTR = ASCII Pointer

G. Error Codes:

None

H. Messages:

None

I. Length:

41_8 or 33_{10} words

J. Source:

TYPINT ALGOL

K. Files Referenced:

None

3.1.7.4 BCDASC

Purpose:

To convert a BCD word to ASCII

Description:

Names of disk files and other CTSS oriented items are expressed in six-bit BCD codes packed into a single computer word. These are generally defined and created by .BCD. declaration in the AED source code of the various Intrex modules. BCDASC provides a convenient method a converting such a BCD string into ASCII where it can be printed by TYPEIT.

A. Operation: PTR = BCDASC(BCDVAR)

The BCD word which was preset by the AED compiler is passed to BCDASC as an argument. BCDASC creates a pointer to this word with a length of six (maximum number of BCD codes in one word) in the decrement. This pointer is passed to ASCITC which converts the codes to lower case ASCII. Again, as in LOCMES above, the pointer to the ASCII string is stored in the array named FREPTS for return of free-storage later by TYPEIT.

B. Procedures Calling BCDASC:

DROP, ERRGO, IFSRCH, LISFIL, LIST, LISTSL, MONTOR, QUIT, SUMOUT, USE

C. Procedures Called by BCDASC:

ASCITC

D. COMMON References:

None

E. Arguments:

BCDVAR: a BCD-coded integer

F. Values:

Ptr = ASCII Pointer

G. Error Codes:

None

H. Messages:

None

I. Length:

31_8 or 25_{10} words

J. Source:

TYPINT ALGOL

K. Files Referenced:

None

3.1.7.5 INDENT

Purpose:

To indent output

Description:

The print-out of data in the catalog fields provided by Intrex is desired in a presentable and controlled format. Part of this format control is the variable amount of indentation from the left margin. The convention presently employed is to indent the first line of a field two spaces and subsequent lines, if any, four spaces. The INDENT procedure is designed to accept an indentation setting (TAB) and communicate its value to TYPEIT by way of a commonly shared variable.

A. Operation: INDENT(M)

The procedure INDENT is compiled in the same file as the main typing control procedure TYPEIT, allowing it to share the same integer declarations. INDENT takes the contents of the argument M and copies it into the local parameter called TAB. If TAB is non-zero, the number it contains will be the number of spaces which TYPEIT will insert at the start of each new line of output (except the first line produced by the call, which will be two less). In this case, a local flag (DENT), also common to TYPEIT, will be set to TRUE.

If TAB is zero, the indentation mode is turned off by resetting DENT to FALSE.

B. Procedures Calling INDENT:

FSO, FSOCLN

C. Procedures Called by INDENT:

None

D. COMMON References:

None

E. Arguments:

M: integer

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

22_8 or 18_{10} words

J. Source:

TYPINT ALGOL

K. Files Referenced:

None

3.1.7.6 INCHAR

Purpose

To increment character counter (TYPEIT)

Description:

The procedure SPCTRN (Section 3.2.8.9) controls output of catalog fields on the special Intrex console and allows the display of special characters, such as Greek letters and superscripts and subscripts. This procedure must produce function codes for changing type fonts, etc. which are not output via TYPEIT. They are produced by calling a CTSS procedure named WRHIGH (Section 3.2.8.13). Since these codes are counted in the character counting logic of the Varian 620I, they must also be counted by the TYPEIT line control counter. INCHAR provides SPCTRN with the necessary access to TYPEIT's counter, CHARCT.

A. Operation: INCHAR(N)

The argument, N, is added to the local parameter, CHARCT, which is common with TYPEIT. A test of the new size of CHARCT is then made to determine if the maximum line length has been exceeded. If it has, a carriage return is put out via a call to the TYPEIT sub-procedure, PUTS. In this case, CHARCT is reset to the number found in the argument N.

If the DENT flag is set, meaning indentation is required after each carriage return, then the PUTS procedure is again used, this time to produce the required number of spaces of indentation. This number, specified by the parameter, TAB, is then added to CHARCT, which counts the accumulated total of characters in the line, and to OLDCNT, which contains the number of characters in this line previously put out in calls to TYPEIT which use the continuation (CONT) mode.

B. Procedures Calling INCHAR:

SPCTRN

C. Procedures Called by INCHAR:

PUTS

D. COMMON References:

None

E. Arguments:

N: integer

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

50_8 or 40_{10} words

J. Source:

TYPINT ALGOL

K. Files Referenced:

None

3.1.7.7 TYPASH

Purpose:

To type ASCII characters on console

Description:

The remote consoles used to communicate with the CTSS computer use either six-bit or twelve-bit BCD codes for printing. Since the main output procedure, TYPEIT, handles mostly ASCII data from the Intrex catalog, it constructs all of its message data in ASCII. TYPASH is used by TYPEIT (see Section 3.1.7.2) to convert the ASCII message characters to twelve-bit BCD and to transmit these characters.

A. Operation: TYPASH(ASCPTR)

The argument passed to TYPASH contains a pointer to an ASCII character string which begins at the address and byte position indicated by the address and tag of the pointer. The length of the string (in characters) is found in the decrement of this pointer and used to compute the number of computer words being employed to hold the string. If the length of the string is zero, TYPASH returns immediately to the calling program.

The amount of free storage needed to hold the converted 12-bit codes would normally b[illegible]umber of words used by the ASCII string. To allow room for han[illegible]ible untranslatable code[illegible] (which are printed as triplets of octal digits), an area twice the size of the input array is attempted to be obtained from free storage by calling the procedure FRER (Section 3.4.1.6). If FRER returns a zero value, that amount of free storage is not available. In this case, only the minimum amount of 4/3 the size of the input string is requested. If even this is unavailable, an error message (1) is printed via WFLX (Section 3.6.1.1) and processing is terminated by calling DORMNT.

If a free storage block is obtainable via the procedure FREE (Section 3.4.1.1), the address of that block is stored in the local parameters named OUTAD and OUT. The address just beyond the end of this output array is computed and saved in a parameter labelled ENDBUF.

The translation of codes is then performed by calling a procedure named TRASH (see next section), using the TYPASH argument ASCPTR and the pointer in OUT as arguments to TRASH.

Upon return from TRASH, the level two interrupt flag (INT2 of SST.) is examined to see if an interrupt has occurred since the previous test of this flag. If it has, TYPASH skips ahead to merely return the output area to free storage and return control to the calling program.

If no interrupt has called for a cessation of typing, the pointer in OUT, which now contains in its decrement the length of the converetd 12-bit string, is passed to the procedure PRT12 (Section 3.1.7.9) where the output of these codes will be finally accomplished.

The details of the code conversion and printing are described in the following two sections.

As each output string is printed, its free storage area is returned for repeated use by calling the procedure FRET (Section 3.4.1.2). TYPASH then returns to the calling program.

B. Procedures Calling TYPASH:

TYPEIT

C. Procedures Called by TYPASH:

FRER, FREE, FRET, TRASH, PRT12, WFLX, DORMNT

D. COMMON References:

None

E. Arguments:

ASCPTR: ASCII pointer

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "FREE STORAGE NOT AVAILABLE TO CONVERT OUTPUT FOR CONSOLE." (WFLX)

I. Length:

154_8 or 108_{10}

J. Source:

TYPASH ALGOL

K. File References:

None

3.1.7.8 TRASH, PUTOUT

Purpose:

To convert to BCD and store

Description:

These sub-procedures operate in conjunction with TYPASH, described in the preceding section, to perform the conversion from ASCII to 12-bit BCD codes and pack the converted codes into consecutive bytes of the output array.

A. Operation: TRASH(ASCPTR, OUT)

TRASH initializes itself by copying the pointers found in the arguments into local parameters so that they may be incremented without disturbing the original pointers. A character counter (to keep track of output codes) named CNT12 is reset to zero, and the number of characters to be converted is copied from the decrement of the first argument to a local parameter named NOCHAR.

NOCHAR is used to terminate a loop which takes each ASCII code from the input string (via GET and INC described in Section 3.4.4) and converts it to 12-bit BCD by calling a procedure named ASCTSS (Section 3.4.2.6). If the conversion is successful (not illegal code), then the BCD code is passed to a procedure named PUTOUT, along with the pointer to the output area.

PUTOUT checks the address of the current position of the [illegible]ut pointer against the end-of-buffer address computed and saved by TYPASH. If the pointer has reached this address, then this attempt at conversion is abandoned. The (insufficient) free storage array is returned by calling FRET and an attempt is made to increase the size of the output area by returning to TYPASH at the point where free storage is requested. If the end of the output area has not been reached, the 12-bit code is stored into it at the current byte position via two calls to PUT6 (Section 3.4.4.6) and two calls to INC6 (Section 3.4.4.5). Half of the 12-bit code is stored by each PUT6. (It was never considered worth-while to write a PUT12 procedure.) As each 12-bit code is stored, the counter CNT12 is incremented by one. The two calls to INC6 update the output pointer to the next available storage byte.

TRASH detects illegal codes in two ways. If the input (ASCII) is greater than octal 200, it is beyond ASCII range and ASCTSS is not even called. If the code returned by ASCTSS is a BCD "null" code and the ASCII input code was not a null, then the ASCII code has no corresponding BCD equivalent. In either of these cases, the ASCII code is put out in its octal form as three octal digits enclosed in angle brackets (e.g., <345>). This involves storing five 12-bit BCD codes via PUTOUT. In normal Intrex operation, this situation should never occur.

When all input codes, as governed by NOCHAR, have been processed, the number of output codes (in CNT12) is inserted into the decrement of the pointer to the output string. Any unused bytes in the last word of the output string are padded out with null codes (empty bytes will cause zeroes to be printed in BCD).

TRASH then returns to the calling program.

B. Procedures Calling TRASH:

TYPASH

C. Procedures Called by TRASH, PUTOUT:

ASCTSS, INC, GET, INC6, PUT6

D. COMMON References:

None

E. Arguments:

ASCPTR:	ASCII pointer	(TRASH)
OUT:	BCD pointer	(TRASH)
CHAR12:	12-bit code	(PUTOUT)
BUF12:	BCD pointer	(PUTOUT)

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

255_8 or 173_{10}

J. Source:

TYPASH ALGOL

K. File References:

None

3.1.7.9 PRT12

Purpose:

To print 12-bit characters

Description:

This is the lowest level typing control procedure written by Intrex personnel. It accepts a pointer to the output string converted by TRASH (as described in the previous section) and passes this output string in small groups of computer words to the basic CTSS printing procedure WRFLXA.

A. Operation: PRT12(OUTPUT)

The number of 12-bit BCD codes to be printed is contained in the decrement of the pointer found in the argument to PRT12. This number is used to compute the number of computer words utilized by this string of characters, which is packed with three codes to a word. The number of words is saved in a parameter, WORD12, which is common to the interrupt procedures contained in this same sour file (see Section 3.1.8.3).

Before beginning the printing process, the switch which controls the print-mode governing whether 6-bit or 12-bit codes will be interpretted is set to the 12-bit mode by a call to the CTSS procedure SETFUL (Section 3.5.5.5).

The string is printed in 28-word segments. After each call to WRFLXA (where characters finally are produced on the console), a check is made of INT2(SST.) to see if the interrupt button was [illegible] during the printing of that group of characters. If the interrupt flag is set, no further calls to WRFLXA are made.

Before returning to the calling program TYPASH, PRT12 calls SETBCD (Section 3.5.5.6) to switch the console back to the normal 6-bit mode and resets the parameter WORD12 to zero as an indication to the interrupt procedure INT TWO that no typing is now in progress (should an interrupt occur before the next call to PRT12).

B. Procedures Calling PRT12:

TRASH

C. Procedures Called by PRT12:

SETFUL, SETBCD, WRFLXA

D. COMMON References:

None

E. Arguments:

OUTPUT: 12-bit BCD pointer

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

114_8 or 80_{10}

J. Source:

TYPASH ALGOL

K. File References:

None

3.1.8 Interrupt Controls

3.1.8.1 ININT

Purpose:

To initialize interrupt controls

Description:

The CTSS software allows the programmer to control the flow of action when the system detects that a user has pressed the ATTN or BREAK key (sometimes referred to in this documentation as the "interrupt button"). Intrex uses two interrupt procedures to direct control of the Intrex system. These two procedures correspond to two "interrupt levels". Level one is defined to be anywhere in the Intrex system except within TYPEIT. Level two is defined as anywhere within the scope of TYPEIT or the procedures called by TYPEIT.

ININT is called during the session initialization phase of Intrex operation (DYNAMO, described in Section 3.1.3.1) to set the interrupt controls of CTSS to transfer to the level one procedure, INTONE, if the user presses the ATTN key.

A. Operation: ININT( )

The address of the first instruction of the procedure INTONE (next section) is placed in the POT in a location labeled INT1AD(POT.) where it will be available to any module of the Intrex system. This address is then passed as an argument to the CTSS procedure SETBRK (Section 3.5.7.5) which will route logical flow to this address in the event of an interrupt.

B. Procedures Calling ININT:

DYNAMO

C. Procedures Called by ININT:

SETBRK

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
INT1AD(POT.)	INTONE address	x	x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

21_8 or 17_{10} words

J. Source:

TYPASH ALGOL

K. File References:

None

3.1.8.2 INTONE

Purpose:

To control interrupt level one

Description:

The address at which Intrex was operating when the interrupt was detected by CTSS is obtained and returned to after a flag is set in the System State Table to indicate that an interrupt has occurred. No further action is taken on the interrupt until the flag is detected upon the next call to TYPEIT.

A. Operation: INTONE( )

Three important machine conditions are immediately saved upon entering INTONE via the CTSS interrupt mechanism. They are: the contents of the accumulator, the contents of index register 1 (used in index-modified addressing), and the contents of index register 4 (used in sub-routine linkage).

Because of the time lag between the CPU's execution of output commands and the remote terminal's actuation of that output, it is possible for output (the main concern here is the printing of the READY message) to be processed by the CPU but not yet printed by the console when the interrupt occurs. To circumvent this problem, a flag, GETFLG(SST.), is set upon entering the procedure GETLIN (which produces the READY message just before waiting for input from the user). If INTONE finds this flag set, control is sent back to the procedure LISTEN in the Intrex supervisor. This causes a new call to the Command Language Processor (CLP) and a repeat of the READY message.

If GETFLG(SST.) is not found to be set upon reaching INTONE, the flag INT1(SST.) is set to TRUE.

The address of the point of interruption is obtained by calling the CTSS procedure GETBRK (Section 3.5.7.4) which returns this address as a value. INTONE stores this address in a local parameter labeled RETAD. Since the transfer of control must be made "indirectly" through RETAD, and since AED provides no means of indirect transfer except through procedure calls, the "machi" facility of AED is used to implement the transfer in machine language.

Before leaving INTONE (either to LISTEN or to the point of interruption the interrupt level, which was dropped back to zero automatically when the ATTN key was pressed, must be raised again to one by a call to SETBRK.

Before returning to the point of interruption, INTONE restores the index registers and accumulator saved upon entrance to the procedure.

B. Procedures Calling INTONE:

None (called by CTSS interrupt logic)

C. Procedures Called by INTONE:

SETBRK, LISTEN, GETBRK

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
GETFLG(SST.)	GETLIN Flag	x	
INT1AD(POT.)	Inter. level one address	x	
INT1(SST.)	Inter. one flag		

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

54_8 or 44_{10} words

J. Source:

TYPASH ALGOL

K. File References:

None

3.1.8.3 INTTWO

Purpose:

To control interrupt level two

Description:

As in interrupt level one, control is returned to the point of interruption after a flag is set by INTTWO to indicate a level two interrupt. This can occur only when a TYPEIT call is being processed when the ATTN key is pressed. There are several points within TYPEIT where the flag set by INTTWO is tested to determine if processing of output should be halted

A. Operation: INTTWO( )

As in INTONE, the accumulator and index registers one and four are saved for restoration when INTTWO is ready to return. The flag, INT2(SST.), is set to TRUE and the address of the interrupt is obtained via GETBRK and saved in RETAD.

A parameter called WORD12, which is shared with PRT12 (Section 3.1.7.9) is examined to see if the interrupt occurred during the execution of PRT12, where the output buffer is actually written via RDFLXA. If WORD12 is not zero, it contains the number of computer words of buffer space currently being written. These words are then filled with null codes by INTTWO so that no further characters will be printed when INTTWO returns to PRT12.

The accumulator and index registers are restored and INTTWO transfers "indirectly" through RETAD. This is done using machine language instructions via the "machi" feature of AED.

B. Procedures Calling INTTWO:

None (called by CTSS interrupt logic)

C. Procedures Called by INTTWO:

GETBRK

D. COMMON References

Name	Meaning	Interrogated?	Changed?
INT2(SST.)	inter. two flag		x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

50_8 or 40_{10} words (approx.)

J. Source:

TYPASH ALGOL

K. File References:

None

3.1.8.4 LISTEN

To provide a Supervisor entry point

Description:

A. Operation: LISTEN( )

The procedure LISTEN acts as a switching station within the Intrex supervisor, directing the flow of processing according to system conditions at the time LISTEN is called.

If the ISI(SST.) flag is set, indicating that Intrex is in "sign-in" mode, control is transferred to that early section of SUPER where the procedure SIGNIN is called to ask the user to identify himself. This has the effect of causing the sign-in request to be repeated if the user interrupts it before it is finished.

If the flag GETFLG(SST.) is set, this means that LISTEN was called by the procedure INTONE and that the interrupt took place after GETLIN had called READY (see 3.2.1.3 and 3.1.8.2). This is somewhat of a special case in which the usual message about the interrupt is not written into the Monitor File by TYPEIT. It must, therefore, be written by LISTEN. To accomplish this, LISTEN calls GETBRK (to obtain the address of the interrupt point), OCTASC (to convert the address to octal ASCII digits), ASIDE (to write the message, "interrupt at level 1 at location . . . . . . "), and MONTIM (to write the timing message).

If GETFLG(SST.) is FALSE, instead of the above procedure calls, FSOCLN is called to reset any conditions left by an interrupted call to FSO or GETFLD.

If IBEG(SST.) is set, then the user is in the pre-sign-in stage of Intrex where he is being asked to type the word BEGIN. An interrupt at this stage will cause LISTEN to loop back to the point in SUPER where the user is again asked to type BEGIN.

If none of the above flags are set, LISTEN transfers to the point in SUPER where the command Language Processor (CLP) is called to prepare to accept the user's next command.

B. Procedures Calling LISTEN:

INTONE, TYPEIT

C. Procedures Called by LISTEN:

GETBRK, OCTASC, ASIDE, MONTIM, LOCMES, FOSCLN (via CALLIT)

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
ISI(SST.)	In sign-in flag	x	
IBEG(SST.)	In begin flag	x	
GETFLG(SST.)	In-GETLIN flag	x	
STM.(POT.)	System time array		x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

130_8 or 88_{10} words

J. Source:

SUPER ALGOL

K. File References:

None

3.1.9 Overlay Controls

3.1.9.1 SYSGEN

Purpose

To generate and initialize overlay system segments

Description:

SYSGEN is called by SUPER to initialize an overlay segment. SYSGEN updates the overlay table, (sysnam) .TBLE., initializes a segment and writes it out as (sysnam) SGMTOn.

A. Operation SYSGEN( )

See explanation of overlay mechanism (Section 2.5)

B. Procedures Calling SYSGEN:

SUPER

C. Procedures Called by SYSGEN:

BCDEC	FRALG	GETCOM	RJUST
BUFFER	FREE	INITDB	SEGINT
CHFILE	FRESET	LINKUP	SETMEM
CHNCOM	FRET	OCABC	SENTRY
CLOSE	FERRTN	OPEN	WRFLX
DELFIL	FCLEAN	RDFLX	WRFLXA
DORMNT	FSTATE	RDWAIT	WRWAIT

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

"Type system name"	(WRFLXA)
"New segment or replacement"	"
"Type number of segment to be replaced"	"
"Is it a new system"	"

".tble. file not found"	(WRFLXA)
"Old .tble. is too short"	"
"Segment not in segment table"	"
"addr1 Linkup new Addr2"	"
"Proc not found in system directory"	"
"Segment exceeds old memory bound"	"
"Main must be re-initialized"	"
"Error reading (MOVIE TABLE)"	"
"Linkup not found in (MOVIE TABLE)"	"
"SEGMT0n starts at Addr1 ends at Addr2"	"
"D base is now Addr1 Sysnam is dormnt"	"

I. Length:

770_8 or 504_{10} words

J. Source:

SYSNEW FAP

K. Files Referenced:

sysnam	.TBLE.
sysnam	SGMT01
sysnam	SGMT02
sysnam	SGMT03
sysnam	SGMT04
(MOVIE TABLE)	

3.1.9.2 CALLIT

Purpose:

To call procedures in overlay segments

Description:

CALLIT is used to transfer control from a procedure in the main body to one in a segment. It can still be used if the called procedure is moved to the core-resident portion of the system.

A. Operation: Value = CALLIT (.BCD./PROC/, Arg1, Arg2)

Procedures in overlay segments that are called via CALLIT

ATLCLN	DYNAMO	INFO	LISTSL	RANGE	SIGN2
AUTHOR	EXIT	INIFIX	LONG	SAVE	TABLE
BUFSCN	FSO	INI	MONTOR	SEARCH	TIME
CHKSAV	FSOCLN	INIVAR	OPFILE	SEEMAT	
COMENT	GETLIN	LIBRY	PREP	SHORT	
CONDIR	GO	LISFIL	QUIT	SIGNIN	

Procedures in the core-resident portion of the system that are called via CALLIT

AND.	FCLEAN	OR.
ASCINT	IN.	OUT.
CHKNUM	IOUT	RESTOR
CLEANP	LIST	SUBJ.
DROP	NAME	TITLE
EVAL	NUMBER	USE
FAPDBG	NOT	WITH.

B. Procedures Calling CALLIT

SUPER (Proc = INIFIX, INIVAR, DYNAMO, SIGNIN, FSO, EVAL, SEARCH, SEEMAT)
LISTEN (Proc = FSOCLN)
INIFIX (Proc = PREP, TABLE)
CLP (Proc = GETLIN, any command interpretor)
NUMBER (Proc = CLEANP)
ANDER (Proc = BUFSCN)
RESTOR (Proc = CLEANP)
DROP (Proc = CHKSAV, CONDIR)
LIST (Proc = CHKSAV, LISFIL, LISTSL)
USE (Proc = CHKSAV, MOVEIT, CONDIR)

C. Procedures Call[illegible] by CALLIT:

LINKUP, MONTIM, RDWAIT, WRFLX, WRFLXA

D. COMMON Refer[illegible]es:

Name	Meaning	Interrogated?	Changed?
STM.(POT.)	System Time Monitor	x	

E. Arguments:

.BCD. PROC : procedure name expressed in 6-bit BCD

Arg1. Arg 2: argument of call to procedure PROC

F. Values:

Value = value returned (if any) by PROC

G. Error Codes:

None

H. Messages:

1. "PROC not in table" (preset)
2. "PROC is not linked" (preset)

I. Length:

430_8 or 280_{10} words

J. Source:

SYSGEN FAP or SYSNEW FAP

K. Files Referenced:

(sysnam)	SGMT01
(sysnam)	SGMT02
(sysnam)	SGMT03
(sysnam)	SGMT04

3.1.4.5 SENTRY

Purpose:

To allow entry into segment

Description:

A. Operation: SENTRY( )

SENTRY is used in the overlay initialization phase to obtain the names of the entry points within a particular overlay segment. In the execution phase, CALLIT passes to SENTRY the name of a procedure, to which SENTRY passes control.

B. Procedures Calling SENTRY:

LINKUP

C. Procedures Called by SENTRY:

WRFLX

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "PROC not found for segment n" (preset)

I. Length:

40 - 80 words

J. Source:

nSENT FAP

K. Files Referenced:

None

[illegible].1.4.4 MAINBD

Purpose:

To give main body boundary

Description

A. Operation: Loc MAINBD( )

MAINBD returns as its value the address of the beginning of the overlay area. This value is used by .C.ASC and FRALG to determine if an area may be returned to free storage.

B. Procedures Calling MAINBD:

FRALG, .C.ASC

C. Procedures Called by MAINBD:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

Loc = beginning location of segment area, in binary

G. Error Codes:

None

H. Messages:

None

I. Length:

6 words

J. Source:

SYSGEN FAP, SYSNEW FAP

K. Files Referenced:

None

5.1.4.8 LINKUP

Purpose:

To provide segment entry point

Description

A. Operation: LINKUP( )

LINKUP is the entry point into an overlay segment. Since it is always the first routine in a segment and since it is identified in all segments, the location of its entry point is the same in each segment.

B. Procedures Calling LINKUP:

CALLIT

C. Procedures Called by LINKUP:

SENTRY

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H Messages:

None

I. Length:

1 word

J. Source:

LINKUP FAP

K. Files Referenced:

None

3.1.14 Error Controls

3.1.14.1 ERRGO

Purpose:

To provide a centralized error exit

Description:

ERRGO allows access from any point in Intrex to the standard error exit to which most I/O errors are routed automatically. Those which are not automatically sent there by the CTSS error logic (as set up during initialization of Intrex by a call to FERRTN, described in Section 3.5.4.1) may call ERRGO. This provides the opportunity to print error diagnostic data and close the Monitor File. (Note: ERRGO is internal to INIRES, described in Section 3.3.2.1).

A. Operation: ERRGO ( )

The first program statement of ERRGO is labeled EREXIT. This label is used as the error condition-transfer point for any I/O error produced by a procedure call which does not specify local error returns. This first statement is a call to IODIAG, a CTSS utility (Section 3.5.4.2) which stores various data about the I/O error in a local array. This data is then typed and recorded in the Monitor File by calling TYPEIT with the various elements of data as arguments. This produces a message of the form shown in Part H.

Having recorded the error data, the Monitor File is closed and DORMNT is called to return control to CTSS level. If Intrex is in the HOLD mode, the subsystem, INXSUB (Section 3.1.10.3) will then take control.

B. Procedures Calling ERRGO:

IFSRCH

C. Procedures Called by ERRGO:

IODIAG, TYPEIT, BCDASC, OCTASC, BFCLOS, DORMNT

D. COMMON References:

None

Arguments:

None

Values:

None

G. Error Codes:

None

H. Messages:

1. "Error Code^{1} N^{2} in call to^{3} ROUT^{4} at location^{5} AAAAA^{6} involving file^{7}, NAME1 NAME2^{8}"

<u>Notes</u>:

(1) $IOERR1$
(2) Code found in IODIAG array word 3
(3) $IOERR2$
(4) Procedure name found in IODIAG array word 2
(5) $IOERR3$
(6) Address found in IODIAG array word 1
(7) $IOERR4$
(8) File names found in IODIAG array words 5 and 6

Length:

102_8 or 66_{10} words (included in INIRES)

J. Source:

RESLIS ALGOL

K. Files Referenced:

Monitor File

3.1.10.2 SETRTN

Purpose

To set up error return address

Description:

The CTSS utility procedure FERRTN (Section 3.5.5.1) allows specification of a standard or central I/O error return. The format of the call to FERRTN, however, requires that the argument specifying the labeled address to be used as the error return be of the form PZE LABEL rather than TXH LABEL. This necessitates writing the calling sequence in machine language rather than in AED. SETRTN is called from the AED-compiled procedure INIRES (Section 3.3.2.1) and in turn calls FERRTN with the proper machine instructions.

A. Operation: SETRTN(EREXIT)

SETRTN is compiled as an AED procedure (in an ALGOL file) but consists of only two machine language instructions using the "machi" facility of AED.

The two machine instructions are:

```
TSX  FERRTN, 4
PZE  EREXIT
```

The AED compiler sets up the necessary instructions for inserting the proper argument address into the PZE location, and for generating the return transfer instructions.

B. Procedures Calling SETRTN:

INIRES

C. Procedures Called by SETRTN:

FERRTN

D. COMMON References:

None

E. Arguments:

EREXIT: Label

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

11_8 or 9_{10} WORDS

J. Source:

SETRTN ALGOL

K. File References:

None

3.1.10.3 INXSUB

Purpose

To control recycling of Intrex

Description:

Intrex tries to prevent the user from reaching CTSS command level as each session is concluded. By immediately and automatically restarting Intrex, users are prevented from using CTSS for other operations and are relieved from resuming Intrex to start each session.

Intrex also tries to recover automatically from error exits and refuses to allow the user to quit via the ATTN key and issue other CTSS commands (except for a select few such as RSTART).

This control over the user's activities is made possible by the CTSS subsystem logic. This system allows the specification of a given SAVED program file to be used by CTSS whenever normal program operation is terminated by certain conditions, such as: program stop, call to DORMNT, call to CHNCOM, or I/O error.

When any of these conditions are encountered, CTSS will then automatically initiate execution of the specified subsystem— in this case, INXSUB. The subsystem then examines a condition code to determine the reason it was called, and takes the appropriate action. Usually this action involves an automatic resumption of Intrex, though not necessarily at the Log-in stage.

A. Operation:

1. The first step of INXSUB is to turn off the "blip" feature (in case a search was being performed at the time of an Intrex failure. This is done by calling SETBLP (Section 3.5.7.12) with arguments of zero.

2. INXCON is called (Section 3.1.3.2) to determine if an Intrex console is being used. If so, the line length must be re-set to the shorter 55 character length limitation of the Intrex console. This is done by calling the procedure SETLIN (a sub-procedure of a general-purpose version of TYPEIT designed for non-Intrex use) which changes the line length used by TYPEIT when it produces messages to the user from the subsystem.

3. A local array is filled with identification data by calling the CTSS utility procedure WHOAMI (Section 3.5.7.9).

4. The condition code word is obtained from CTSS by calling GETSYS (Section 3.5.7.18). Another special A-core word is obtained by calling GETWRD (Section 3.5.7.11). This word is used to communicate one of several possible pieces of information (to be discussed later) from Intrex to the subsystem.

5. The left-most eighteen bits of the condition code are extracted and examined. The individual bits of this half-word will indicate which condition caused the subsystem to be employed. A 1 in this half-word indicates that the subsystem was trapped by the issuance of a new CTSS command, meaning that the user quit via the ATTN key and tried to use a CTSS command other than RSTART. In this case, TYPEIT is called to print a message (1) telling the user what he did and that Intrex is being resumed at the point of interruption. A core-image of Intrex at the point where the ATTN key was pressed is automatically SAVED by CTSS before the subsystem is called. This core image is given a first name consisting of the programmer number. The command buffer is then set to contain the programmer number obtained from WHOAMI as a RESUME command argument. When the core image file is resumed, the user continues where he was before he quit.

6. Before testing for other conditions which might have called INXSUB into action, the A-core word mentioned above is examined to see if it is the word "STOP". This word is inserted by Intrex before execution of certain vital I/O operations. If one of these operations triggers an error, it is fruitless for the subsystem to attempt to restart Intrex. Therefore, if this word is found, the subsystem prints a message (2) telling the user that Intrex is unable to continue and that he should inform Intrex personnel of the error condition. The core-image of Intrex, saved by the CTSS logic as -PROGNO[1*]-SAVED, is renamed BOMB -time- and given a permanent mode via CHFILE (Section 3.5.3.1) for future post-mortems. The time of day, obtained via GETTM (Section 3.5.7.23) is used as the last

* See footnotes at end of section.

name of the permanent core-image. The option bits which control CTSS's communication with INXSUB are set to zero via LDOPT (Section 3.5.7.16) and DORMNT is called to return to CTSS command level.

7. If the A-core word is not "STOP", it is assumed to be either the first name of a Password File or the dialog mode (LONG or SHORT). If the Intrex system was started up in the HOLD mode (see Sections 3.1.3.1 and 3.3.2.1), the holding password and the dialog mode will be contained in a Password File whose name has been obtained by INXSUB via the above-mentioned call to GETWRD. This mode is detected by calling the procedure FILCNT (Section 3.4.3.1) with the A-core word as the first argument and FILE as the second argument. If a file by the name of -word- FILE exists, the value returned from FILCNT will be the length of that file. If no such file exists, the value returned is zero. If the latter is the case, HOLD mode is not in use so the A-core word is assumed to be the dialog mode and INXSUB jumps ahead to the system-error-alert logic mentioned below (Paragraph 9).

8. If the file exists, it is opened for buffered reading using the CTSS utilities OPEN (Section 3.5.1.1) and BUFFER (Section 3.5.1.3). The first word of the file is read into a local password location, and the second word is read into a mode location. The file is then closed via CLOSE (Section 3.1.5.2). If an I/O error occurs while reading the file, an error message (4) is printed and DORMNT is called.

9. If the Password File is only two words long, this indicates that the subsystem was called because of an error condition within Intrex. In this case, INXSUB prints a message (3) to the user informing him of a system error. Since the STOP was not found in the A-core word, this error is probably of a less critical nature. INXSUB, therefore, resumes Intrex from the beginning in hopes that a fresh start will allow the user to continue his session. The command buffer of CTSS is set up to contain the command arguments; INTREX, SKIP, -MODE-^{2}, HOLD^3, PWORD-^{4}. Here again, the core-image is renamed and and saved for post-mortems by calling GETTM and CHFILE as described above.

10. If the Password File contains three rather than two words, the third word was written by the QUIT command logic of Intrex. This means that Intrex has not encountered an error condition but is merely re-cycling. In this case, the CTSS command buffer is set up to contain the arguments; INTREX, LONG, HOLD[3] -PWORD-[4] BEGIN.

11. The BEGIN argument is included only if the condition code contains a 4 (4 is CHNCOM call indication). A subsystem trap caused by calling CHNCOM at the completion of the Intrex QUIT routine (Section 3.1.4.4) indicates that the user typed BEGIN during an Intrex session (without a prior QUIT). When this occurs, QUIT is called internally and automatically (from GO, Section 3.1.4.1) and ends with a call to CHNCOM. A trap caused by CHNCOM forces the BEGIN argument to be included in the "resume Intrex" command, which skips the request for the user to type BEGIN and goes right to the LOGIN request (see Sections 3.1.3.1 and 3.1.1.1).

12. If a call to DORMNT (the usual exit from the QUIT procedure) is the cause of the subsystem being called, the BEGIN argument is omitted from the "resume Intrex" command so that the user will be asked to type BEGIN to start the next session.

13. In all cases of Intrex being resumed (either continued or restarted), one central exit route is taken from INXSUB (labeled GO). At this area, the subsystem option bit (40 octal), which indicates that the present operating program is the subsystem, is reset to zero, (This bit is set by CTSS just before starting to execute the subsystem. It prevents recursive subsystem calls.) Finally, the previously prepared command buffer is activated by calling the CTSS procedure SCLS (Section 3.5.7.24), and the resume Intrex command is executed by calling the procedure NCOM (Section 3.5.7.25).

B. Procedures Calling INXSUB:

Activated by CTSS subsystem Logic

C. Procedures Called by INXSUB:

SETBLP, INXCON, SETLIN, GETSYS, GETWRD.
WHOAMI, TYPEIT,* LOCMES,* LDOPT,
DORMNT, FILCNT, GETTM, CHFILE, OPEN,
RDWAIT, BUFFER, CLOSE, RSOPT, SCLS, NCOM

*General purpose version

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "You have left Intrex by hitting the ATTN button twice. Intrex will now continue at the point of interruption." (LOCMES)

2. "Intrex is unable to continue because of a system error. Please bring this to the attention of Intrex personnel." (LOCMES)

3. "Intrex has experienced a system error. Your search status is being reset in hopes that you may be able to proceed as if you had just logged in to Intrex. If you encounter further difficulty in system operation, please notify Intrex personnel. In any case, please bring this error to our attention at your convenience. Wait for READY message." (LOCMES)

4. "Error reading pass file in Subsystem." (LOCMES)

I. Length: (Separate program, not part of Intrex)

332_8 or 218_{10} words (main program only).

17640_8 or 8096_{10} words (including all sub-procedures)

J. Source:

INXSUB ALGOL

K. File References:

Password File
INTREX SAVED
progno SAVED

Footnotes - (Part A)

1. -PROGNO- represents the programmer number logged in to that console as obtained via WHOAMI.

2. -MODE- is the dialog mode, LONG or SHORT, as obtained from the Password File.

3. HOLD is the argument to a "resume Intrex" command which causes automatic re-cycling of Intrex through the sub-system when the user issues a QUIT command.

4. -PWORD- is the password which prevents recycling after a QUIT command, as obtained from the Password File.

3.2 Command Control Logic

3.2.1 General Command Processing

3.2.1.1 INICON

Purpose

To set up delimiter and trim tables.

Description

A. Operation: INICON( )

INICON initializes CLP by setting certain variables and by generating a character string via .C.ASC. FRALG returns the space used by the procedure to free storage.

B. Procedures calling INICON:

INIFIX

C. Procedures called by INICON:

FRALG, .C.ASC

D. COMMON references:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

17_8 or 15_{10} words

J. Source:

CLP ALGOL

K. Files Referenced:

None

3.2.1.2 CLP

Purpose

To process command lines.

Description

When the system is ready to accept commands from the user, SUPER calls CLP. CLP functions as a second-level supervisor over the process of accepting and interpreting user input.

A. Operation: Code = CLP( )

CLP calls GETLIN, which accepts a command line from a user and returns an ASCII pointer to that line. CLP gives this pointer to NEXITM, which returns a pointer to the first substring in the command line (substrings are set off by the characters colon, space, slash or carriage return). If NEXITM cannot find a substring — for instance, if CLP has processed all the substrings — CLP returns control to SUPER. Otherwise, CLP gives the substring to LOOKUP, which searches the command table. If it finds a match, it returns the BCD name of the module associated with the command. If LOOKUP reports that the substring is not a command name, CLP asks RESTOR to check the string against the list of list names. If the string is not a list name either, CLP prints "illegal command" and returns to SUPER. If the substring is a command, CLP gives CALLIT the name of the required subroutine — as supplied by LOOKUP — and CALLIT transfers control to the routine. CLP then calls NEXITM again to search for the next command.

B. Procedures Calling CLP:

SUPER, SIGNIN

C. Procedures Called By CLP:

1. Direct Calls:

CALLIT, DORMNT, FRET, GETLIN, LOOKUP, NEXITM, RESTOR, TYPEIT, WRFLXA

2. Calls via CALLIT

AND.	IN.	NOT.	SHORT
AUTHOR	INFO	OR.	LONG
COMENT	LIBRY	OUT.	SUBJ.
DROP	LIST	QUIT	TIME
EVAL	MONTOR	RANGE	TITLE
EXIT	NAME	SAVE	USE
GO	NUMBER	SEEMAT	WITH.

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
IBEG (SST.)	In begin stage	x	
MODEG (POT.)	Anding mode		x
TEXTX(POT.)	Text pointer		x
COMTB.(POT.)	Command table	x	
VERBOS (POT.)	Long/short mode	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes

- -1: user has not typed begin or unintelligible command (message 1)
- -2: command line is too long (message 2)
- -3: illegal command (message 3)
- -4: error return from command interpreter

H. Messages

1. "System error attempting to interpret command line." ($CLP1)

2. "Please break your request into requests not exceeding 200 characters in length." ($CLP2$)

3. "X is not a legal command name. Check for typing errors. See part 6.2 of Guide for full list of commands. ($CLP3$)

   "Please rephrase your request."

4. "Your command could not be understood." ($CLP4$)

I. Length:

320_8 or 208_{10} words

J. Source:

CLP ALGOL

K. Files Referenced:

None

3.2.1.5 GETLIN

Purpose

To get a command line.

Description

CLP calls GETLIN to obtain a string of user commands. GETLIN waits for input from the user in the wait state, waking up periodically so that the console is not automatically logged out. When a user types a character, the system is automatically forced out of the wait state, and GETLIN receives the user's command line. GETLIN converts this line from 12-bit CTSS code into 9-bit ASCII code. GETLIN returns an ASCII pointer to this string.

A. Operation Ptr = GETLIN( )

1. Main Flow

GETLIN initializes itself by allocating a fixed amount of storage for both the original 12-bit string and the converted 9-bit string. GETLIN calls GETIME to get the current time of day and then falls asleep for a pre-determined amount of time. When it wakes up it checks the time and computes how long it has been asleep. It then uses FILCNT to check for the existence of either the disk file "Hold Up" or "Hold It". If neither of these files exist (more about this in Section 2), it compares the amount of time that it has been asleep against the specified sleep time. If these times differ, GETLIN assumes that it was forced out of wait state by the user and RDFLXA is called to receive the user's command line. RDFLXA returns control to GETLIN when the user types a carriage return. GETLIN examines each 12-bit character in the command line, converts it to ASCII and stores it in the ASCII string area. If the line ends with a continuation character - the hyphen -GETLIN calls RDFLXA again and waits for more input. Otherwise, GETLIN inserts a slash at the end of the line, constructs an ASCII pointer to the converted line, and returns control to CLP.

2. Minor Details

(1) The presence of a file HOLD UP signifies that a systems programmer is updating the message files. HOLD IT is used for updating segments. In each case, the appropriate files are closed and the system goes to sleep. After 5 minutes GETLIN will resumed normal operation.

(2) Since CTSS accepts any character in a 12-bit mode, GETLIN must itself implement the CTSS kill and erase feature.

(3) GETLIN writes the input line in the Monitor File via ASIDE. If the system is in CATII mode, the input line is written into CATII output.

B. Procedures Calling GETLIN:

CLP, SIGNIN

C. Procedures Called by GETLIN:

ASIDE, CLFILE, FRET, FREZ, GET, GETIME, GET12, FILCNT, INC, INC12, INITYP, INIDSK, NAP, OPFILE, PUT, RDFLXA, SETBCD, TSSASC, WAIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMLIN(POT.)	Command line buffer	x	
MAXLIN(POT.)	Max. lines input	x	
VERBOS(POT.)	Long/short mode	x	
DFLN1(POT.	Dialog file long	x	
DFSN1(POT.)	Dialog file-short	x	
TEXTX(POT.)	Text pointer	x	
RTM. (POT.)	Ready message time pointer	x	
GETFLG(SST.)	Interrupt in GETLIN	x	x
HELD(SST.)	Hold for DIRGEN	x	x
CATII(SST.)	Off-line output	x	
TIMES(SST.)	Times requested	x	
INFOX(SST.)	Info. requested	x	

E. Arguments:

None

F. Values:

Ptr = ASCII pointer to command line

G. Error Codes:

Ptr = 0: command line is empty
-1: command line is too long

H. Messages:

1. "READY" (/Redmes/) ["R" -short mode]

I. Length:

770_8 or 504_{10} words

J. Source:

GETLIN ALGOL

K. Files Referenced:

None

3.2.1.4 NEXITM

Purpose

To parse character strings

Description

The procedure NEXITM is designed to find the "next item" in a string. A table of delimiters is used to determine what NEXITM will return as the next substring in a string of ASCII characters. Trimming of leading and terminating characters and culling of common words is also possible.

A. Operation:

The normal call to NEXITM contains a pointer to the string, a pointer to a table of delimiters and an argument which will contain the terminator found upon returning.

next = NEXITM (staptr, termfo, tertap) $,

staptr is a standard Intrex pointer to the string of ASCII characters.

termfo is the character found that NEXITM used to delineate this substring.

tertap is a pointer to the delimiter table.

next is a pointer to the substring of characters starting at the beginning of the string (at the location pointed to by staptr) and ended by but not including the character in termfo.

The number of characters in this substring is in the decrement of next, and the length of the remaining string is computed and staptr is updated to point to the part of the string that follows the substring returned.

Four optional arguments may be used. They must appear in the following fixed order:

next = NEXITM (staptr, termfo, tertap, frtrim, endtrim, cultab, noast)$

staptr, termfo, and tertap are as above, and the four optional arguments are:

frtrim is a pointer to a table of characters that will be trimmed off at the beginning of the substring.

endtrim is a pointer to a table of characters that will be trimmed at the end of the substring.

cultab is a pointer to a table of common words that will be culled out if they appear as the next item.

noast will disable the standard Intrex catalog convention for asterisks as special characters. If this argument is not present any asterisks that appear will be treated as pairs that enclose a group that is not to be broken up. (See "Input/output Representations of Special Characters", Intrex Memorandum 4). This is a boolean argument, and does not pass any value to NEXITM, but is merely present or not present.

The use of an optional argument that follows some unused optional argument requires the unused argument to be given as -0. If noast is present (with any value), asterisks will be treated as ordinary delimiters.

The delimiter, front trim, and end trim tables are ASCII characters packed 4 per word. The pointers to these tables are standard Intrex pointers.

The cull table and the cull table pointer have a special construction. The pointer cultab has the table length (in computer words) in the decrement. The address portion contains the address of the beginning of the table. The table consists of English words of 4 or less letters. The ASCII codes for the words are left justified in the computer word, and unused bytes are filled as zeroes. The last computer word in the table is all octal 7's. The table searching procedure used is a FAP coded procedure that uses a 2 instruction loop. This procedure (TBSRCH) is available along with a modified version for a table with more than one computer word per entry (VSRCH). The current table for culling common words contains the 13 English words used in Inverted File generation. These are: a, by, as, at, in, of, on, to, and, for, the, with, from.

B. Procedures Calling NEXITM:

AND.	IN.	OUT.	S.T
AUTHOR	LIST	QUIT	TABLE
CLP	MONTOR	RANGE	TIME
DROP	NAME	SAVE	USE
INFO	NUMBER	SIGNIN	WRT

C. Procedures Called by NEXITM:

COPY	FREZ	ISARG
DEC1	GET	ISARGV
FIND	INC	PUT
FRET	INC 1	TBSRCH

D. COMMON References:

None

E. Arguments:

See Section A

F. Values:

ASCII pointer

G. Error Codes

next = 0 and termfo = 0 if an empty string pointer has been used. (i.e. the decrement of staptr is zero.)

next = -1 if no delimiter is found in the string.

next = -2 and termfo = 52_8 if only one asterisk is found and the argument noast is not present.

next = 0 and termfo =-delimiter found - if there are no characters remaining in the substring after front and end trimming.

H. Messages:

None

I. Length:

1000_8 or 512_{10} words

J. Source:

NEXITM ALGOL

K. Files Referenced:

None

3.2.1.5 LOOKUP

Purpose

To identify a command.

Description

A. Operation Bcdstr = LOOKUP (Ascptr, Tabptr)

LOOKUP is used by CLP to associate a user command with the Intrex program module which will act on it. CLP gives two arguments to LOOKUP, an ASCII pointer to a word from the user's command line and a pointer, found in COMTB.(POT.) to the Intrex command table. LOOKUP takes the first four characters of the ASCII string and uses VSRCH to find them in the table. If they are found, LOOKUP extracts the word which follows these matching characters from the table. This is the BCD name of the routine which CLP will call (via CALLIT) so that the user's command may be carried out. LOOKUP returns to CLP with this BCD string as its value. If the four character ASCII string was not found, LOOKUP returns to CLP with a value of 0.

B. Procedures Calling LOOKUP:

CLP

C. Procedures Called by LOOKUP:

GET, PUT, INC1, VSRCH

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMTB.(POT.)	Command table pointer	x	

E. Arguments:

Ascptr:	ASCII pointer to user command
Tabptr:	pointer to command table

F. Values:

Bcdstr = name of procedure associated with command, in BCD

G. Error Codes:

0: command not found

H. Messages:

None

I. Length:

160_8 or 112_{10} words

J. Source:

NEXITM ALGOL

K. Files Referenced:

None

3.2.2 Subject/Title Command Interpretation

3.2.2.1 INIS.T

Purpose

To initialize subject/title interpreter.

Description

A. Operation:

INIS.T is called by INIT2, which is called by INIFIX, during the fixed parameter initialization phase. It sets up strings of delimiters and pointers to these strings for NEXITM to use in extracting the command line words. It is given over to free storage via FRALG after execution.

B. Procedures Calling INIS.T:

INIT2

C. Procedures Called by INIS.T:

FRALG, FREZ, .C.ASC

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

43_8 or 35_{10} words

J. Source:

INTPRT ALGOL

K. Files Referenced:

None

3.2.2.2 SUBJ.

Purpose

To interpret SUBJECT commands

Description

A. Operation: Code = SUBJ.(Staptr)

The first task undertaken by this routine is to clean up any search structures (pointers, counts, indicators) (see Fig. 3.1) left from the previous search. This is accomplished by calling one of two procedures. If the current list is the result of a search, then CLEANP (in a source file of the same name) is called. If the current list is a restored NAMED list, then less clean-up is necessary and the little that is required is done by calling DELIST (residing in RESLIS). The choice of which routine to call is made by testing the system state indicator, RLIC(SST.), which is set when a restored list is in core.

SUBJ. then resets an internal flag called TIT, which is used to tell the sub-procedure S.T. whether it is processing a subject or a title command. A "subject searchform" array of six words is obtained from free storage and the pointer placed in the command list (SSF.(CL)). Then S.T. is called, passing along a pointer to the command line.

S.T. will attempt to set up the search structures for the individual words in the search term (described later). If it succeeds, a value of zero is returned to SUBJ. If it fails, (because no unculled search words were found) a negative value is returned.

When SUBJ. sees a negative return it returns the search form to free storage, zeroes the pointer SSF.(CL.), and prints out an error message to the user stating that no searchable words were found in his request.

If SUBJ. gets a zero return from S.T, it sets the SNX(SST.) indicator (search not yet executed) and returns to CLP.

B. Procedures Calling SUBJ:

CLP(via CALLIT)

C. Procedures Called by SUBJ:

CLEANP, DELIST, FRET, FREZ, S.T, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
RLIC(SST.)	Restored list in core	x	
SNX(SST.)	Search not executed	x	x
SSF(CL.)	Subject search from		x

E. Arguments:

Staptr: ASCII pointer to command line

F. Values:

Code = 0

G. Error Codes:

-2: error dissecting command line

H. Messages:

1. "Your search term contained no searchable words. Please review your search request." ($inter3$)

I. Length:

60_8 or 48_{10} words

J. Source:

INTPRT ALGOL

K. Files Referenced

None

3.2.2.3 TITLE

Purpose

To interpret TITLE commands

Description

A. Operation: Code = TITLE (Staptr)

This procedure performs most of the functions of the procedure SUBJ. except that it sets up a pointer to the title search form (TSF.(CL.)) and turns on the title indicator, TIT. Before calling CLEANP to clean up the previous search structure, it examines SNX(SST.) (search-not-yet executed flag) to see if a previous subject command on the same line has set up a structure which should not be deleted. The value returned from S.T. is processed in the same way as in the procedure SUBJ.

B. Procedure Calling TITLE:

CLP (via CALLIT)

C. Procedures Called by TITLE:

CLEANP, DELIST, FRET, FREZ, S.T, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
RLIC(SST.)	Restored list in core	x	
SNX(SST.)	Search not executed	x	x
TSF.(SST.)	Title search form		x

E. Arguments:

Staptr: ASCII pointer to command line

F. Values:

Code = 0

G. Error Codes:

Code = -2: error dissecting command line

H. Messages:

"Your search term contained no searchable words. Please review your search request"($inter3$)

I. Length:

64_8 or 52_{10} words

J. Source:

INTPRT ALGOL

K. Files Referenced:

None

3.2.2.4 S.T

Purpose

To construct search form.

Description

A. Operation: Code = S.T(Staptr)

Up to seven simultaneous search words can be handled by Intrex. S.T calls free storage for an array of seven slots for pointers to the word search forms. The pointer to this array is called the "simple search list pointer" and is inserted into the appropriate component of the subject search form, SSL.(SSF.). The mode component of SSF., which specifies the searching mode, is currently set to zero, indicating a subject/title search with no affix searching. (Pointers to the affixes of the words are later set up for possible affix matching in some future stage of development.) NEXITM is then called to find the end of the search command (next/) and extract the entire search term from the command line. The term is copied into free storage and the pointer to it is inserted into the "name pointer" component of the subject search form, N.(SSF.).

Another (six-element) array is obtained from free storage for each word in the term and its pointer is placed into one of the seven slots of the array pointed to by SSL. These six-word arrays are called the "Inverted File search forms" and the pointers to them are the IFSFP's, Inverted-File-Search-form-pointers.

NEXITM is then called repeatedly to isolate the words of the subject term and pointers to the words are passed to the stemming procedure, STEM (described below). The pointer to the stemmed word, returned by STEM, is inserted into the name component of its own Inverted File search form. Pointers to the affix and the search mode are also inserted into the appropriate components of the IFSF, along with the weight 5 attribute pointer if TIT indicates a title search term.

After all search words have been processed, the number of words (from one to seven) is inserted in the decrement of the SSL. pointer and a zero value is returned to the calling routine.

A sample search structure is presented in Fig. 3.1 showing the various pointers and structures as they appear after the described interpretation of the search request.

B. Procedures Calling S.T:

SUBJ., TITLE

C. Procedures Called by S.T:

COPY, FRET, FREZ, NEXITM, STEM, TYPEIT

D. COMMON References:

None

E. Arguments:

staptr:	ASCII pointer to command line

F. Values:

Code = 0

G. Error Codes:

-2:	error dissecting command line

H. Messages:

"You have used more words in your search request than the system can handle. Intrex will now search on the first seven significant words you have given." ($inter1$)

I. Length:

1307_8 or 711_{10} words

J. Source:

INTPRT ALGOL

K. Files Referenced:

None

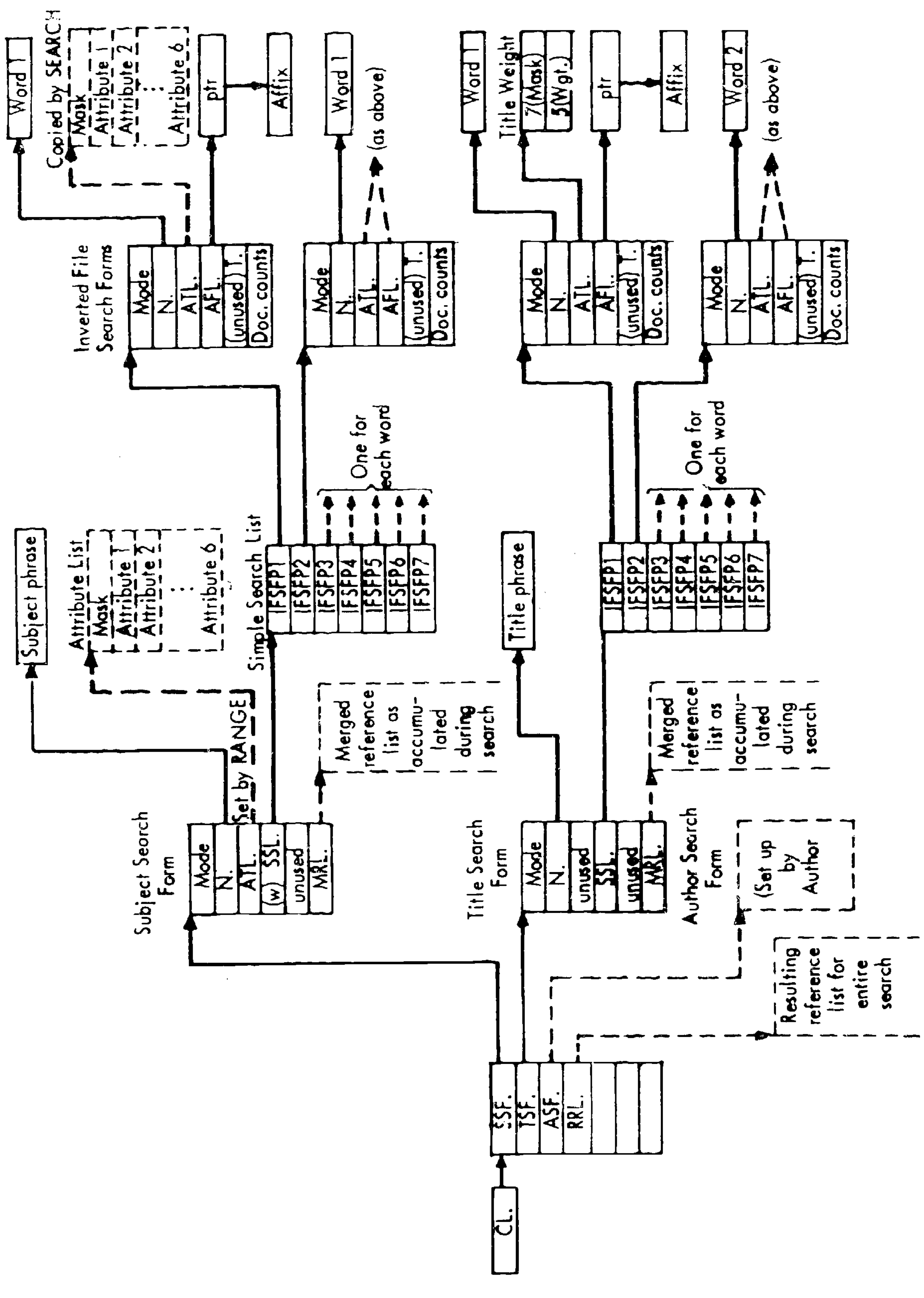


Fig. 3.1   Subject/Title Search Structure

3.2.2.5 STEM

Purpose

To remove endings from search words.

Description

A. Operation: STR = STEM(WPTR, CODE)

STEM is called by S.T. for each word of a subject or title term. It receives an argument containing a pointer to the word to be stemmed, and returns a pointer whose decrement component has been decreased to contain only the number of characters in the stem. If no ending can be removed, the pointer is unchanged and the stem is the entire word.

STEM begins by extracting from the word-pointer the address and length of the character string to be stemmed (which we will refer to as the "word"). The length of the word is used to determine the maximum length of the ending which can be removed (or equivalently the minimum length of the stem). Words of over thirteen characters have the maximum ending length set at eleven, while those under four characters are not stemmed at all. For all other lengths, the minimum stem length is three. (All character counts are in terms of ASCII characters, not Intrex special characters.)

The word-pointer is then incremented by three characters, skipping past the minimum stem to point at the first possible ending character. The remaining characters of the word (the possible ending) are copied into a temporary storage area for comparison to the entries in the ending table. The actual comparison is made by a call to VSRCH (described in Section 3.4.5.5), which searches the ending table starting at beginning of the ending-group of this length. If no match is found in that length group, the ending length is reduced by one, the copied ending is shifted one character to the left (by calling SHIFT, described with VSRCH), and STEM loops back to call VSRCH again. This is repeated until either a matching ending is found or the ending length reaches zero. If the length becomes zero, the attempt to stem is unsuccessful and STEM returns the original pointer to the calling program.

If a potential matching ending is found, VSRCH returns the address at which the ending resides. This address is used to calculate the depth into the group of this particular ending. This depth and the length of the ending which matched combine to form the "ending-code" — a 12-bit representation of the ending which is being removed from the word.

However, since VSRCH only finds a match on the first computer word of the ending (up to four characters), the remaining computer words, if any, must also be compared before a match can be claimed. This is done within the body of STEM. If a mismatch is found in one of the remaining words of characters, the ending-pointer is moved up to the next ending in the table and VSRCH is called again.

Once a full match has been made, the "condition code" associated with this ending is extracted from the ending table. The word-pointer is then advanced, if necessary, to point at the ending to be removed if the condition code allows.

Now, the condition code is used to compute the selection of a switch number, which will route STEM to one of many possible tests, each one designed to determine if the ending should be removed in the existing circumstances. For a detailed explanation of the stemming algorithm, see Reference 8.

If the condition test is passed, STEM finishes by adjusting the pointer to the new stem, constructing the ending-code and inserting it in the location provided by the second argument of the call to STEM, and returning to the calling program.

B. Procedures Calling STEM:

S.T

C. Procedures Called By STEM:

INC, COPY, VSRCH, SHIFT, GET, DEC1

D. COMMON References:

None

E. Arguments:

WPTR:	ASCII Ptr.
CODE:	to be filled

F. Values:

STR = pointer to stemmed word

G. Error Codes:

None

H. Messages:

None

I. Length:

1453_8 or 811_{10}

J. Source:

STEM1A ALGOL

3.2.3 Author Command Interpretation

3.2.3.1 INIAUT

Purpose

To initialize the AUTHOR procedure

Description

A. Operation: INIAUT( )

This procedure initializes the delimiter strings and pointers to be used by NEXITM in dissecting the command line. It is called by the segment initializer during the fixed parameter initialization phase, and its coding area is given over to free storage via FRALG after execution.

B. Procedures Calling INIAUT:

SEGINT

C. Procedures Called By INIAUT:

FRALG, .C.ASC

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

24_8 or 20_{10} words

J. Source:

AUTHOR ALGOL

K. Files Referenced:

None

3.2.3.2 AUTHOR

Purpose

To interpret AUTHOR commands.

Description

A. Operation: Code = AUTHOR(Ptr)

AUTHOR, like SUBJ. and TITLE, sets up a new search structure and discards any old structure which may exist from a previous search or restored list. If the current list is a restored NAMEd list (RLIC(SST.) = true), then DELIST is called. If it is the result of a search, then SNX(SST.) is examined to see if the search form was just set up by a previous search on the same command line. If this is not the case, CLEANP is called to discard the previous search form. AUTHOR then performs its work of setting up a new search form by calling three sub-procedures, GATP, LN, and AI which are described in detail in the following sections.

Figure 3.2 shows the structure set up by an AUTHOR command.

B. Procedures Calling AUTHOR:
CLP (via CALLIT)

C. Procedures Called By Author
AI, CLEANP, DELIST, GATP, LN

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
RLIC(SST.)	Restored list in core	x	
SNX(SST.)	Search not executed	x	

E. Arguments:
Ptr: ASCII pointer to command line

F. Values:
Code = 0

G. Error Codes:
Code = -2: error parsing command line

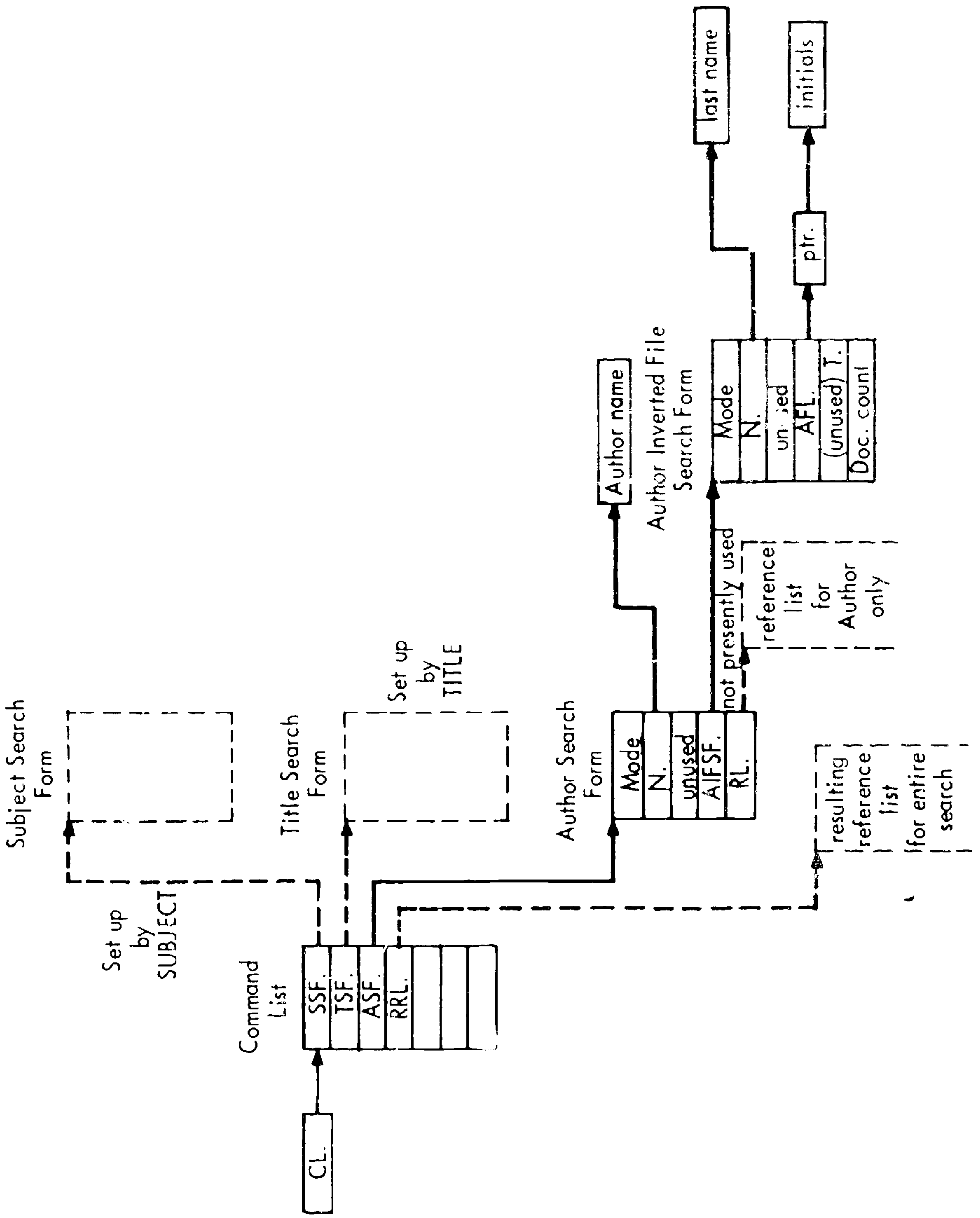


Fig. 3.2 Author Search Structure

H. Messages:

None

I. Length:

26_8 or 22_{10} words

J. Source:

AUTHOR ALGOL

K. Files Referenced:

None

3.2.3.3 GATP

Purpose

To get author template

Description

A. Operation:

A five-element array is obtained from free storage and its pointer deposited in the Author Search Form component of the command list (ASF.(CL.)). Then a six-element array to be used as the Author Inverted File search form is borrowed from free storage. A pointer to this array is inserted into the AIFSF component of the ASF. The search mode of both these forms is then set to the value found in RAM(POT.)(ordinarily set to 10 by INIPOT during fixed parameter initialization). Finally, SNX (SST.) is set to indicate a search is pending and a zero value is returned to AUTHOR.

B. Procedures Calling GATP:

AUTHOR

C. Procedures Called By GATP:

FREZ

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
SNX(SST.)	Search not executed		x
RAM(POT.)	Residual author mode	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

[illegible] words

J. Source:

AUTHOR ALGOL

K. Files Referenced:

None

3.2.3.4 LN

Purpose

To process author's last name

Description

A. Operation: Code = LN( )

The pointer to the command line passed to AUTHOR by CLP is used by LN to read the author's last name from the command. NEXITM is called with this pointer as an argument and the returned pointer to the name is deposited in the name slot of the author-search-form, N.(ASF.(CL.)). If NEXITM fails to extract a name, a -2 error code is returned to AUTHOR.

If the name is found, an appropriate amount of free storage is utilized and the full name (including first name or initials, if any) is copied into the free storage area. The pointer in N.(ASF.(CL.)) is then changed to point to this copy.

Now NEXITM is employed to isolate the last name only by including a comma in the list of item delimiters. The last-name-pointer is inserted into the name slot of the Inverted-File-search-form, N.(AIFSF.(ASF.(CL))), a copy of the last name is then placed in free storage and the N. pointer is modified to point to the copy.

If the item delimiter found by NEXITM was a slash, no first name or initials were found and LN returns to AUTHOR with a value of zero. Otherwise, the mode of both the author-search-form and the Inverted-File-search-form is modified to indicate that an affix search is to be performed (mode $\geq$ 100). Slots for the author's initials and for its pointer are obtained from free storage and value of 1 is returned to AUTHOR.

B. Procedures Calling LN:

AUTHOR

C. Procedures Called By LN:

COPY, FREE, FREZ, NEXITM

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
RAM(POT.)	Residual author mode	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

Code = -2: last name not found

H. Messages:

None:

I. Length:

215_8 or 141_{10} words

J. Source:

AUTHOR ALGOL

K. Files Referenced:

3.2.3.5 AI

Purpose

To process author's initials

Description

A. Operation: Code = AI( )

This procedure extracts the initials as given in the search command and uses them as the affix part of the search form.

First, an initial-counter is reset to zero. Then NEXITM is called to get the next item from the command line, using space, period, and slash as the item delimiters. If NEXITM fails to return a pointer, the delimiter found (TF) is examined to see the end of the command has been reached. If the delimiter is not a slash, an error code of -2 is returned to AUTHOR. Otherwise, all the initials have been extracted.

When NEXITM returns a pointer to an initial (or first name), the first character is removed and packed* into the slot reserved for the affix by LN. If no slash has yet been found, the initial counter is incremented by 1 (if not over the limit of 3) and another call to NEXITM is made.

If more than three initials are seen, an error message is typed and only the first three are processed.

When all initials have been packed into the affix slot, its pointer is deposited into the appropriate part of the Inverted File-search-form, AFL.(AIFSF.(CL.)), along with the relevent counts. A zero value is returned to AUTHOR.

B. Procedures Calling AI:
AUTHOR

C. Procedures Called By AI:
GET, NEXITM, TYPEIT

D. COMMON References:
None

*Initials are ASCII characters packed three to a computer word, left justified. Note that, if a first name is given, only its first letter is used.

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

Code = -2: initials not found

H. Messages:

"The AUTHOR command can accept a maximum of three initials. All others are ignored." ($inter2$)

I. Length:

224_8 or 148_{10} words

J. Source:

AUTHOR ALGOL

K. Files Referenced:

None

3.2.4 Primary Search Control

3.2.4.1 SEARCH

Purpose

To control search of Inverted Files

Description

A. Operation Code = SEARCH ( )

The main, controlling procedure called SEARCH is very short and simple. It does some initializing, such as turning on the "blip" feature (which indicates to the user when his search is being processed by CTSS), and resetting the appropriate indicators, state flags and modes. It then calls the three search routines, SSRCH (subject), TSRCH (title), and ASRCH (author) in that order. If any of those routines returns to SEARCH with a negative value, indicating a failure to locate the search request, no further search attempts are made.

After calling the individual search facilities, the RRL(CL.) pointer to the resulting reference list is examined to see if it has been filled by a successful search. If so, the RRLE bit of the system state table is set to indicate that a resultant reference list exists. The document count is copied from the pointer in RRL. to the DCNT slot of the command list.

Finally, the blip is turned off, the last Inverted File segments to be opened during the search are closed, if necessary, and SEARCH returns to the Intrex supervisor (SUPER).

B. Procedures Calling SEARCH
SUPER (via CALLIT)

C. Procedures Called By SEARCH:
ASRCH, CLOSE, SETBLP, SSRCH, TSRCH

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
SNX(SST.)	Search not executed		x
RRLE(SST.)	Reference list exists		x
BLIP(POT.)	Blip characters	x	
IFS2(POT.)	Name2 of Inverted Files	x	

E. Arguments:
None

F. Values:
Code = 0: search succeeds

G. Error Codes:
Code = -1: search fails (I/O error)

H. Messages:
None

I. Length:
157_8 or 111_{10} words

J. Source:
SEARCH ALGOL

K. Files Referenced:
AInnn date

3.2.4.2 SSRCH, TSRCH, STRCH

Purpose

To control subject/title searches

Description

A. Operation: Code = SSRCH( )
Code = TSRCH( )
Code = STRCH(SFP)

The heart of the search module, as far as subject/title searches are concerned, is in STRCH. This is where the individual query word's search forms (set up by S.T. in INTPRT) are selected, passed to IFSRCH for look-up in the Inverted Files, and intersected by calls to ANDER. Since STRCH is used for both subject and title searches, it is set up for one or the other by the smaller, control routines named SSRCH and TSRCH, respectively. Although these are separate routines, they are so closely associated that, for purposes of operational description, they may be clumped together with STRCH.

SSRCH is the first search procedure called by SEARCH. It examines the subject-search-form slot of the command list (SSF.(CL.)) to see if a subject search has been requested. If that slot is empty, return is immediately made to SEARCH with the "subject flag" reset.

If a pointer is found in SSF.(CL.), the "subject flag" is set, the "anding mode" for ANDER is set to zero (causing term number matching), and STRCH is called with the pointer as an argument. Upon return to SSRCH, the pointer to the rearranged Inverted File search form-pointers (explained below), is stored in a component of the command list called RESUB. The currently active Inverted File segment name is saved for possible closing later in SEARCH, and SSRCH returns to SEARCH with the same value it received from STRCH (a zero if no error condition found).

TSRCH which is called next by SEARCH, performs essentially the same operations as SSRCH, except that it takes the title-search-form-pointer from command list and, if not zero, passes it to STRCH. It also sets the "title flag" and, if the "subject flag" is also on, sets the "anding mode" to 1 so that ANDER will ignore term numbers in intersecting the title reference lists with the subject list.

In TSRCH, upon return from STRCH, the pointer to the re-arranged title Inverted File search form pointers is inserted into the command list slot called RETIT.

The first thing STRCH does, when called by SSRCH or TSRCH, is to extract the number of query words (Inverted File search forms) involved in the search from the decrement of the simple search list pointer. If this number is greater than 1, it means that reference lists from the Inverted Files are going to have to be intersected by ANDER (the general Boolean procedure). In order to improve the efficiency of the "anding" process, it is desirable to start off with a list which is not too large. The object is to take advantage of the fact that the resulting list can be no longer than the smallest individual list. An easy and quick way of getting some idea (although not an exact figure) of the size of the lists involved in the search before they are finally looked-up in the Inverted File is to look at the main directory and see how many sections are devoted to holding each list. The actual method of doing this is described below under MEADIR. At this point, it suffices to point out that a procedure named REORD is called by STRCH which uses MEADIR to find a list not longer than one Inverted File segment among those which will be looked-up. If one of that size is found, it is placed at the top of the group of search form pointers, trading places with the one which was there. If none is found to be less than one section in length, then the smallest one is put at the top. This is done by creating a new list of Inverted File search form pointers. A pointer to this new list is returned to SEARCH from REORD to be used in setting up sequential calls to IFSRCH for locating the search words and extracting their reference lists. This pointer to the re-ordered list is later inserted into the command list as described above.

Once the list of pointers is re-ordered, they are extracted, one by one, and fed to IFSRCH for the look-up. Other preparations are made on each search form, however, before IFSRCH is called. If an attribute list exists in the main search form, it is transferred to the individual-word search form and the attribute search indicator is turned on. If the number of words to be looked-up is one, then a code is sent to IFSRCH to determine the selection of common buffers into which to read the list. (See description of IFSRCH in Section 3.2.5.1)

IFSRCH is called with three arguments. They are the Inverted File search form pointer, the number of the word being processed, and the buffer code. IFSRCH returns with a pointer to the resulting reference list (or a zero if the search failed) and the name of the Inverted File segment in the argument which sent the buffer code.

STRCH then examines the "result" slot of the search form for the presence of IFSRCH error codes. If an error code is found, an error message is recorded in the Monitor File and another is typed to the user. If this occurs, the search is aborted and an error code of -3 is returned to SEARCH.

If no error is found in the "result" slot, the reference pointer is examined. If it contains a zero, the search was unsuccessful and STRCH returns a value of -1 to SEARCH.

A successful search returns a reference list pointer from IFSRCH to STRCH. Before this pointer is used for further processing of the search query, the need for attribute matching is determined. If the attribute search indicator is on and STRCH is processing other than the first word of a search, then a second attribute search indicator is turned on to be used as an argument when calling ANDER for the intersection of this list with the previously acquired list. On the other hand, if this word is the first and only one of the search, then a special call must be made to the attribute screening routine, ATSCRN, from STRCH. ATSCRN accepts the reference pointer as an argument, selects only those references which match the required attributes, and returns a pointer to the new list.

Next the main reference pointer at RRL.(CL.) is examined to see if any previous look-up has produced a list. If this pointer is zero, the current reference pointer supplied by IFSRCH or ATSCRN is inserted. For this first obtained list, the pointer is also saved aside for future use by STRCH and the name of the Inverted File segment is remembered.

If a pointer already exists in RRL.(CL.), then the list it points to must be intersected with the current list. ANDER is called with these two list pointers as arguments. Additional arguments are the "anding mode" (set by SSRCH and TSRCH) and the most recently set attribute search indicator, which will cause ANDER to screen attributes, before intersecting the lists, via calls to BUFSCN (see Section 3.3.4.7). ANDER returns a pointer to the intersected list which becomes the new RRL.(CL.), after the old augmented list pointer associated with the old RRL.(CL.) is deleted.

The current list pointer (before ANDing) is also deleted and the source Inverted File is closed.

If the new resultant reference list pointer is a zero, then STRCH returns a -1 value to SEARCH indicating that the search failed. If the new pointer is less than zero, this indicates an error condition during ANDing and a -2 value is returned to SEARCH to signal the error.

When the new pointer is a list pointer, the number of documents involved in the list is extracted from the decrement and inserted into the result slot of the Inverted File search form of this word.

If there are more words to process, STRCH then loops back to select the next search form and use it in calling IFSRCH etc.

After processing all the words, or search forms, the resultant reference list is saved in the search form of this word for possible future use (unused at present), the first list pointer is deleted, unless it is the only one, and a zero value is returned to SSRCH or TSRCH and hence to SEARCH.

B. Procedures Calling SSRCH, TSRCH, STRCH:

SEARCH (calls SSRCH, TSRCH)
SSRCH (calls STRCH)
TSRCH (calls STRCH)

C. Procedures Called By SSRCH, TSRCH, STRCH:

ANDER, ATSCRN, CLOSE, IFSRCH, LOCMES, REORD, SHOWER, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
SSF.(CL.)	Subject search form	x	
RESUB(CL.)	Reordered subject list		x
TSF.(CL.)	Title search form	x	
RETIT(CL.)	Reordered title list		x
RRL.(CL.)	Resultant reference list	x	x

E. Arguments:

SFP: address of title or subject search form

F. Values:

Code = 0 search successfull
Code =-1 search failed

G. Error Codes:

Code = -2: error during ANDER
Code = -3: error during IFSRCH

H. Messages:

1. "An error in th files caused zero documents to be retrieved in s the word x. Avoid using this word, if error re-occ ($serr$, $serr2$)

I. Length:

516_8 or 334_{10} words

J. Source:

SEARCH ALGOL

K. Files Referenced:

AInnn date

3.2.4.3 ATSCRN

Purpose

To screen attributes in reference words

Description

A. Operation: Ptr = ATSCRN(Ap)

This procedure resides in IFSRCH ALGOL but is called only by STRCH and so will be included here and not in the IFSRCH description. It is used to compare specified components of the references in a list to one or more attribute specifications as chosen by the user in his search request. One argument is passed to ATSCRN. This is a pointer to the list of references to be screened. A common buffer is selected (for reading the balance of the list from the disk, if necessary) by using the buffer number stored in the tag of word three of the augmented list pointer by IFSRCH. This insures against using a buffer which is being used for holding another Inverted File list. The address of the first reference in the list to be screened is also obtained from the pointer and all the necessary counters and indices are reset to zero.

Common buffer 1 is used as an output storage area for the references which match the required attributes unless the entire list is currently in core. If some of the references to be screened are disk-stored, the disk address and number of disk references are also extracted from the pointer. A parameter for keeping track of the number of references in a disk record is set to 432 unless the disk file is an Inverted File segment, in which case it is one less because of the presence of a section header.

At this point, a sub-procedure named BUFSCN is called (Section 3.3.4.7). It performs the actual comparison of each reference to the required attributes for those references currently in core. References which BUFSCN finds acceptable are stored in the output area whose address is supplied in an argument to the call. Having processed this buffer, ATSCRN decides if there are more references left on disk to be processed and, if so, reads the next block via RDWAIT into the same common buffer area. The RDWAIT variable RELLOC is updated and the number of references left to read is reduced.

The starting location of this new batch of references is set (skipping the section header if we are reading an Inverted File segment) and ATSCRN goes back to call BUFSCN again.

After all the references are screened, it is determined whether all the references were originally in core. If so, the size of the list was simply reduced by the attribute screen but the list is still core-stored and continues to exist in the same place in core. Otherwise, the output buffer is written into the Dump File (finishing a job possibly started by BUFSCN if it filled the buffer earlier). The Dump File record count is updated and a new augmented list pointer is constructed in place of the old one passed to ATSCRN. A pointer to this augmented pointer is returned to the calling program (STRCH).

B. Procedures Calling ATSCRN:

STRCH

C. Procedures Called By ATSCRN:

BUFSCN, RDWAIT, WRWAIT, PREPTR

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBFn(POT.)	Input buffer	x	
COMBF1(POT.)	Output buffer	x	
TOTBLK(POT.)	Dump File relocation	x	
DFN1(POT.)	Dump File name1	x	

E. Arguments:

Ap: pointer to list of references

F. Values:

Ptr = pointer to screened references

G. Error Codes:

None

H. Messages:

None

I. Length:

455_8 or 301_{10} words

J. Source:

IFSRCH ALGOL

K. Files Referenced:

AInnn date
DUMnnn FILE

3.2.4.4 REORD, MEADIR

Purpose

To reorder search terms

Description

A. Operation: Ptr = REORD (Lptr, Nifs)

REORD attempts to find a search word among those in the group of Inverted File search forms presented to STRCH whose reference list is not long enough to require more than one section. If none of the lists are short enough, REORD selects the shortest one available. The selected list's search form pointer is moved to the top of the pointer list (unless it already happens to be there) and the pointer which was there (the user's first search word) is moved to the old location of the small list pointer. This rearrangement is used by ANDER to optimize the intersection of the lists involved in the search. It is reflected in the COUNT results which are listed in this rearranged sequence. The original order is also saved, however, for the feed-back of the user's search request (by EVAL).

Arguments passed to REORD includes a pointer to the list of Inverted File search form pointers and the number of pointers in that list. Initialization of REORD includes obtaining the core addresses of the two Inverted File directories (via IFSET), obtaining an array for storing the re-ordered list of pointers from free storage, and setting a variable which will hold the "smallest list size found so far" to a high enough value to force selection of the first list as being the smallest so far.

Then, each search word is passed to the measuring routine, MEADIR, which looks it up in the Inverted File directories and determines its approximate size. The look-up is done in the same way as in LOCSEC, described under IFSRCH. The initial letter of the search word is used to select a related slot in the first directory, which supplies the address within the second directory where that alphabetic group begins. Comparison is then made between the search form word (converted to 5-bit code by NAM5) and the abbreviated (no more than 7 characters) entries of the directory.

As long as the search word is greater than (beyond) the directory entry, the scan continues down through the directory. If the search word is found to be less than (before) the directory entry, the "find point" has been passed and it can be assumed that the list for the word in question does not exceed one section but was contained (if it exists at all) in the section just passed. In the case where the search word and directory entry are equal, an "overflow-section counter" is incremented and the comparison continues on to the next directory entry.

Once the "find point" has been passed, the scan of the directory ends and the section counter just mentioned is returned to REORD.

Segments of the Inverted File which extend beyond 10 sections because of a large list in the tenth section present a special problem since the extra sections are not represented in the directory. When the "find point" turns out to be in Section 10, therefore, a call to the CTSS routine, FSTATE, is made to determine the length of the segment, from which is computed the additional sections to be added to the section count.

The section count returned to REORD is ero if the list was contained in a single section. This prompts the immediate switching of this list pointer with the one at the top of the list. REORD then has done its job and returns a pointer to this re-ordered list to STRCH. If the section count was greater than zero, it is compared to the variable holding the smallest section count so far (set to a high value during initialization). If this section count is smaller, it replaces the old count and the position of this search word pointer within the list is remembered. Then REORD loops back to call MEADIR with the next search word pointer as an argument. If all search words have thus been measured without finding a non-overflow list, the position of the smallest one is used and switched with the top list.

B. Procedures Calling REORD, MEADIR:

STRCH (calls REORD)
REORD (calls MEADIR)

C. Procedures Called By REORD, MEADIR:

FREZ, FSTATE, GET, IFSET, NAM5

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
IFS2(POT.)	Name 2 of Inverted File	x	

E. Arguments:

Lptr:	address pointer
Nifs:	integer

F. Values:

Ptr = pointer to new list of pointers

G. Error Codes:

None

H. Messages:

None

I. Length:

360_8 or 240_{10} words

J. Source:

SEARCH ALGOL

K. Files Referenced:

AInnn date

3.2.4.5 ASRCH

Purpose

To control author searches

Description

A. Operation: Code = ASRCH( )

ASRCH performs the same functions for author search requests that SSRCH, TSRCH, and STRCH perform for subject/title searches. The author search form pointer is extracted from the command list. If it is zero, ASRCH returns a zero to SEARCH.

If a pointer exists, the Inverted File search form pointer is extracted from the author search form and passed along to IFSRCH for lookup. The other arguments to the IFSRCH call are a 1 (representing the number of words in the search phrase — in this case only a last name) and a buffer selection code (1 or 2, depending upon whether or not a subject and/or title search has already been performed). IFSRCH returns a pointer to the reference list, if the search was successful, and the document count in the result slot of the Inverted File search form. If ASRCH finds an error code in this slot, it enters one error message in the Monitor File and prints another to the user. In this event, an error code of -3 is returned to SEARCH to indicate an I/O or file structure error has occurred.

If no error code is returned, the reference pointer is examined. If this is zero, it means the search failed. In this event, a zero is inserted into the resultant reference list pointer, RRL.(CL.), and a -1 is returned to SEARCH. A successful search of the author Inverted File calls for possible further processing of the reference list. If the search was not a combined one (no resultant reference list already exists) and a screen on attributes is called for, then ATSCRN is called to perform the screening and supply a pointer to a new list of screened references.

It is important to keep in mind, here, that the usual kind of attribute restriction, namely RANGE, is not applicable to author searches and that the attribute matching in this case is really an affix (initials) screen.

If a resultant reference list already exists, the list returned from IFSRCH must be ANDed with it. The pointers to the two lists are passed as arguments to ANDER, along with an "anding mode" of 2(which prevents trying to match on term numbers) as an attribute search indicator. (If this indicator is on, ANDER, will call BUFSCN to screen references with matching attributes from the author reference list.)

ANDER returns a pointer to the intersected list which becomes the new resultant reference list. The augmented pointer to the author list is deleted from the table, as is the old resultant reference list pointer.

If ANDER returns an error code (less than zero), then ASRCH returns a -2 to SEARCH. Otherwise, the Inverted File segment opened by IFSRCH is closed, the new document count is copied from the decrement of the resultant reference list pointer to the result slot of the Inverted File search form, and ASRCH returns a zero (success code) to SEARCH.

B. Procedures Calling ASRCH:

SEARCH

C. Procedures Called By ASRCH:

ANDER, ATSCRN, CLOSE, IFSRCH, LOCMES, SHOWER, TYPEIT

D. COMMON References:

None

E. Arguments:

None

F. Values:

Code	= 0:	if search succeeds
	= -1:	if search fails

G. Error Codes:

Code	= -2:	error during ANDER
	= -3:	error during IFSRCH

H. Messages:

"An error in the computer files caused zero documents to be retrieved in searching on the word BAD*. Avoid using this word if error re-occurs" ($serr$, $serr2$)

*The word which triggered the error is printed here.

I. Length:

353_8 or 235_{10} words

J. Source:

SEARCH ALGOL

K. Files Referenced:

None

3.2.4.6 CLEANP, STCLN, ACLN

Purpose

To delete search structures

Description

A. Operation: CLEANP( ), STCLN( ), ACLN( )

Before a new search structure is put together at the command interpretation stage, the old structure, if one exists, must be deleted and returned to free storage (via FRET). This is also necessary when a list is formed by the DOCUMENT command or a NAMEd list is being restored to active status.

CLEANP examines the main search form pointers of the Command List, SSF.(CL.), TSF.(CL.), and ASF.(CL.) to see if any search structure exists for subject, title, or author searches, respectively.

The presence of a subject and/or a title search structure causes CLEANP to call the procedure STCLN, which accepts the search form pointer as an argument and uses it to chain through the structure returning the free storage arrays obtained by S.T, SUBJ., TITLE (see Section 3.2.2.4) and REORD (see Section 3.2.4.4).

If CLEANP calls STCLN to return a title search structure, an indicator is set which tells STCLN that it must also return an attribute and mask for each search word in the user's query. In this case the attribute list, ATL.(TSF.(CL)), is returned by calling the procedure ATLCLN. (see Section 3.2.8.12).

If an author search structure exists, CLEANP calls ACLN to return those structure arrays. ACLN accepts the ASF.(CL.) as an argument and calls FRET to return the areas obtained by AUTHOR when it interpreted the user's AUTHOR search request. The search mode is examined for the presence of an affix (initials) search, and the affix list is returned if one exists.

After CLEANP has used these two sub-procedures, the current augmented list pointer is deleted from the pointer table by a call to DRPPTR (Section 3.2.4.8), unless the current pointer has been NAMEd (list type 4).

Now RRL., RESUB, and RETIT of the Command List are zeroed and the RRLE(SST.) indicator is reset, thus destroying all indications of a "resultant reference list".

Then, the Dump File is truncated to zero length and the number of records in the file, saved in TOTBLK(POT.), is set to zero.

Finally, CLEANP calls FCLEAN (see below) to clean up OUTPUT and RESTRICT structures.

B. Procedures Calling CLEANP:

SUBJ., TITLE, AUTHOR, RESTOR, NUMBER

C. Procedures Called By CLEANP:

DRPPTR, FILCNT, TRFILE, FCLEAN, FRET, ATLCLN (via CALLIT)

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
SSF.(CL.)	subject search form	x	
TSF.(CL.)	title search form	x	
ASF.(CL.)	author search form	x	
RRL.(CL.)	resultant refer. list	x	
RESUB(CL.)	reordered subj. forms		x
RETIT(CL.)	reordered title forms		x
DFN1(POT.)	Dump File name-one	x	x
TOTBLK(POT.)	Dump File block count		x
RRLE(SST.)	result-refer-list-exists flag.		x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

465_8 or 309_{10} words

J. Source:

CLEANP ALGOL

K. File References:

DUMnnn FILE

3.2.4.7 DELIST

Purpose

To clean up list pointer and and associated structures.

Description

A. Operation: DELIST( )

DELIST first examines RRL.(CL.) to make sure that a current reference list exists. If so, the type of list is examined. Any type other than NAMEd (type 4) will prompt DELIST to call the procedure DRPPTR (described next) to remove the augmented list pointer from the table.

Next, FCLEAN (Section 3.2.4.9) is called to wipe out the old OUTPUT and secondary search (RESTRICT) specifications. Other tasks of DELIST include: setting the main list pointer, RRL.(CL.) to zero; truncating the Dump File to zero and setting its block count in TOTBLK(POT.) to zero; resetting the System State Table bits, RRLE and RLIC (resultant reference list exists, and restored list in core) to false.

B. Procedures Calling DELIST:

SUBJ., TITLE, AUTHOR, RESTOR, IFSRCH, NUMBER

C. Procedures Called By DELIST:

DRPPTR, FCLEAN, TRFILE

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
RRL.(CL.)	result.reference list	x	x
DFN1(POT.)	Dump File name-one	x	
TOTBLK(POT.)	Dump File block count		x
RRLE(SST.)	result.-refer.- list-exists flag		x
RLIC(SST.)	restored- list-in-core flag		x

D. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

55_8 or 45_{10} words

J. Source:

CLEANP ALGOL

K. File References:

DUMnn FILE

3.2.4.8 DRPPTR

Purpose

Drop list pointer from table

Description

A. Operation: DRPPTR( )

The pointer in the Resultant Reference List slot of the Command List, RRL.(CL.), points to a three-word augmented list pointer. DRPPTR inserts a zero into each of the three words of this pointer through use of the "full word component" facility of AED.

B. Procedures Calling DRPPTR

DELIST, AND., NAME

C. Procedures Called By DRPPTR

None

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
RRL.(CL.)	result. refer. list	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

25_8 or 21_{10} words

J. Source:

CLEANP ALGOL

K. File References:

None

3.2.4.9 FCLEAN

Purpose

To clean-up OUTPUT request form

Part of the process of removing all traces of a previous search in preparation for the creation of a new list is to discard the output structures set up by OUT. (described in Section 3.2.7.2) and the secondary search (RESTRICT) structure set up by IN. (described in Section 3.2.7.3).

Description

A. Operation: FCLEAN( )

The Output Request List Pointer in ORL.(CL.), which points to a permanent array containing field numbers to be output, has its decrement (count of field numbers) set to zero by masking off all but the address portion of the word.

Next, the Field Search List pointer in FSC.(CL.) is reduced in the sameway (eliminating the count in the decrement), and the pointers stored in the ten-word array used for holding RESTRICT specification pointers are examined one-by-one. The free-storage area to which each one of these pointers points is returned via FRET, and the pointer itself is changed to zero.

B. Procedures Calling FCLEAN:

CLEANP, DELIST, NEWPT

C. Procedures Called By FCLEAN:

FRET

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
ORL.(CL.)	output request list		x
FSL.(CL.)	field search list		x

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
74_8 or 60_{10} words

J. Source:
FCLEAN ALGOL

K. File References:
None

3.2.5 Inverted File Lookup

3.2.5.1 IFSRCH

Purpose

To search Author or Subject Inverted Files for a list

Description

A. Operation: Ptr = IFSRCH (IFSFP, WRDNO, IFS1)

IFSRCH accepts, as one of its arguments, an address pointer to an Inverted-File-search-form*. This array is of the following form:

```
WORD 1 - mode (1-3 decimal digits)
WORD 2 - name pointer (to ASCII string)
WORD 3 — attribute pointer (to list of attributes)
WORD 4 - affix pointer (to list of pointers)
WORD 5 - time array pointer (for time checks)**
WORD 6 - result (filled by IFSRCH)
```

The name pointed to by word 2 is looked up in the Inverted Files according to the mode in word 1. Although the mode specifies that an exact stem match must be made (0 in the units digit)***, a match on affix strings as well as stems may or may not be called for (1 or 8 in the hundreds digit). The author file will be searched if the tens digit is a 1, while an zero in that place specifies a subject/title file search.

If an attribute list is to be considered in the selection of qualifying references, then a pointer containing the address of the attribute mask (which must immediately precede the attribute list) appears in word 3 of the search form. The length of the attribute list must appear in the decrement of word 3.

Word 4 may contain a similar pointer to a list of Intrex pointers, which in turn point to ASCII strings of endings⁴ or author's initials. The pointer in word 4 will not be put to use in an affix string match, however, unless the mode in word 1 calls for affix matching.

The other two arguments passed to IFSRCH by SEARCH, WRDNO and IFS1, are used to govern the selection of common buffers which IFSRCH

---

* 1. Set up by SUBJ., TITLE, or AUTHOR

** 2. Not presently used.

*** 3. Only exact match mode is currently in use.

4. At present only author's initials are used in searching with affixes.

will use in reading the Inverted File. First WRDNO tells if the current word is the first word of a search request. If it is, IFS1 will indicate if it is a one-word subject, title or author search or a combination of them. The conventions used in this determination follow the outline below.

Search Type	IFS1 =	Buffer Used
1-word subject	1	5
1-word title, no subject.	1	5
Author, no subj. or title	1	5
1-word title with subj.	2	4
Author with subj. or title	2	4

If WRDNO is greater than 1, then IFSRCH used buffer 3 for reading in the Inverted File sections. IFS1 is also used to relay back to SEARCH the first name of the Inverted File segment where the sought word was found.

After establishing the buffer selection and the search mode and initializing a few parameters, IFSRCH passes the pointer to the search word to a sub-procedure named LOCSEC which will look the name up in the Inverted File directories. LOCSEC selects the appropriate section number of the appropriate Inverted File segment where searching should begin. The proper RELOC, or depth into the file, is computed using this section number, and reading of the file begins.

After checking for the presence of a section header fence, the offset of the first list in the section (also in the header) is used to compute the starting location of the list. If this offset is zero (meaning no list starts in that section) either the next section or the previous section is read, depending on the direction of scan. Almost always the scan is forward. The one exception will be explained in the writeup of LOCSEC.

When a list header is located, a check is made for the list fence and a pointer to the "name" (search word) is set up. The number of characters in the name and number of affix codes associated with it are set aside. The string-comparing procedure COMPUL is called to match the list name against the one pointed to by the search form. The return from this procedure indicates a high, low, or equal comparison. A high compare means that the search form word is beyond that point in the Inverted File and we must keep looking. A low compare means that the "find point" has been passed without making a match and that (unless we are going to work backward) the search has failed. In the former case, the list header is used to locate the start of the next list and IFSRCH loops back to call COMPUL again. If the end of the section is reached without reaching the "find point", the next section is read and the search continues. If all the sections of a segment are scanned unsuccessfully, that segment is closed and the next one in sequence is opened for reading.

If COMPUL finds that the two names match to the end of the search form word, their lengths are compared. An inequality in length constitutes a failure and the search ends in a mismatch. If the two lengths are the same, however, a match has been found and the reference and document counts are extracted from the header of the Inverted File list.

At this point, the affix-search indicator is examined and, if on, the affix matching procedure, MATAFF, is called. If affix matching is successful, a pointer to a list of satisfying references will be filled upon return from MATAFF. If it is unsuccessful, that pointer will be zero and the miss-match exit is taken from IFSRCH.

If affix matching succeeds, or none is required, an indicator is set and the name of the Inverted File segment containing the matching list is plugged into the IFS1 argument to be returned to the calling program.

Now the address of the first reference word in the list is computed by moving down from the top of the list header the length of the header, name field, and affix field. Since affix fields may spill over to the next section, a test must be made to see if the references for this list are actually in the next section and, if so, how far in.

Once the actual starting address of the reference list is established, the number of references that might possibly be contained in the remainder of the buffer is computed. If this number is less than the total references in this list, it is used as the core-stored count in word two of the three-part augmented pointer created by the sub-procedure, PREPTR. The remaining references become the disk-stored count in word three of the augmented pointer and, for these cases, the name of the Inverted File segment goes into word one.[1] The disk and core addresses are also inserted by PREPTR and the list type is set in the tag of word two. If all the references are contained in the buffer, the list is considered a core-store list (type 0) and words one and two of the pointer are left blank.

After the augmented pointer has been created, it is inserted into the list pointer table (part of RESLIS ALGOL) by a call to TABENT.

Finally, the document count of the list, extracted earlier from the list header, is inserted into the decrement component of both the "result" slot of the search form and the reference pointer returned to SEARCH, which will become the "Resultant Reference List" pointer or RRL.(CL.).

When a search fails, the reference pointer is checked to see if a previous call to IFSRCH for another word in the same search term produced a list pointer. If so, that pointer is deleted from the list pointer table by a call to DELIST since all words of a query must match the Inverted File in order to make a successful search. The last scanned segment is closed, the result parameters are zeroed, and a zero value is returned to SEARCH.

B. Procedures Calling IFSRCH:

SEARCH

C. Procedures Called By IFSRCH:

IFSET, LOCSEC, RDWAIT, COMPUL, CLOSE, OPEN,

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
IFS2(POT.)	Name 2 of Inverted File	x	
COMBFn(POT.)	Common buffer	x	

E. Arguments:

IFSFP:	pointer to inverted file search form
WRDNO:	sequence number of word being processed
IFS1 :	combination code

F. Values:

Ptr = Pointer to reference list augmented pointer or zero (if no match made).

1. A detailed description of the three-part list pointer may be found in Appendix B.

G. Error Codes (in last word of search form):

- -3 Premature end-of-file reading segment
- -5 Section fence missing
- -6 List fence missing

H. Messages:

"Error opening I.F. Seg. NAME1", (Seg. name) (LOCMES, BCDASC)

I. Length:

2535_8 or 1373_{10} words (including ATSCRN, BUFSCN, MATAFF, LOCSEC)

J. Source:

IFSRCH ALGOL

K. Files Referenced:

SInnn date
AInnn date

3.2.5.2 LOCSEC

Purpose

To locate a section of an Inverted File segment

Description

A. Operation: LOCSEC(Namptr, Aut)

IFSRCH passes two arguments to LOCSEC as it attempts to narrow down the Inverted File search to the appropriate segment. One argument is a Boolean variable which, if set, indicates an author search is in progress. In this case the file names are set up to use the Author Inverted File and directories. Otherwise, subject-title file names are used.

The first step in the search is to extract (using GET) the first character of the name or search word (the other argument to LOCSEC). The ASCII code for this character is then adjusted and used to select a corresponding address in the first directory, IFTABS (or IFTABA). The contents of this directory address points to the depth of the second directory (IFDS or IFDA) where the abbreviations (up to 7 letters) of list words which start with this character may be found.

Comparison is then begun at this position of directory two between the name passed to LOCSEC and the name abbreviations in the directory. These abbreviations are seven characters or less coded in 5-bit ASCII form. The argument-name then must also be converted to 5-bit ASCII by the procedure NAM5 in a separate ALGOL file of the same name. Each word of directory two represents one section of the Inverted File. The name of the first list in that section was condensed to 5-bit code and inserted into the appropriate directory slot by IFGEN during the creation of the Inverted Files. Thus, directory two is a real index — almost a table of contents — to the sections of the Inverted File. Sections of the file which contain only references from a large list and no beginning of a new list are represented by the negative form of the list name code in their corresponding directory slots. During LOCSEC's matching process, if a negative name is seen, it is skipped. A full-computer-word comparison is made between the converted argument-name and the trial directory-abbreviation. If the argument is greater (further down the alphabetical order) than the trial, the next trial in sequence is taken from the directory. When a trial is found which is greater than the argument, the sought word is assumed to be in the previous section of the Inverted File. The directory index is reduced by 1 and used to compute the proper segment and section numbers. For example, if the trial in slot number 205 was found to be the first one which was greater than the argument, then the sought name would be expected to be found in segment 20 (200/10), section 4.

If the argument and trial happed to be equal, an indicator (EQU) is set and no reduction by 1 is made to the index. Thus, the assumption is made that the full word being sought will be found at or beyond the point indicated by the matching abbreviation. This is not always true. Therefore, IFSRCH will back up and try the previous section when the "find-point" has been passed and the (EQU) indicator is on.

Having selected a segment of the Inverted File for searching, LOCSEC constructs the first name of that segment from the segment number it selected and opens the file for reading by calling the CTSS procedure OPEN before returning to IFSRCH. If an error occurs because the file is already open, a normal return is made. Any other kind of OPEN error results in an error message (1) and an Intrex abort.

B. Procedures Calling LOCSEC:
IFSRCH

C. Procedures Called By LOCSEC:
DEFBC, GET, NAM5, OPEN

D. COMMON References:
None

E. Arguments:

Namptr:	ptr to search word
Aut :	Boolean switch (used to distinguish between authoı and subject searches).

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Error opening I.F. segment <u>x</u>," (LOCMES, BCDASC)

I. Length:

See length of IFSRCH

J. Source:

IFSRCH ALGOL

K. Files Referenced:

SIxxx date
AIxxx date

3.2.5.3 NAM5

Purpose

To convert to 5-bit code

Description

A. Operation:* Str = NAM5 (Ptr)

NAM5 is called by LOCSEC with an argument containing a pointer to an ASCII string (the "name" field of the user's query search form). Up to the first seven characters of this string are converted to a 5-bit representation of the ASCII code. This is a simple matter of masking off all but the first (low-order) 5 bits in most cases. The only special case is when a non-alphabetic character (code less than 100 octal) is found within a string which started with a letter. (e.g. H* Sub 2*). Since the five-bit code for * is the same as the five-bit code for J, which would cause ambiguity problems, the non-alphabetic characters in these cases are 5-bit coded as zero. If however, the first character is non-alphabetic, then standard bit masking is done throughout the "word". The ambiguity with letters would not matter in this case since the scan of the directory would start in the early, non-alphabetic area as controlled by the directory to the directory.

Character strings of less than seven codes will be left-adjusted with zero bits filling in the unused right portion of the word. The word holding the converted string is returned to the calling program.

B. Procedures Calling NAM5:

LOCSEC

C. Procedures Called By NAM5:

GET, INC1

D. COMMON References:

None

E. Arguments:

Ptr: pointer to ASCII string

* The coding scheme described here is not ambiguity-proof and a revised algorithm is being prepared for testing.

F. Values:

Str = left-ajusted 5-bit character string

G. Error Codes:

None

H. Messages:

None

I. Length:

171_8 or 121_{10} words

J. Source:

NAM5 ALGOL

K. Files Referenced:

None

3.2.5.4 MATAFF

Purpose

To match an affix string (author's initials)

Description

A. Operation: Code = MATAFF

The design specifications of this routine and the whole strategy of matching affixes have undergone such extensive revision over the life of Intrex that many awkward and redundant operations are left in the present version of MATAFF. Currently, only author's initials are searched as affixes. Although it is undoubtedly difficult to follow the scheme by reading the source code, the general technique is as follows.

First the address of the affix header/code pairs of computer words is located within the Inverted File list whose name matched the searched word. The number of such affix pairs is extracted from the list header. Then an array is obtained from free storage which will be used to hold the matching affix position numbers (from 1 to as many sets of initials as there are in the list). An index to this array is set up.

Now the number of affix pointers (always one for authors) and the pointer to the pointers is extracted from the Inverted File search form which was passed to IFSRCH. MATAFF is then ready to start comparing the affixes of the user's query to the affixes of the Inverted File list.

As the compare logic loops through the affix lists, tests are made for a possible spill-over into the next Inverted File section, which would necessitate reading the next block into core. As each I.F. affix is taken, the initials count of that affix is taken from the affix header. The corresponding initials count of the query name is taken from the decrement of the affix pointer. A comparison (using COMPUL) is made between these two initial strings to the end of the shorter one. If a match is made, the sequence or position number of the list affix is inserted into the next available slot of the array set up for that purpose, and an indicator is set designating that the search has been successful.

Whether a match is made or not, the next affix in the list is taken and comparison is repeated. All matching affix position numbers are saved in the array. When all affixes in the list have been compared the search is over. If any matches have been made, the array of affix position numbers becomes a list of attributes and the attribute-search-mode indicator is turned on. This forces a later stage of the search to use the affix numbers in an attribute screen.

B. Procedures Calling MATAFF:

IFSRCH

C. Procedures Called By MATAFF:

COMPUL, FRET, FREZ, RDWAIT

D. COMMON References:

None

E. Arguments:

None

F. Values:

Code = 1:	match is successful
= 0:	match fails

G. Error Codes:

None

H. Messages:

None

I. Length:

See IFSRCH

J. Source:

IFSRCH ALGOL

K. Files Referenced:

AInnn date

3.2.6 Search Command Play-Back

3.2.6.1 INIEVL

Purpose

To initialize EVAL

Description

A. Operation: INIEVL

This initializing procedure is called by INIT2 in OVNEW ALGOL, which, in turn, is called by INIFIX in INITLY ALGOL during the "fixed parameter" initialization. It sets up pointers to character strings, message labels, and mask bits used by EVAL. After execution, the code used by INIEVL is given to free storage by a call to FRALG.

B. Procedures Calling INIEVL:

INIT2

C. Procedures Called By INIEVL

FRALG, .C.ASC

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

62_8 or 50_{10} words

J. Source:

EVAL ALGOL

K. Files Referenced:

None

3.2.6.2 EVAL

Purpose

To print the results of a search command

Description

A. Operation: Code = EVAL( )

Since the function of EVAL is to produce a message to the user reporting on the results of his search command, it must construct the message according to several conditions. First, it checks to see if an "output" request or a "restrict" request was given on the same command line with the search request by examining the Field Search and Output Not Executed bit of the System State Table, FXONX (SST.). If on, the message is set up to tell the user that the selected fields will be output "now". If off, the message is set up to say that the fields will be produced when the user types an output command.

Next, the resultant reference list pointer is examined as to the existance of a list. If there is one, the document count is extracted from the command list's result slot. When the document count is only 1, the message elements pertaining to a single "document" are selected for use. If it is more than 1, the components referring to plural "documents" are used.

Next, the pointer to output commands is taken from the command list, the address of the table of field names is obtained via FLDNAM (in TABLE2), and the dialog mode (long or short) is determined. Initialization of routine parameters is completed and the message to the user is begun by calling TYPEIT in the "continuation mode" (CONT) to print out the first, introductory words of the message. In "long" mode, this section (OP1) consists of the phrase, "A search on your request". In "short" mode it is blank. Now the request itself is repeated, as Intrex understood it. Pointers to the three search forms are extracted from the command list and put through a construction and typing loop, one by one. If any of the three types are unused, a count is incremented which is used to select the proper message label corresponding to the next type and the loop is repeated. Types which were used, and therefore provide a pointer to a search structure, cause extraction of the name or term and the endings or initials. A string of ASCII characters is constructed in core which joins each ending to the corresponding stem (separated by a hyphen) or follows an author name by initials. A record is kept of all the individual words looked up in the search.

Another TYPEIT call is made whose first argument is OP14, OP15, or OP16 producing the search types SUBJECT, TITLE, or AUTHOR respectively. The selection of one of these three message labels is determined by the count incremented each time through the loop. The second argument in the TYPEIT call is the pointer to the in core search term just constructed. A third argument keeps TYPEIT in the continuation mode.

If a subject command is played back to the user, the attribute slot of the search structure is examined to see if a RANGE restriction were given. If so, the word RANGE (OP14A) is displayed, followed by the range numbers (or names if long mode) the user specified.

After this loop processes all three types of search requests, another TYPEIT call, still continuing the same line, puts out the phrase, "found d* document(s)."**

If the number of documents found is zero and the user is in "long mode", the search-word count is examined. If more than one word was involved in the search (including combinations of search types), an automatic count of matching documents will be displayed by calling the procedure SEEMAT (in SEEMAT ALGOL). The call to this routine, which

---

* The number of documents as specified by the count in the command list.

** "s" added unless d = 1.

does exactly what a user-issued COUNT command does, is actually made from the supervisor, SUPER, upon return from EVAL. EVAL merely puts forth a message telling the user that the COUNT results are about to be displayed (op10). If only one search word were used, a COUNT display would be meaningless, so a message suggesting the use of other search terms or commands is given (OP18).

When some documents have been found, another TYPEIT call, using message OP4a, starts a new sentence with the phrase, "The catalog fields" (long mode) or the abbreviation,"O:" (in short mode). At this point, the bits of the output request list pointer are examined to see if any of 5 possible special fields (NORMAL, ALL, STANDARD, MATCH, and TEXT) were requested by the user. This is done in a loop which masks off one of the appropriate bits at a time to see if it exists. If so, the corresponding field name (or label for it) is selected from an array set by INIEVL and fed to TYPEIT.

Next, regular fields are processed as specified by the output request list which contains an array of field numbers selected by the user in his output command. In long mode, the numbers are used to select the corresponding position in the field-name table (constructed by TABLE) whose address was obtained earlier by FLDNAM. In short mode, only the field numbers themselves are printed.

If the routine passes this point without having typed any field labels, a bit in the output request list pointer is examined which would have been set by OUT. (the output request interpreter, residing in INOUT ALGOL) if field 1, the document number, were requested. If this bit is on, the word DOCUMENT (in long mode) or the number 1 (in short) is printed. If this bit is not on, then no output fields were requested and EVAL assumes that the user wants the NORMAL output (i.e. TITLE, AUTHOR, LOCATION). A call to TYPEIT with the message label (OP46) which produces this bit of text is made. This call is also in the "continuation mode" and now another call to TYPEIT extends the message to say "for those documents ---" (or "---that document---", as the case may be). If field restrictions were requested via RESTRICT, a set of restrict specifications will have been set up in IN., also residing in INOUT. These specifications, which are pointed to by the field search list pointer of the command list, FSL.(CL.), are also played back to the user as part of EVAL's response. If any exist, the message continues, "---which also match your field restrictions---", or just "R:" in short mode. Then the actual restrictions are displayed (as IN.understood them).

In short mode, the message to the user is now complete. In long mode, however, it goes on to say that his specified output "---will be output now," or "---will be output when you type o (for output)", depending upon whether or not an output (or restrict) command was issued with the search command.

Additional TYPEIT calls are made, in long mode, to tell the user he "--- may terminate the output at any time by hitting the ATTN key once ---", etc, and to suggest what he might do next.

EVAL returns to the supervisor with a value of zero, which is the conventional signal that a module has completed its job successfully.

An example of a response displayed to the user by an EVAL call, which exercises most of the outputs, is given under Part H.

B. Procedures Calling EVAL:

SUPER(via CALLIT)

C. Procedures Called By EVAL:

COPY, DIST, FLDNAM, FRET, FREZ, INC, PUT, RNGNAM, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
FSONX(SST.)	FSO not executed	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

None

H. Messages: (samples)

LONG

"A search on your request SUBJECT magnet-ic reson-ance/RANGE MAJOR, SECONDARY/TITLE spectroscop-y/AUTHOR Smith, rf found 6 documents. The catalog fields TITLE, AUTHOR, LOCATION for those documents which also match your field restrictions RESTRICT AFFILIATION Harvard will be output now. You may terminate this output at any time by hitting the ATTN key ONCE."

SHORT

"S: magnet-ic reson-ance/RA: 1, 2/T: spectroscop-y/
A: Smith, rf found 6 docs O: NORMAL/R: 22 Harvard"

I. Length:

1300_8 or 704_{10} words

J. Source:

EVAL ALGOL

K. Files Referenced:

None

3.2.6.3 GETEND

Purpose

To convert ending code to ASCII string

Description

A. Operation: PTR = GETEND(CODE)

As EVAL reconstructs the user's search term for play-back after the search is done, the ending codes pointed to in the affix list of the search structure are converted to their corresponding ASCII letter codes. To do this conversion, EVAL calls GETEND with an argument containing the ending code.

GETEND breaks the ending code into its two parts. The left-most four bits give the length of the ending, thus specifying which length subset must be located within the ending list. The other eight bits of the code specify which ending within the subset is to be extracted.

The location of the ending pointer table is obtained by calling INIEND (see Section 3.1 2.15). The length subset extracted from the ending code indicates which pointer is to be chosen from the table. The address portion of this pointer is then used to locate the first ending of this length in the list and the ending number is used to adjust this address to the proper ending location.

Finally, the ending is copied (via COPY) from the ending list into a declared array within GETEND and a pointer to this copy is constructed and returned to the calling program.

B. Procedures Calling GETEND:
EVAL

C. Procedures Called By GETEND:
INIEND, COPY

D. COMMON References:
None

E. Arguments:

CODE:	12-bit ending code

F. Values:

Ptr = pointer to ASCII ending

G. Error Codes:

None

H. Messages:

None

I. Length:

217_8 or 143_{10} words

J. Source:

STEM2A ALGOL

K. File References:

None

3.2.7 Output Command Interpretation

3.2.7.1 INIOUT

Purpose

To initialize the procedures OUT. and IN.

Description

A. Operation: INIOUT ( )

INIOUT initializes IN. and OUT. by setting a few character strings used by those procedures.

B. Procedures calling INIOUT:
INIT2

C. Procedures called by INIOUT:
FRALG, .C.ASC

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
20_8 or 16_{10} words

J. Source
INOUT ALGOL

K. Files Referenced:
None

3.2.7.2 OUT.

Purpose

To interpret OUTPUT command

Description

A. Operation: Code = OUT. (Ptr)

CLP calls OUT. to interpret a user's request for output. OUT. examines the command line for field specifications. Fields are specified either by their number or by their name. When it finds one, it either sets a bit in the output request list pointer (ORL.) or it enters the field number for that field in the output request list, depending on the field requested. An output request always wipes out the previous output request.

If the user asks for a non-existent field, OUT. prints an error message and returns to CLP. If the user specifies more than 10 fields, OUT. accepts the first 10 and ignores the rest. If the field number is 90 and the user's console is not a CRT, OUT. prints an error message. If there are no arguments, all of the bits in the ORL except those in the address portion are zeroed out. This indicates a request for output with a default specification (fields 24, 21, 23, 47). If the previous command was "output 90" or "output text", and this current request is not for field 90, OUT. sets TEXTX (POT.) to 0 and transmits a form feed via WRFLXA. If SNX(SST.) is not true, OUT. sets GCE (SST.) to true and returns to CLP.

The special fields are handled in the following way:

field 1:	set bit 0 of ORL.
field 74:	set bit 1 of ORL.
field 75:	set bit 2 of ORL.
field 76:	set bit 18 of ORL.
field "all"	set bit 19 of ORL.
field 90:	set bit 20 of ORL.
no field:	zero out bits 0-20.

B. Procedures calling OUT.:

CLP (via CALLIT)

C. Procedures called by OUT.:

ASCINT	FREE	TYPEIT
COMPUL	FREZ	
COPY	LEGFLD	
FLDNAM	NEXITM	

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
ORL. (CL.)	Output request list		x
TEXTX (POT.)	Text pointer		x
BLIP (POT.)	Blip characters	x	
SNX (SST.)	Search not executed	x	
GCE (SST.)	Go command exists		x
FSONX (SST.)	FSO not executed		x

E. Arguments:

Ptr: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

Code = -3: illegal field name
-5: "text" not a valid field for this console

H. Messages:

1. "x is not a legal designation. Check for typing errors. See Part 15 of Guide for full list of types of catalog information." ($IN. 3$)

2. "See Part 8 of Guide for details on correct use of OUTPUT command. Please rephrase your request." ($OUT.3a$)

3. "The OUTPUT command accepts only 10 fields. You may output the remainder on a subsequent request." ($OUT.4$)

4. "To see TEXT you must say output fiche and get the fiche location of the text." ($OUT.5$).

I. Length:

260_8 or 176_{10} words

J. Source:

INOUT ALGOL

K. Files Referenced:

None

3.2.7.3 IN.

Purpose

To interpret RESTRICT command

Description

A. Operation Code = .IN.

IN. processes a user's request for a secondary search. IN. expects two arguments in the command line pointed to by Ptr: the first is the name or number of the field to be searched and the second is the character string which FSO will search for . IN. stores an ASCII pointer to the string and the binary equivalent of the field number in a word in the field search list. The ten-word list can hold ten search specifications. After FSO has performed the searches, the strings are returned to free storage and the list is zeroed out .

If the first argument of the user request cannot be identified as a field, or if the second argument is missing, IN. prints an error message and returns to CLP. If the user has made more than 10 search requests since the last output request, IN. prints an error message. Otherwise, IN. stores an ASCII pointer to the string in the next available word in the Field Search List and puts the field number in bits 3-9 of that same word.

B. Procedures calling IN.:

CLP (via CALLIT)

C. Procedures called by IN.:

ASCINT	FREE	TYPEIT
COMPUL	FREZ	
COPY	LEGFLD	
FLDNAM	NEXITM	

D. COMMON references:

Name	Meaning	Interrogated?	Changed?
FSL. (CL.)	Field search list		x
BLIP(POT.)	Blip characters	x	

E. Arguments

Ptr: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

Code	= - 1:	field search list is full
	= - 2:	no search string
	= - 3:	illegal field name

H. Messages:

1. "Only 10 RESTRICT restrictations allowed on a search." "Please begin a new search." ($IN.1$)
2. "You have not fully specified your RESTRICT command. See Part 9.5 of Guide for details on correct use of RESTRICT command."

   "Please rephrase your request." ($IN.2$)
3. "X is not a legal designation. Check for typing errors. See Part 15 of Guide for full list of catalog information." ($IN.3$)
4. "See Part 9.5 of Guide for details on correct use of OUTPUT command."
   "Please rephrase your request." ($IN.3a$)

I. Length:

240_8 or 160_{10} words

J. Source:

INOUT ALGOL

K. Files Referenced:

None

3.2.8 Output Command Controls

3.2.8.1 FSO

Purpose

To output Catalog information

Description

When a user requests output by means of the OUTPUT command, the system will store his specifications and then call FSO.

FSO will transmit to the user four kinds of data:

1. Document numbers, obtained from the reference list.
2. Catalog information from CR** INTREX.
3. Fiche locations from FICHE DIRECT.
4. Texts of articles via microfiche.

I. Document Numbers

If the user specifies "Output 1", FSO will print the document numbers that it finds in the reference list. Since the catalog is not referenced, this form of retrieval is inexpensive.

II. Catalog Information

FSO scans the catalog index, CATDIR INTREX, for the locations of the requested documents in the catalog. FSO reads a document's record into core and prints the fields that have been requested.

There are several special features associated with this basic procedure:

a. Secondary searches: If the user has given the RESTRICT command, FSO will, for each document, compare the search string with the specified fields. If the match succeeds, FSO tells the user this and tries the search on a new document. FSO outputs the requested information.

b. Field 74 (MATCH): FSO uses the term numbers which it finds in the reference words to print out only those terms in field 73 which met the conditions of the search request.

c. Field 75 (STANDARD): FSO prints the Author, Title, Corporate Author and Location fields in a format similar to a bibliographic entry.

d. Field 76 (Normal): This is the default option if the user gave no field specifications in his output request. FSO prints the Author, Title, Corporate Author and Location fields as four separate fields.

e. ALL: If "ALL" is specified, all of the fields of each document will be printed.

III. Fiche Locations

If the user has specified "5" or "fiche" in his output request, FSO will call TRETRI, which will reference FICHE DIRECT to give the user the fiche address of each document after any other requested catalog information has been given.

IV. Text Access

If the user has specified "90" or "text", FSO will call TRETRI, which will evoke the automatic text retrieval mechanism after any requested catalog information has been given.

A. Operation

1. If CATII(SST.) is true, the typewriter is turned off and CATII OUTPUT is opened. (This is the mode for creating output on disc files for offline printout.

2. The subroutines WHOAMI and GETP are used to determine if the user is working from an INTREX console. If he is, a switch is set so that special symbols in the catalog data will be decoded by SPCTRN.

3. Data is extracted from the output request list pointer. Flags are set for the fields Document, Fiche, Standard, Normal, all and Text.

4. GCE(SST.) and FSONX(SST.) are set to false.

5. If there is no document list, FSO prints an error message and returns to SUPER.

6. If there are secondary search specifications and field 90 has been requested, FSO prints an error message and returns to SUPER.

7. GETLIS is called to initialize the document list.

8. FSO obtains a document number by calling the sub-procedure FETCH:

   a. If secondary searches are being performed and the immediately previous search was successful, FETCH moves the previously current document number to a new document list. If the new list contains 432 references, it is written out in the Dump File.

b. If the list of references which are in core is exhausted, and if there are no more references on disk, FETCH returns to FSO with a value of 0. If there are more references on disk, 432 of them are read into core.

c. The document number of the current reference word is stored as the value of FETCH. FETCH increments the reference list pointer by one and returns to FSO.

9. If the document number returned by FETCH is the same number that it previously returned, FETCH is called again (Step 8).

10. If FETCH returns a zero, FSO transfers to its exit routine Step 25.

11. If TEXTX(POT.) greater than zero and the document number returned by FETCH is greater than TEXTX(POT.) FSO branches back and calls FETCH again (Step 8). This process continues until a document number lower than TEXTX(POT.) is returned by FETCH. This procedure is followed because the user must cycle through "output text" requests (one request for each document) as contrasted with other output requests which give all documents at once.

12. A carriage return is typed.

13. If there is no secondary search and only field (Document) is requested, the header for the current document is printed and control is transferred back to Step 8.

14. The current field list pointer is set to point to the complete list of fields, the standard fields or the list the user has entered.

15. GETINT is called. GETINT will scan the directory CATDIR CATS2 (POT.) for the pointers of the first 50 documents and read into memory the first of these documents. If GETINT returns an error code, FSO will print the header and an error message, reset TYPEIT by a call to INDENT, and branch back to Step 8.

16. If a secondary search has been requested, FSO will match each specified field against the corresponding search string. If the field is missing from the document, that part of the search is arbitrarily ruled to be successful. If any part of the secondary search fails, HEADER is called. HEADER will report that the search failed and then transfer control to Step 8.

17. If the CATII option is on, the typewriter will temporarily be turned back on to type a header.

18. If field 75 was requested, fields 24, 21, 23, and 47 will be printed together as a combined field and header. If the INTREX console is being used, SPCTRN will be called to decode any asterisk expressions in field 24.

19. The standard header is printed if field 75 was not requested.

20. If the current list of fields is not empty, FSO will step through list, printing each of these fields:
    a. GETFLD is called to obtain a pointer to a requested field;
    b. SUBHEAD is called to print the subheading for the field;
    c. Either SPCTRN or TYPEIT is called to print the field;
    d. A new field number is obtained from the current list of fields, and control is transferred to Step a.

21. If field 74 has been requested, the following steps are performed:
    a. GETFLD is called to retrieve a pointer to field 73;
    b. The term number is extracted from the current reference word;
    c. If term number is 77, which means that the list was generated by the document command, or if there is an author search form but not a title o[illegible]ject form also, FSO prints a message explaining that a request for field 74 is nonsensical;
    d. FSO extracts the correct term from field 73, using the last two characters of a term,");" as a delimiter;
    e. The term is printed, using either TYPEIT or SPCTRN;
    f. FETCH is called. If it returns a new document number, control is transferred out of the field 74 routine, otherwise, control is transferred to Step b.

22. If field five has been requested, TRETRI is called with a second argument of 0.

23. If field 90 has been requested, TRETRI is called with a second argument of 1. The value that TRETRI returns is stored in TEXTX(POT.).

24. Control is transferred to Step 8.

25. A concluding message is printed. The free storage used by GETINT is returned and INDENT is called with an argument of zero. If a new list has been generated by means of a secondary search, NEWPT will establish it as the current list. Finally, control is returned to SUPER.

B. Procedure calling FSO:
   SUPER

C. Procedures called by FSO:

CLOSE	GETFLD	INIDSK	TYPEIT
DIST	GETINT	MATCH	WHOAMI
	GETLIS	RDWAIT	WRWAIT
FIELDS	GETP	SPCTRN	
FLDNAM	INC	STANDL	
FRALG	INC1	TABENT	
FSOCLN	INDENT	TRETRI	

D. COMMON References

Name	Meaning	Interrogated?	Changed?
BLIP (POT.)	blip characters	x	
TEXTX(POT.)	text pointer	x	
VERBOS (POT.)	TYPEIT mode	x	
COMBF1(POT.)	common buffer	x	
TOTBLK(POT.)	Dump File pointer	x	
COMBF4(POT.)	common buffer	x	
DFN1(POT.)	Dump File name 1	x	
CATII(SST.)	off line output	x	
GCE (SST.)	go command exists		x
FSONX(SST.)	FSO not executed		x
SSF. (CL.)	subject search form	x	
TSF. (CL.)	title search form	x	
ASF. (CL.)	author search form	x	
ORL.(CL.)	output request list	x	
FSL. (CL.)	field search list	x	
RRL.(CL.)	resultant reference list	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

Code = -1 no reference list

H. Messages:

1. "Search and Output completed. You may now see other catalog information from this/these document/documents by making an output request (for information on how to do this, see Part 8 of the Guide or type info. 8.

   You may also select a subset of these documents by making a RESTRICT request (see Part 9.5). Otherwise, you may make a new search (see Part 2) or make other requests (see Part 1)".

   (/fso1/, /fso2a/, /fso2b/, /fso2c/, /fso2d1/, fso2d2/, /fso2e/, /fso5/); [short form is null]

2. "No documents found. You may make a new search (see Part 2 of Guide, or type info 2) or make other requests (see Part 1)." [No documents found] (/fso4/)

3. "Your last active list has been retained." ($fso6$)

4. "The catalog record for this document can not be retrieved at this time. Error code = 6 ($get1$)

5. "INTREX is unable to print this field" ($get2$)

6. "field empty" (/empmes/)

7. "MATCH invalid for this request" (/doc6/)

8. "NO MATCH" (/doc5/)

I. Length:

2350_8 or 1256_{10} words

J. Source:

FSO ALGOL

K. Files Referenced:

DUMnnn FILE
NAMnnn FILE
SInnn - date-
AInnn - date-

3.2.8.2 NEWPT

Purpose

To finish new reference list

Description

NEWPT is called by FSOCLN to finish up the action of FSO either after an interrupt or after FSO has processed all of the current documents.

A. Operation

1. NEWPT calls FCLEAN, which will clean up secondary search specifications, if there are any.
2. If there has been a secondary search and entries have been generated for a new reference list, NEWPT writes out these entries on the Dump File. A three-word reference list pointer is created for them, making them the current list.
3. If the original reference list which FSO used was on disk, its file is closed.
4. If Intrex is in the CATII mode, the file CATII OUTPUT is closed and the typewriter is turned on.

B. Procedures calling NEWPT:

FSOCLN

C. Procedures called by NEWPT:

CLOSE, FCLEAN, INIDSK, TABENT, WRWAIT

D. COMMON References:

		Interrogated?	Changed?
DFLNI (POT.)	Dump File name 1	x	x
TOTBLK(POT.)	Dump File pointer	x	x

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

200_8 or 128_{10} words

J. Source:

FSO ALGOL

K. Files Referenced:

DUMnnn FILE

3.2.8.3 INIFLD

Purpose

To initialize FSO

Description

A. Operation:

INIFLD is called by SEGINT to preset variables used by FSO.

B. Procedures calling INIFLD:

SEGINT

C. Procedures called by INIFLD:

.C.ASC

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

H. Messages:

None

I. Length:

120_8 or 80_{10} words

J. Source:

FSO ALGOL

K. Files Referenced:

None

3.2.8.4 GETINT

Purpose

To initialize GETFLD

Description

A. Operation Code = GETINT (Lptr, Llen)

FSO calls GETINT twice for each document. On the first call, GETINT looks up in CATDIR INTREX the pointers for up to 50 catalog pointers. GETINT uses this list of pointers to read the current document into core. When the list is exhausted, another batch of 50 is read. After all of the desired fields have been extracted, GETINT is called with zero arguments to clean up.

GETINT has two arguments Lptr and Llen. Lptr points to the current document on the reference list and Llen is the length of the remainder of the list. If Lptr is zero, GETINT interprets this as a request to clean up and returns the free storage last used by GETFLD. If GETINT's list of catalog pointers is empty, it will step through this list of document numbers and replace each one with a catalog pointer extracted from CATDIR INTREX.

GETINT will use the current catalog pointer to read the catalog record into memory. If there is no pointer for a document, GETINT returns a -1. If the pointer points beyond a segment of the catalog, -4 is returned. If the catalog record is larger than 864 words, a -6 is returned. If a fence cannot be found in word five of the record, a -5 is returned. If GETINT successfully reads in a catalog record, it returns to FSO with a value of zero.

B. Procedures calling GETINT:

FSO, FSOCLN

C. Procedures called by GETINT:

CLOSE, FRET, OPEN, RDWAIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
CATS2(POT.)	Name2 of Catalog	x	

E. Arguments:

Lptr:	pointer to resultant reference list
Llen:	Length of remainder of list

F. Values:

Code = 0

G. Error Codes:

- -1: CATDIR INTREX does not contain a pointer for the record.
- -4: A catalog segment has been read beyond its end of file.
- -5: The beginning of the header of a catalog record cannot be found.
- -6: There is not enough room in core storage for the record.

H. Messages:

None

I. Length:

411_8 or 265_{10} words

J. Source:

GETFLD ALGOL

K. Files References:

CATDIR	INTREX
CRnnn	INTREX

3.2.8.5 FSOCLN

Purpose

To reset FSO parameters.

Description

FSOCLN calls routines which finish up the work of FSO. It is called at the end of FSO or it is called by LISTEN after the user transmits an interrupt,

A. Operation

NEWPT is called to clean up after a secondary search. GETINT is called to return free storage. INDENT is called to reset the margin to zero.

B. Procedures calling FSOCLN

FSO, LISTEN

C. Procedures called by FSOCLN:

GETINT, INDENT, NEWPT

D. COMMON references:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

14_8 or 12_{10} words

J. Source:

FSOCLN ALGOL

K. Files Referenced:

None

3.2.8.6 GETFLD

Purpose

To get a field from catalog

Description

A. Operation: Ptr = GETFLD (Fldno, Expand)

GETFLD has been prepared by a call to GETINT, which reads the catalog record into core. GETFLD has two arguments, Fldno and Expand.Fldno is the desired field and Expand is a Boolean switch which indicates whether or not an encoded field should be decoded.

GETFLD searches for the field Fldno in its list of fixed fields. If it finds it there, GETFLD transfers to a specialized routine which extracts the field from the first four words of the catalog record. If Expand is false, as it will be if FSO wants the field for a secondary search, GETFLD converts the binary value that it has extracted into an ASCII representation of the cataloger's code that the value represents. If FSO has requested the field for purposes of output, GETFLD will use the binary value to construct a TYPEIT message label.

If GETFLD cannot find the field on its fixed field list, it will scan the header of the record for the Fldno. If it finds Fldno in the decrement of word in the header, then the address portion will contain the byte count of the last byte of that field. The beginning of the field can be derived using the byte count of the previous field. GETFLD uses this data to find the beginning of the field and its length. An area one word more than half of the length of the field in bytes is allocated for conversion from digram to ASCII.. If there is not enough room, GETFLD returns with an error code of -6. The table in GETTAB is used to convert the field. A pointer is returned to the converted field.

B. Procedure calling GETFLD:

FSO

C. Procedures called by GETFLD:

COPY	FREZ	INC	TYPEIT
DEFBC	GETINC	INC1	
FREE	GETSET	INTASC	
FRER	GETTAB	PUT	
FRET	INC	PUTINC	

D. COMMON References:

None

E. Arguments:

Fldno: number of field requested
Expand: if TRUE, then decode fixed fields

F. Values:

Ptr = ASCII pointer to field

G. Error Codes:

Ptr = -6: not enough room for digram-to-ASCII conversion
Ptr= -3: field missing

H. Messages:

None

I. Length:

1240_8 or 672_{10} words

J. Source:

GETFLD ALGOL

K. Files Referenced:

None

3.2.8.7 GETTAB

Purpose

To get digram-ASCII conversion table

Description

A. Operation: Ptr= GETTAB( )

GETTAB returns as its value the location of a 256 word table which is used by GETFLD to convert a catalog field from digram to ASCII.

B. Procedures calling GETTAB:

GETFLD

C. Procedures called by GETTAB:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

Ptr= Location of table

G. Error Codes:

None

H. Messages:

None

I. Length:

404_8 or 260_{10} words

J. Source:

GETTAB ALGOL

K. Files Referenced:

None

3.2.8.8 TRETRI

Purpose

To retrieve fiche no. or text

Description

A. Operation Docno = TRETRI (Docno, Flag)

TRETRI opens the FICHE DIRECT, reads the entry for Docno and closes the file. If Flag is one, TRETRI sends to the terminal, via WRFLXA, a special character string containing the address of the text to be retrieved and special control characters. These control characters are interpreted by special logical circuitry attached to the terminal which sends the appropriate request to the text-access system. TRETRI returns to FSO with Docno as its value. If there is no fiche for Docno, a value of -Docno is returned.

If Flag is 0, TRETRI prints via TYPEIT the location of the fiche. If the text of the document is not in the fiche collection, a special TYPEIT message explains where it may be found.

B. Procedures calling TRETRI:

FSO

C. Procedures called by TRETRI:

DERBC	RDWAIT	SETFUL
OPEN	SETBCD	WRFLXA

D. COMMON References:

Name	Meaning	Interrogated	Changed?
VERBOS(POT.)	TYPEIT mode	x	
COMBF1 (POT.)	Common buffer	x	

E. Arguments:

Docno:	document number
Flag:	0: fiche locating requested
	1: text requested

F. Values:

Docno = document number (fiche available)
Docno = - document number (fiche not available)

G. Error Codes:

None

H. Messages:

1. "This document is not yet available from the text-access subsystem. You may see a hard copy by asking a member of the Intrex staff for it." (/text1/)

2. "Text is available only for the individual parts of this document (that is, for articles or chapters) which were separately documented" (/text2/).

3. "Request a hard copy of text from a member of the Intrex staff. Hard copy is found at library with code name x. See Part 15.11 of the Guide for explanation of code." (/text3/, /text5a/, /text 5b/)

I. Length:

535_8 or 349_{10} words

J. Source:

TRETRI ALGOL

K. Files Referenced:

FICHE DIRECT

3.2.8.9 SPCTRN

Purpose

To translate special characters

Description

SPCTRN is used to translate the ASCII representation of special graphical symbols, such as Greek letters, into special code sequences which, when transmitted to an INTREX console, will appear as graphical symbols.

A. Operation

If the user is at an INTREX console, FSO will call SPCTRN(Ptr) printout fields instead of TYPEIT. SPCTRN will scan the field pointed to by Ptr, looking for special graphical representations, which are always bracketed by Asterisks. When it finds a special string, it calls TABLK, which translates it into the special codes that are used by the INTREX console. Ordinary ASCII substrings within the field are transmitted by SPCTRN by means of TYPEIT. A detailed description of SPCTRN, and possible refinements of it, is given in Reference 12.

B. Procedures calling SPCTRN:

FSO

C. Procedures called by SPCTRN:

COMPAR	FIND	TABLK
COMPUL	INC	TYPEIT
DIST	INCHAR	

D. COMMON References:

None

E. Arguments:

Ptr: ASCII pointer to catalog field

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "DISPLAY PROGRAM HAS ENCOUNTERED AN ERROR." "THIS PART WILL BE DISPLAYED AGAIN WITHOUT TRANSLATION." (preset)

I. Length:

$321_8 = 209_{10}$ words

J. Source:

SPCTRN ALGOL

K. Files Referenced:

None

3.2.8.10 TABLK

Purpose

To look up specially coded characters

Description

A. Operation: Chars = TABLK (Ptr, Flag)

TABLK translates the representation of a special character into the binary codes that will activate the Intrex console to display that character. This process involves the use of four tables. DTABL contains an entry for each special graphic. Each word contains a pointer to the representation of the graphic, stored in STBL, and a pointer to the binary equivalent, stored in OTBL. The entries in DTABL, and consequently the entries in STBL and OTBL, are ordered according to the length of the representation. The table DRCT indexes the table DTABL. Each word in DRCT points to the pointers in DTABL which apply to a representation of a given length.

Using the length specification in the decrement of Ptr as in index, TABLK extracts a pointer from DRCT. TABLK steps through the pointers in DTABL, looking for the string pointed to by Ptr in the table STBL. If it finds the string, it looks up the binary equivalent in OTBL. This value is transmitted to the console by a call to WRHGH.

B. Procedures calling TABLK:

SPCTRN

C. Procedures called by TABLK:

COMPAR	FREZ	INCHAR	WRHGH
COMPUL	GET	OTBL	
FRET	INC	STBL	

D. COMMON References:

None

E. Arguments:

Ptr: ASCII pointer to special graphic representation

Flag = 1: string translated

0: string not translated

F. Values:

Chars = number of display positions used by special character

G. Error Codes:

None

H. Messages:

None

I. Length:

722_8 or 466_{10} words

J. Source:

TABLK ALGOL

K. Files Referenced:

None

3.2.8.11 STBL

Purpose

To define special symbols

Description

A. Operation: Addr = STBL ( )

STBL returns the address of a packed table of all of the ASCII equivalents of the special graphical characters. TABLK searches this table for the substring which SPCTRN has extracted from a field.

B. Procedures calling STBL:
TABLK

C. Procedures called by STBL:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
Addr = location of table

G. Error Codes:
None

H. Messages:
None

I. Length:
267_8 or 183_{10} words

J. Source:
OTBL ALGOL

K. Files Referenced:
None

3.2.8.12 OTBL

Purpose

To define binary codes

Description

A. Operation: Addr= OTBL( )

OTBL returns as its value the location of a table containing all of the special character sequences which must be transmitted to an INTREX console by TABLK in order to display special graphics.

B. Procedures calling OTBL:
TABLK

C. Procedures called by OTBL:
None

D. COMMON references:
None

E. Arguments:
None

F. Values:
Addr= location of table

G. Error Codes:
None

H. Messages:
None

I. Length:
137_8 words or 95_{10} words

J. Source:
OTBL ALGOL

K. Files Referenced:
None

3.2.8.13 WRHGH

Purpose

To transmit special codes.

Description

A. Operation: WRHGH (String, Length)

WRHGH is a Fap-coded procedure which is used as a means of calling the CTSS A-core procedure by the same name. The CTSS procedure will transmit to the user's console the 12-bit binary string defined by the arguments String and Length.

B. Procedures calling WRHGH:
TABLIC

C. Procedures called by WRHGH:
WRHGH

D. COMMON References:
Name

E. Arguments:

String:	location of character string
Length:	length of string in words

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
17_8 or 15_{10} words

J. Source:
WRHGH FAP

K. Files Referenced:
None

3.2.8.14 INIRNG

Purpose

To initialize RANGE, ATLCLN

Description

A. Operation: INIRNG( )

INIRNG is called by SYSGEN via SEGINT at system generation time. Five variables are set; two of them are ASCII pointers, which are created by calls to .C.ASC. The procedure returns itself to free storage via FRALG.

B. Procedures calling INIRNG:
SYSGEN (via SEGINT)

C. Procedures called by INIRNG:
FRALG .C.ASC

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
23_8 or 19_{10} words

J. Source
RANGE ALGOL

K. Files Referenced:
None

3.2.8.15 RANGE

Purpose

To interpret RANGE command

Description

RANGE is called by CLP to set up an attribute list in the INTREX data structure, based on the user's RANGE specifications.

A. Operation Code = RANGE(Ptr)

RANGE processes a user command by calling the internal procedures INIR, SBSRCH, NO.ATL, GETLIS, NUMOK, NXTARG and FRMRAL.

1. RANGE calls INIR, which
   a) obtains the location of the array of pointers to the ASCII names of the different ranges.
   b) obtains the location of the subject search form (SSF.(CL.)).
2. RANGE calls SBSRCH, which prints an error message if a subject search has not been requested or if the search has already been performed.
3. RANGE calls NO.ATL, which prints an error message if there are already range attributes associated with the search request.
4. RANGE calls GETLIS, which extracts the arguments in the command line which follow the RANGE command, and builds a list of range specifications.
   a) GETLIS calls NXTARG, which uses NEXTITM to return an [illegible] to the variable ARGPTR. If there are no more [illegible]s, NXTARG assumes the Boolean value of FALSE.
   b) [illegible]TARG is true, GETLIS [illegible]lls NUMOK to check the argument and convert it to its binary equivalent.
      (1) NUMOK uses ASCINT to convert the argument to ASCII. If this yields a value outside the range of values for RANGE, the argument is compared with the ASCII names of possible values.
      (2) If the argument is illegal, NUMOK prints an error message and returns a value of FALSE.
   c) If NXTRAG and NUMOK are TRUE, GETLIS adds the binary equivalent of the RANGE (0-4) to the next sequential location in an array.
   d) GETLIS loops back to Step a for another argument. If there are no more, the unused portion of the array is returned to free storage and GETLIS returns to RANGE with a pointer to the array as its value.

5. RANGE calls FRMRAL, which copies the array of ranges into a new array and returns the old one to free storage. The new array has a 7 in the first word instead of a range value.

6. A pointer to the array is stored in the ATL. slot of the subject search form.

7. RANGE returns to CLP with a value of zero. If there have been any errors, RANGE returns a -1, which terminates the processing of the user's command line by CLP and prevents the execution of a search request.

B. Procedures calling RANGE:

CLP (via CALLIT)

C. Procedures called by RANGE:

ASCINT, COMPUL, FRALG, FRET, FREZ, NEXITM, RNGNAM, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
SSF.(CL)	subject search form		x

E. Arguments:

Ptr: ASCII pointer

F. Values:

Code = 0

G. Error Code:

None

H. Messages:

1. "X is not a legal RANGE. Check for typing errors. See Part 9.2 of the Guide for details of correct usage of the RANGE command." ($raccl$)
2. "The RANGE command may only be used along with a SUBJECT search as explained in Part 9.2 of the Guide," ($raerr2$)
3. "You may use the RANGE command only once on each SUBJECT search as explained in Part 9.2 of the Guide.", ($raerr3$)

I. Length:

600_8 or 384_{10} words

J. Source:

RANGE ALGOL

K. Files Referenced:

None

3.2.8.16 ATLCLN

Purpose

To clean up attributes

Description

A. Operation: ATLCLN(Ptr)

ATLCLN is called by CLEANP to return the attribute list to free storage.

ATLCLN calls FRET with the arguments Len and Loc, where Len is one more than the decrement of Ptr, and Loc is the address portion of Ptr. Ptr is then set to zero.

B. Procedures calling ATLCLN:
CLEANP (via CALLIT)

C. Procedures called by ATLCLN:
FRET

D. COMMON References:
None

E. Arguments:
Ptr: word pointer to attribute list

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
24_8 or 20_{10} words

J. Source:
RANGE ALGOL

K. Files Referenced:
None

3.2.9 Miscellaneous Command Controls

3.2.9.1 INIVRB

Purpose

To initialize miscellaneous procedures

Description

A. Operation: INIVRB( )

INIVRB initializes variables for LONG, SHORT, COMENT, LIBRY, WRT and GO.

B. Procedures calling INIVRB:

SEGINT

C. Procedures called by INIVRB:

.C.ASC, FRALG

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

23_8 or 19_{10} words

J. Source:

VERBOS ALGOL

K. Files Referenced:

None

3.2.9.2 COMENT

Purpose

To record a user's comment

Description

A. Operation: Code = COMENT (Ptr)

COMENT calls WRT to add the comment pointed to by Ptr to the Monitor File.

B. Procedures calling COMENT:
CLP (via CALLIT)

C. Procedures called by COMENT:
WRT

D. COMMON References:
None

E. Arguments:
Ptr: ASCII pointer to user's comment

F. Values:
Code = 0

G. Error Codes:
None

H. Messages:
None

I. Length:
15_8 or 13_{10} words

J. Source:
VERBOS ALGOL

K. Files Referenced:
None

3.2.9.3 LIBRY

Purpose

To make a library request

Description

A. Operation: Code = LIBRY (Ptr)

LIBRY calls WRT to print a library request in the Monitor File. This is distinguished from COMENT by setting the variable Lib to true.

B. Procedures calling LIBRY:

CLP (via CALLIT)

C. Procedures called by LIBRY:

WRT

D. COMMON References:

None

E. Arguments:

Ptr: pointer to request fo: ibrary assistance

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

None

I. Length:

17_8 or 15_{10} words

J. Source:

VERBOS ALGOL

K. Files Referenced:

None

3.2.9.4 WRT

Purpose

To write a user message

Description

WRT extracts a user message from a command line and writes it in the Monitor File via ASIDE.

A. Operation: WRT(Ptr)

1. NEXITM extracts the message up to the next slash.
2. If LIBRY has set Lib to true, WRT prints via ASIDE: ***********LIBRARY REQUEST*********
3. WRT prints the message and then prints a line of asterisks

B. Procedures calling WRT:

COMENT, LIBRY

C. Procedures called by WRT:

ASIDE, NEXITM

D. COMMON References:

None

E. Arguments:

Ptr: ASCII pointer to message

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

32_8 or 26_{10} words

J. Source:

VERBOS ALGOL

K. Files Referenced:

None

3.2.9.5 LONG

Purpose

To enter long mode

Description

LONG sets up TYPEIT to use the file of long messages rather than the short one.

A. Operation Code = LONG ( )

1. VERBOS (POT.) is set to 1 (long mode).
2. INITYP(DFLN1(POT.)) is called; the parameter DFLN1(POT.) contains the first name of the long message file (LMFILE).
3. If the system is in CLAMP mode, the value of the second word of the Password File is changed to the BCD string "ΔΔLONG". (If an I/O error occurs, the system prints a message and goes dormant). If the system is not in CLAMP mode, the value of the special A-core word is changed to "ΔΔ LONG"

B. Procedures calling LONG:

CLP (via CALLIT)

C. Procedures called by LONG:

BUFFER, CLOSE, INITYP, OPEN, SETWRD

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
VERBOS(POT.)	TYPEIT mode		x
DFLN1(POT.)	Long mess. file name 1	x	
PFN1(POT.)	Password File	x	
COMBF6(POT.)	Common buffer	x	
CLAMP(SST.)	Hold mode	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

None

I. Length:

67_8 or 55_{10} words

J. Source:

VERBOS ALGOL

K. Files Referenced:

None

3.2.9.6 SHORT

Purpose

To enter short mode

Description

A. SHORT works exactly like LONG except that it sets up TYPEIT to use the short message file.

A. Operation Code = SHORT ( )

1. Verbos (POT.) is set to 0.
2. INITYP is called with an argument of DFSN1(POT.), which contains "SMFILE".
3. The second word of the Password File (or A-core) is set to "SMFILE".

B. Procedures calling SHORT:

CLP (via CALLIT)

C. Procedures called by SHORT:

BUFFER, CLOSE, DORMNT, INITYP, OPEN, SETWRD, TYPEIT.

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
VERBOS(POT.)	TYPEIT mode		x
DFSN1(POT.)	Short mess. file	x	
PFN1(POT.)	Password file	x	
COMBF6(POT.)	Common buffer	x	
CLAMP(SST.)	Hold mode	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

None

I. Length:

76_8 or 62_{10} words

J. Source:

VERBOS ALGOL

K. Files Referenced:

None

3.2.9.7 INFO

Purpose

To print a section of the on-line guide

Description

A. Operation: Code = INFO (Ascptr)

CLP transfers control to INFO when it finds the command "info" in the user's command line. INFO uses NEXITM to extract an argument. If no argument is specified, "1" is assumed. A guide file message label is constructed by converting the argument to BCD and appending it to the letter "A". The Guide Message File is opened by a call to to INITYP. TYPEIT is called with the constructed label as an argument. If TYPEIT cannot find this label in the directory for the guide, it zeroes out the erroneous argument and returns control to INFO without typing its usual error message. INFO calls INITYP again to close the Guide File and reopen the Message File. If the argument given to INFO was zeroed out, an error message is printed.

B. Procedures calling INFO:

CLP (via CALLIT)

C. Procedures called by INFO:

CHKNUM, CTSIT6, FRET, INITYP, NEXITM, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
INFOX(SST.)	INFO requested		x
INFO1(SST.)	INFO1 requested		x
INFO2(SST.)	INFO2 requested		x
DFLN1(POT.)	Long message file name	x	
DFSN1(POT.)	Short message file name	x	

E. Arguments:

ASCPTR: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

1. "X is not a valid argument to the 'INFO' command ($infoer$)

I. Length:

120_8 or 80_{10} words

J. Source:

SQUIRE ALGOL

K. Files Referenced:

None

3.2.9.8 SEEMAT

Purpose

To display results of Inverted File search upon COUNT command.

Description

CLP calls SEEMAT in response to the user command "COUNT" or "C". SEEMAT shows how the lists of documents generated by each term in a search specification are combined to form a resultant list.

The search words are printed out in the order in which they were looked up. Two numbers follow each word: the total number of documents associated with that word and the total number of documents which match on all of the words looked up thus far. If the document list is empty, or if it has been reduced in size by a RESTRICT operation, the message "current list size is x" will appear.

A. Operation: Code = SEEMAT( )

1. If the user is in LONG mode, SEEMAT will print a header.
2. SEARCH re-ordered the subject and title terms so that the shortest lists would be looked up first. SEEMAT obtains these re-ordered lists via RESUB(CL.) and RETIT(CL.).
3. If SEARCH did not look up a word (because the search had failed before SEARCH got to this word), SEEMAT prints "Not looked up" after it.

B. Procedures calling SEEMAT:

CLP (via CALLIT)
SUPER (via CALLIT)

C. Procedures called by SEEMAT:

COPY, GETEND, INC, INC1, INTASC, PUT, TOTTIM, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
VERBOS(POT.)	TYPEIT mode	x	

E. Arguments:

None

F. Values:

Code = 0

G. Error Codes

None

H. Messages:

1. "Current list size is <u>x</u>" (preset)
2. "Not looked up" (preset)
3. "WORD(STEM-ENDING) NO. OF DOCUMENTS THAT MATCH THIS STEM, ALL STEMS SO FAR."

(/sm01/), (null)

I. Length:

612_8 or 402_{10} words

J. Source:

SEEMAT ALGOL

K. Files Referenced:

None

3.3 List Manipulation Logic

3.3.1 Document Selection

3.3.1.1 NUMBER

Purpose

To control DOCUMENT command

Description

A. Operation: Code = NUMBER (Ascptr)

The number routine is called directly from CLP (Command Language Processor) as a function of the DOCUMENT (or D) command. It accepts a pointer to the remainder of the command line as in argument which is assumed to contain one or more document numbers that the user would like activated as his current list.

An array declared in the source program is used in holding the document numbers as they are extracted from the command line. This array is currently 50 words in length and is usually sufficient to hold more numbers than can be typed in the two-line limit per command (unless they are all 1- or 2-digit numbers).

Common buffer 5 is used to read into core the current list of references (if there is one, and if it is at least partially on the disk) so that their document numbers might be checked against those that the user selected. If the number requested is found to exist in the current list, the entire reference word is copied from the list into the 50-word array which will become the new list. This preserves such useful data as term and word number of the original search words and allows OUTPUT MATCH commands to work on the new reduced list.

If the desired document number is not already in the current list, a pseudo-reference word is constructed with a special code (77) in the term number position. This makes OUTPUT MATCH an invalid request on this document. It is possible that the user will present a string of numbers, some of which will be on the current list and some not. In this case, the resulting list will contain a mixture of real and pseudo-reference words. OUTPUT MATCH will then partially work for the user and a COUNT command will produce data about the last search (whose list is partially retained).

Before being compared to the current list and stored in the new one, each document number must be read from the command line (via a call to NEXITM) and converted from ASCII characters to a binary number (using ASCINT). If this converter is fed an argument which is non-numeric or a number greater than 32,767, an error message informs the user that he has used an "improper document number", and NUMBER loops back to see if there are any others to be found on the command line.

If no current list exists (RRLE(SST) is false), the dummy term number is inserted and the pseudo-reference is deposited into the new list. If a current list does exist, a call to GETLIS (a sub-procedure of ANDER residing in BOOL ALGOL) will provide pointers and counts to that list, reading the first buffer-full from the disk, if necessary. Then each reference word of the old list is extracted by calling a sub-procedure within NUMBER named FETCH. This routine keeps track of when the core-stored references of the list are exhausted and, at that time, reads in more via RDWAIT. If the list being read is from an Inverted File segment, an adjustment of one word is made to the list address and reference counts to allow for the section header which starts the block. If all references have been read and processed, FETCH returns a zero value to NUMBER. If FETCH returns the address of a reference word, its document number is pulled out for comparison to the current command document number.

When the document numbers match, the reference of the old list is stored in the new list by calling STORIT (another sub-procedure of NUMBER), and indicators are set which show that a match has been made. STORIT keeps track of where in the 50-word array to deposit the reference, by incrementing a storage index. If this index reaches 50, then a larger block of free storage is needed to hold all the references created by this DOCUMENT command.* This is obtained by calling another sub-procedure named NEDMOR. In this routine, the size of the new-list array

*With the present limit of only two input lines per command, this overflow is extremely unlikely.

is increased by fifty locations and a new array is obtained from free-storage. The references already stored are transferred into the large array.

If the comparison of document numbers shows that the command-line's document number is below the one in the current position of the old reference list, then NUMBER loops back to call FETCH for the next reference in the old list. If the command document is above the list position, then the document is not on the current list and a pseudo-reference is constructed and stored.

After each command document is handled, a test is made for the end of the command (slash). If this has not been reached, NUMBER loops back to call NEXITM again and take the next command document.

If, after all documents in the command have been processed, the new list storage index is zero, an error message (2) informs the user that he has failed to provide any (legitimate) document numbers in his command.

If a new list has been established, then the old one, if it exists, is deleted. When the indicator shows that at least one command document number was found on the old list, DELIST is called to merely remove the old augmented list pointer from the table. The rest of the old search structure is kept to supply maximum information if a COUNT command follows. If the new list is entirely composed of pseudo-references unrelated to a prior search, then CLEANP is called to erase all remnants of the last search. In this case, RLIC(SST.), the indicator which prevents calling of CLEANP at the outset of a new search request, is set to prevent a repeated cleanup of the search structure.

If the Name File or an Inverted File segment was opened by calling GETLIS, it is now closed.

Other wrapup chores include setting the RRLE(SST.) indicator, sorting the new list by descending document numbers (using a sort routine named DNSORT), and entering the new augmented pointer into the pointer table via TABENT. The address of this augmented pointer goes in RRL.(CL.) and the number of documents involved in the list goes into the decrement of RRL. and into DCNT(CL.).

Finally, the document number command indicator, DNC(SST.) is set to alert the command Language Processor that, if the user did not issue an OUTPUT command with his DOCUMENT command, a message should be printed informing him of the size of his new list and how he can get output.* This message cannot be given by NUMBER since there is no way of its knowing if an OUTPUT command follows.

In all cases, NUMBER returns a zero value to CLP.

B. Procedures Calling NUMBER:

CLP (via CALLIT)

C. Procedures Called By NUMBER:

ASCINT, CLEANP, CLOSE, DELIST, DNSORT, FRET, FREE, GETLIS, NEXITM, RDWAIT, TABENT, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBF5(POT.)	Common buffer	x	
RRLE(SST.)	Reference list exists		x
RLIC(SST.)	Restored list in core		x
DNC(SST.)	Doc. command given		x

E. Arguments:

Ascptr: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

None

* Given in "long mode" only.

H. Messages:

1. "JUNK* is an improper document number and has been ignored in processing your command." ($dnerr1$)

2. "You have not included any (legitimate) document numbers in your command." ($dnerr2$)

*The character string supplied by the user is given here.

I. Length:

$716_8 = 462_{10}$

J. Source:

INTPRT ALGOL

K. Files Referenced:

AInnn (date)
SInnn (date)
DUMnnn FILE

3.3.2 Naming and Restoring

3.3.2.1 INIRES

Purpose

To initialize list pointer table and session files

Description

This procedure was originally written to initialize the augmented list pointer table and the Name File into which named lists would be written. It has grown to include the setting up of file names for the Dump File, the Monitor File, and the Password File, and also contains the "central error exit" which will be taken whenever any I/O error is encountered by Intrex. It is called during session initialization.

A. Operation: INIRES(Snam)

The list pointer table is a declared array (currently 120 locations long) which is used to hold (up to 40) augmented list pointers. The address of this array is established here and is available to the procedure TABENT which stores the pointers.

The first names of the three files are chosen by starting with DUM001 and testing (via FSTATE) to see if it has been opened by another user. If so, the name is changed to DUM002, etc., until a file is found which is not open. Once a number has been selected for the Dump File name, it is also used to construct the names of the Monitor and Password files.

The Password File is only created when the system has been started using the "hold" argument, which causes the "dynamic" initialization routine DYNAMO, called prior to the execution of INIRES, to set an indicator in the System State Table, CLAMP(SST). When this flag is on, INIRES first sets up the Password File names, and then writes (using WRWAIT) the contents of ESCODE(POT) into word one of this file. This location of the POT contains the code word issued immediately after the "hold" argument by the person who resumed Intrex. If none was supplied by the user, the word "escape" is used by Intrex as the password and placed into ESCODE(POT) during the execution of DYNAMO.

The password file contains one other word of data--the word "long" or "short", depending upon which mode the user has selected.

For no reason, other than convenience, the INIRES routine was chosen to contain the central I/O error exit for Intrex. This is set up by calling a procedure named SETRTN (Section 3.1.10 2) which has as an argument the label (EREXIT) of a location to which control is to be returned whenever an error is encountered during Intrex's many reading or writing operations. It should be pointed out here that program stops or other unusual terminations (such as console turn-offs) will not return control to this location.

In one or two places in Intrex, such as in IFSRCH, special localized error returns are specified which over-ride the global one set by SETRTN. This is done to allow certain types of errors (such as "file already open") to be ignored at the point of detection. When that local routine examines the error code and finds an "unacceptable" one, it then must go to the central error location by way of a procedure call. For this reason, EREXIT is embedded in a routine within INIRES named ERRGO (Section 3.1.10.1).

B. Procedures Calling INIRES:

DYNAMO

C. Procedures Called By INIRES

SETRTN, .C.ASC, DEFBC, FSTATE, OPEN, BUFFER, FILCNT, TRFILE, WRWAIT, CLOSE, IODIAG, TYPEIT, BCDASC, OCTASC, BFCLOS, DORMNT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBFO(POT.)	Common buffer	x	
MFN1(POT.)	Monitor file name		x
DFN1(POT.)	Dump file name		x
PFN1(POT.)	Password file name		x
ESCODE(POT.)	Escape code	x	
VERBOS(POT.)	TYPEIT mode	x	
CLAMP(SST.)	Hold mode	x	

E. Arguments:

Snam: system name (in BCD)

F. Values:

None

G. Error Codes:

None

H. Messages:

(See ERRGO in Section 3.1.10.1)

NOTES:

1. The CTSS error code is printed here
2. The name of the I/O procedure in which the error occurred is printed here.
3. The address of the procedure call to the erring routine is given here.
4. The names of the file involved in the error are given here.

I. Length:

375_8 or 253_{10} words

J. Source:

RESLIS ALGOL

K. Files Referenced:

DUMnnn FILE
PASnnn FILE

3.3.2.2 NAME

Purpose

To name a list

Description

The Intrex NAME command causes the "Command Language Processor" (CLP) to call the procedure NAME. This routine should logically reside in RESLIS ALGOL, but it had to be moved out due to the limitations of the AED compiler. Therefore, NAME now is found in ANDOR ALGOL along with the procedures AND., OR., etc.

A. Operation: Code = NAME (listpt)

The one argument passed to NAME consists of a pointer to the remainder of the command line. A call to NEXITM extracts the next word from the line, which is assumed to be the name the user wishes to assign to the current reference list.

Before this occurs, however, a name for the Name File must be established. If no previous NAME command was issued, the name file component of the POT, NFN1(POT), will be empty upon entering the routine. In this case, a first name of the Name File is constructed of the form NAM---, where the last three characters are the same as those of the Monitor and Dump Files. It is then inserted into NFN1(POT) and a zero-length file with that name is created via a call to TRFILE.

The Name File is then opened for writing and buffered (using common buffer 6).

When the name is obtained from the command line, it is looked-up (via LOOKUP in NEXITM ALGOL) in the command table to make sure it is not ambiguous with an Intrex command.* If the ambiguity does exist, a message to that effect (1) is typed to the user, he is asked to use another name, and NAME returns to CLP.

If no command ambiguity is found, a procedure named CHKNAM is called. This routine (Section 3.3.2.4) converts the ASCII name to BCD codes and scans the list pointer table to see if any list already exists with that name. If CHKNAM returns a value greater than or equal to zero, it found a matching name at this location in the table. The user

*The importance of this will be made clear in the description of RESTOR is then informed that his name is ambiguous with a previous name and that he should choose another (2).

If the name passes the two ambiguity tests, it then is compared to the word ALL, since that word also must be prohibited (3) to allow DROP ALL commands. Assuming an acceptable name has been issued, the next step is to take the current list pointer from RRL(CL). Two more conditions, here, may cause the name operation to be aborted. If no pointer is found in RRL(CL), an error message (4) informs the user that he has "no current list". If a pointer exists whose type is 4, meaning it already is a named list, the user is informed that he cannot name the list twice (6).

Lists of any other type are written into the Name File by having the Boolean procedure, ANDER, do the bookkeeping and writing involved, thus saving a duplication of this kind of coding. This is accomplished through a special set of arguments to ANDER (all zero except the list pointer) which tells ANDER that this is a dummy operation intended solely for writing the entire list.

A list that has thus written into the Name File is, of course, entirely disc-resident. The unused third word of its augmented pointer is then to hold the document count of that list. This is necessary when the named list becomes non-current, since the count would then be lost in RRL(CL) and DCNT(CL).

NAME finishes by setting the LISAV(SST) bit, which indicates that at least one list has been named, changing the tag of list-pointer word two to a 4 to classify the list as having been named, inserting the BCD-coded name of the list into word one of the augmented pointer; deleting the present (current) list pointer by calling a procedure in RESLIS named DRPPTR, setting RRL(CL) to point at the named list, informing the user that his list has been named (7), and closing the Name File.

NAME then returns a zero value to CLP to indicate a completed call.

B. Procedures Calling NAME:

CLP (via CALLIT)

C. Procedures Called By NAME:

OPEN, BUFFER, TRFILE, NEXITM, TYPEIT, CHKNAM, ANDER, DRPPTR, CLOSE

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
NFN1(POT.)	Name file name	x	x
COMBF6(POT.)	Common buffer	x	
COMTB(POT.)	Command table	x	
LISAV(SST.)	List saved		x

E. Arguments:

Listpt: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

1. "Your list name is ambiguous with the Intrex command, C[1]. Please use another name." ($nam5$, $nam6$, $nam8$)
2. "Your list name is ambiguous with a previously NAMEd list. Please use another name." ($nam5$, $nam7$, $nam8$)
3. "All is a restricted word for naming lists." ($nam9$)
4. "You have no current list. Your NAME command cannot be processed." ($nam12$)
5. "You have not given a name for your list." ($nam11$)
6. "Your current list has already been named and cannot be named again." ($nam10$)
7. "C[1] is now the name of your current list." (/nameok/)

NOTES:

1. The name supplied by the user is given here.

I. Length:

402_8 or 258_{10} words

J. Source:

ANDOR ALGOL

K. Files Referenced:

NAMnnn FILE

3.3.2.3 TABENT

Purpose

To enter pointer in table

Description

This procedure resides in RESLIS ALGOL, along with the list-pointer table which it fills. When a routine, such as ANDER, IFSRCH or NAME, has constructed a new augmented pointer to go into the table of list pointers, TABENT is called with a pointer to the augmented pointer as an argument.

A. Operation: Addr = TABENT (Ptr)

TABENT first scans down the table, examining the second and third words of each table slot to find one in which both words are empty. (No list can exist without at least one of these being filled.) If no empty slot is found, the user is informed that his list table is full and he must DROP some lists before assigning any more names to lists. The extensive use of the NAME command is the only way in which the table can be filled up, since old search lists, etc., are deleted from the table as soon as they are made inactive by creating a new current list.

If an empty slot is located, the three computer words pointer to by the argument ptr are copied into the table slot. The location (core address) of this newly filled slot is then returned as a value to the calling program.

B. Procedures Calling TABENT:

NAME, IFSRCH, ANDER, NEWPT

C. Procedures Called By TABENT:

TYPEIT

D. COMMON References

None

E. Arguments:

Ptr: points to the 3-word argmented reference list pointer

F. Values:

Addr = location of filled slot in list of NAMEd lists

G. Error Codes:

Addr = 1: list table full

H. Messages:

"Your NAMEd-list file is full. You must DROP one or more list names before re-issuing your NAME command."

I. Length:

77_8 or 63_{10} words

J. Source:

RESLIS ALGOL

K. Files Referenced:

None

3.3.2.4 CHKNAM

Purpose

To check list name against pointer table

Description

A. Operation: Addr = CHKNAM (Ptr, Ln)

This procedure is called for two purposes. First, it converts the list name pointed to by the first argument to a BCD character-string of one-to-six characters. This converted name is placed in the second argument by CHKNAM.

Second, the converted name is then compared to the first word of each entry in the list-pointer table. If a match is found, the core address of the matching name is returned to the calling program.

One exceptional case is made when CHKNAM is called from the SAVE procedure with the list name, ALL. Here, the first argument is changed to contain the location of the top of the list pointer table. This is then used by SAVE in scanning down the table to SAVE all NAMEd lists (see description of SAVE in Section 3.3.3.1).

B. Procedures Calling CHKNAM:

AND, DROP, LIST, NAME, RESTOR, SAVE, USE

C. Procedures Called By CHKNAM:

BZEL, CTSIT6, FRET, RJUST

D. COMMON References

None

E. Arguments:

Ptr: ASCII pointer to name of list
LN: Name of list in BCD (returned to calling procedure).

F. Values:

Addr = location of name in list-pointer table

G. Error Codes:

Addr = -1: Name not in pointer table

H. Messages:

None

I. Length:

107_8 or 71_{10} words

J. Source:

RESLIS ALGOL

K. Files Referenced:

None

3.3.2.5 RESTOR

Purpose

To restore a NAMEd list

Description

This routine is called by CLP as the result of Intrex's only "implied" command. When the user wishes to re-activate or "restore" one of his NAMEd lists, he merely types the name of the list. The Command Language Processor (CLP) first tries to interpret the name as an Intrex command (which is why the NAME routine will not allow the user to assign names which are ambiguous with commands). If CLP fails to find it in the command table, the name is then passed to RESTOR to see if it is a NAMEd list.

A. Operation: Code = RESTOR (Aptr)

The argument Aptr passed on to CHKNAM to see if the name it points to is in the table of NAMEd lists. The value returned from CHKNAM will be less than zero if the name is not found. In this case RESTOR returns an error code to CLP, which will then declare the "name" to be an illegal command.

If CHKNAM returns a positive value, it is the address of the augmented list pointer whose first word contains the same as the one given. This pointer is then made the "active" or "current" list pointer by inserting it into RRL(CL), but only after deleting the old current list from the list table. If that list was also a "restored" list (RLIC(SST) on), then the deletion is made via DELIST. Otherwise, CLEANP is called to clean up the entire search structure.

The document count of the newly restored list is transferred from the third word of the list pointer (where it was saved by NAME) to the decrement of RRL(CL) and DCNT(CL).

The restored-list-is-current indicator (RLIC) and the resultant-reference-list-exists indicator (RRLE) are set and RESTOR returns a zero value to CLP.

B. Procedures Calling RESTOR:

CLP (via CALLIT)

C. Procedures Called By RESTOR:

CLEANP (via CALLIT), CHKNAM, DELIST

D. COMMON References

Name	Meaning	Interrogated?	Changed?
RLIC(SST.)	Restored list in core		x
SNX(SST.)	Search not executed	x	
RRLE(SST.)	Reference list exists		x

E. Arguments:

Aptr: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

Code = -1: Name not found in table

H. Messages:

None

I. Length:

75_8 or 61_{10} words

J. Source

RESLIS ALGOL

K. Files Referenced:

None

3.3.2.6 LIST

Purpose

To display the list names

Description

A. Operation: Code = LIST (Arg)

This procedure is called by CLP in response to a user's LIST command. The LIST command may be used for any of three purposes. The simplest, executed by typing the word LIST with no arguments following it, will cause the procedure to produce a list of the user's NAMEd reference lists. A call to NEXITM will return a zero value if no argument exists. This tells LIST to scan the list-pointer table looking for augmented pointers to NAMEd lists (type 4). Each one that is found is converted to ASCII by BCDASC and printed via TYPEIT. This mode of LIST is also called by the USE procedure described with SAVE in Section 3.3.3.3.

A second use for LIST is implemented when NEXITM returns a pointer to the word FILE. In this case, the user is asking for a listing of the Save File names which are kept in a directory on the disc. This list of file names is produced by calling a procedure named LISFIL, which resides in SAVLIS ALGOL and is described in Section 3.3.3.7.

A third use of LIST is activated by following the LIST command with the name of a save file, whose list names the user would like to see. If the word after the LIST command is not FILE, it is compared to the current Save File name in the POT (if there is one). Failing a match there, the Save File directory is scanned to see if the argument is the name of a non-current Save File. This is done by calling the procedure CHKSAV (Section 3.3.3.2). If CHKSAV says the name is not in the directory, an error message is printed (2) and LIST returns to CLP. If the name is found, LIST calls LISTSL (Section 3.3.3.6) which will read and print the list names stored in that Save File.

B. Procedures Calling LIST:

CLP(via CALLIT), USE

C. Procedures Called By LIST:

Via CALLIT: CHKSAV, LISFIL, LISTSL
Directly: CHKNAM, NEXITM, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
SFN1(POT.)	Save file name	x	

E. Arguments:

Arg: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

1. "N NAMEd Lists currently being held" (/nam0/, /nam4/)
2. "File[1] is not a SAVE file name" (/liser1/)

NOTES:

1. The word supplied by the user is given here.

I. Length:

217_8 or 145_{10} words

J. Source:

RESLIS /ALGOL

K. Files Referenced:

None

3.3.2.7 DROP

Purpose

To drop a named list or a Save File

Description

A. Operation: Code = DROP(Ptr)

This procedure is called by CLP in response to a DROP command. It accepts a pointer to the remainder of the command line and, from it, extracts names of lists which are to be deleted from the list-pointer table. NEXITM is called repeatedly to access the next name until a delimiter other than space (slash) is found to end the string of names.

If no item is found to follow the DROP command, an error message (1) is displayed to the user and DROP returns to CLP.

If NEXITM returns with a pointer to an item, that pointer is passed as an argument to CHKNAM (Section 3.3.2.4) to see if the name is in the table. A negative value from CHKNAM indicates that the name was not found. The user is told that this name is not that of a NAMEd list (1) and the next item is taken (if any).

A positive value from CHKNAM would be the table location of the list pointer to be dropped. Zeroes are deposited into the three words of this table entry, an indicator is set showing that a deletion was made, and the next item is sought.

When no more names are found in the command, the Name File is condensed (if any deletions were made) by calling a procedure named CONNAM (described below). This procedure re-writes the Name File, omitting the deleted lists.

Two exceptional cases are recognized by DROP. If the list name is ALL, then the entire list-pointer table is zeroed out (thus killing the current list also). The "named-list flag" of the System-State-Table, LISAV(SST), is set to false, the Name File is deleted from the disc, and its name is removed from the POT.

If the item found after the DROP command is FILE, the meaning of the DROP command is completely changed. It now becomes a request to delete a Save File from the disc. Another call to NEXITM is made to get the name of the file to be dropped. This time a call to CHKNAM is made merely to convert the file name to BCD. The converted name is then passed to another procedure, CHKSAV, which determines if the name of the file exists in the Save File directory (see Section 3.3.3.2). A zero returned from CHKSAV indicates no such file exists. The user is then informed of his error (4) and DROP returns to CLP.

If CHKSAV indicates the file is there, a call to DELFIL (a CTSS procedure) deletes it from the disc. Its directory entry is removed by calling CONDIR (see Section 3.3.3.5) using the directory address returned by CHKSAV as an argument to CONDIR.

Here, also additional names may be processed by re-calling NEXITM until a command terminator (slash) is found.

B. Procedures Calling DROP:

CLP (via CALLIT)

C. Procedures Called By DROP:

NEXITM, CHKNAM, DELFIL, CALLIT(CHKSAV, CONDIR), CONNAM, TYPEIT, BCDASC

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
NFN1(POT.)	Name file name	x	x
LISAV(SST.)	List saved		x

E. Arguments:

Ptr: ASCII pointer to user command line

F. Values:

Code = 0

G. Error Codes:

None

H. Messages:

1. "You have not provided the name of a NAMEd list" ($nam3$)
2. "List[1] has not been NAMEd" ($nam2$)
3. "NO NAMEd lists currently being held" ($nam0$, $nam4$)
4. "File[1] is not a SAVE file name" ($user1$)
5. "You have not provided the name of a SAVE file" ($user1$)

NOTES:

1 The name provided by the user is given here.

I. Length:

235_8 or 157_{10} words

J. Source:

RESLIS ALGOL

K. Files Referenced:

None

3.3.2.8 CONNAM

Purpose

To condense the Name File

Description

A. Operation:

When the DROP command succeeds in deleting a NAMEd-list pointer from the table, the list itself is removed from the Name File to prevent that file from growing any larger than necessary. This is accomplished by scanning the list-pointer table and noting all lists which are NAMEd (type 4). The table position of each of these lists is saved in an array for use in reconstructing the Name File.

A temporary file named TEM---, where the last three characters of the name correspond to those of the Dump, Monitor, and Name Files, is created to hold the lists which are to be retained. Then the array of table positions supplies pointers to the lists which must be read from the old Name File and written into the new one. As each list is copied, its depth in the new file is inserted into the second word of the augmented pointer.

When all the NAMEd lists have been copied, the new file is renamed (via CHFILE) to the original Name File name, destroying the original file.

If the original Name File is larger than 100 records, the user is warned of a possible delay during re-writing.

B. Procedures Calling CONNAM:

DROP

C. Procedures Called By CONNAM:

FILCNT, TYPEIT, OPEN, BUFFER, CLOSE, RDWAIT, WRWAIT, TRFILE, DELFIL, CHFILE, FRER, FREE,FRET

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
TOTNAM(POT.)	Dump File pointer	x	
NFN1(POT.)	Name File name	x	
COMBF6(POT.)	Common buffer	x	
COMBF6(POT.)	Common buffer	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

"There will be a slight delay while Intrex con[illegible]nses your NAMEd List File. Please stand by." ($co[illegible])

I. Length:

435_8 or 285_{10} words

J. Source:

RESLIS ALGOL

K. Files Referenced:

TEMnnn FILE
NAMnnn FILE

3.3.3 Saving and Using

3.3.3.1 SAVE

Purpose

To save a NAMEd list in a file.

Description

When used alone, the command SAVE will copy the references in a previously NAMEd list into a permanent disk file for use during future Intrex sessions. When used before the word FILE, a file for the above purpose will be opened for writing. The first name of the file or the name of the list to be saved are given as arguments to the command.

A. Operation: Code = SAVE(LIST) or SAVE FILE(SFILE)

The procedure SAVE is called by the command language processor (CLP) in response to the user command SAVE or SAVE FILE. The latter is used to create a Save File name which the user may then fill with NAMED lists that he would like to SAVE for future sessions of Intrex.

The procedure accepts as an argument the pointer to the rest of the command line, and calls NEXITM to extract the next word. If a next word does not exist, an error message (1) is issued and SAVE returns to CLP.

If a next word exists, it is passed to CHKNAM (described in Section 3.3.2.4) which converts the word to BCD and looks it up in the list-pointer table. Names not in the table cause CHKNAM to return a negative value. This prompts SAVE to determine if the "name" is the word FILE or the word ALL. If it is neither, the same error message (1) as above is given and SAVE returns.

The word FILE after a SAVE command produces another call to NEXITM to get the name the user wants to assign to his Save File. Here again, if NEXITM can find no next word, an error message (4) is typed. Otherwise, CHKNAM is called to convert the name to BCD. If the name is FILE, the user is informed (message 2) that this word cannot be used as a Save File name (to allow both USE and USE FILE commands for reactivating Save Files —see USE description in Section 3.3.3.3). If the name already exists in the directory, the user is informed (3) tnat his chosen name is in use and he must select another.

If CHKSAV returns a zero value, the name is new and acceptable and will have been added to the directory by CHKSAV. It is then inserted into SFN1(POT.), which will be used as the first name of the Save File in any forthcoming transactions. A file with this first name and the last name SAVE is then opened for writing. The first block or record of this file will be used to hold a table of augmented list pointers, similar to those in the in-core table of pointers. The first step in constructing this table is to write a block of 432 blank (zero) words. This is accomplished by zeroing out common buffer 6 and writing the buffer into the newly opened Save File. The Save File is then closed until the user deposits something into it via a SAVE command.

TOTSAV(POT), which is a component of the POT used to hold the number of disk records in the Save File (address portion) and to hold the Save File's list-table index (decrement), is set to the initial value of 1 in both cases. SAVE has then finished executing a SAVE FILE command and returns to CLP.

If the word FILE does not follow a SAVE command, then the procedure takes a totally different path.

The only other exceptional case occurs when the word ALL is given by the user who wishes all NAMED lists to be placed in his Save File. When the SAVE procedure finds the word ALL following the SAVE command, a flag (TOT) is set which will cause the list-copying logic to be repeated for all NAMEd lists. A list name which is found in the list-pointer table by CHKNAM will return a table position at which the list pointer is found. But before proceding with the SAVE operation, the SFN1(POT) register is examined to make sure that the user has previously issued a SAVE FILE command. If this slot is empty, an error message (4) is given and SAVE returns to CLP.

If a Save File name exists, the list table index for the Save File is extracted from the decrement of TOTSAV(POT) and checked to see if more lists can be SAVED without overflowing the in-core pointer table, which has a limit of 40 augmented pointers all told (including current pointers resulting from a search etc.). If the limit has been reached (a very unlikely event), then the user is told to assign a new Save File name before saving any more lists.

Next, the Save File is opened for writing and the Name File is opened for reading. Common buffer 0 (at the top of core) is used as an I/O buffer in transferring the data.

The read-position within the Name File and the length of the list to be saved are obtained from the augmented pointer whose location was supplied by CHKNAM. The write position within the Save File is computed using the block count stored in the address of TOTSAV(POT.). This position is also temporarily stored in the augmented pointer's disk-address so that the pointer relates to the list as it will be in the Save File. This pointer is then written into the first record of the Save File at the index position extracted from TOTSAV(POT.). The index is then incremented by three for the next pointer deposit, and the disk-address of the pointer is re-set to its original, Name File address.

Finally, all the data having been established, the transfer of the list from the Name File to the Save File is made using RDWAIT and WRWAIT. If the list is longer than a record of 432 words, repeated reads and writes are made with the two disk addresses being incremented each time and the number of records copied accumulated in TOTSAV(POT.).

When the entire list has been copied, the ALL flag, TOT, is tested to see if SAVE is done or must go back for another possible NAME list. In the case where ALL was used, no table position of the list pointer would have been returned by CHKNAM, but the ASCII pointer argument will be changed by CHKNAM to contain the address of the top of the table of list pointers. This address will be used by SAVE to step through the table copying all NAME lists.

When no more copying remains to be done, the latest table index of the Save File is inserted in the decrement of TOTSAV(POT.) for use in the next SAVE operation. At this point, the delimiter found by NEXITM in extracting the list name is examined. If it is not a command terminator (slash), the procedure loops back to call NEXITM again to get the next list name which the user is saving.

When the command terminator is found, the Save File and Name File are closed and a zero value is returned to CLP.

B. Procedures Calling SAVE:

CLP (via CALLIT)

C. Procedures Called By SAVE:

NEXITM, CHKNAM, OPEN, BUFFER, RDWAIT, WRWAIT, CLOSE, CHKSAV, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
CD(POT.)	command delimiter table address	x	
CFT(POT.)	front trim table address	x	
CET(POT.)	end trim table address	x	
SFN1(POT.)	Save File name one	x	x
NFN1(POT.)	Name File name one	x	
TOTSAV(POT.)	Save File address and table Index	x	x
COMBF6(POT.)	Save File write buffer	x	
COMBF0(POT.)	Name/Save I/O transfer buffer	x	

E. Arguments:

LIST: ASCII pointer to list name

<u>or</u>

SFILE:ASCII pointer to File Name

F. Values:

Code = 0 (Zero)

G. Error Codes:

None

H. Messages:

1. "You have not provided the name of a NAMED list." ($nam3$)
2. "The word FILE may not be used as the name of a SAVED File. Please use another name." ($sav8$, $nam8$)
3. "The name you have assigned to your SAVE file is already in use. Please repeat your SAVE FILE request using another name." ($sav2$)
4. "You have not provided a name for your SAVE file." ($sav3$)
5. "Your current SAVE file is full. You must assign a new name via the SAVE FILE command before saving any more lists." ($sav4$)

I. Length:

521_8 or 337_{10} words

J. Source:

SAVLIS ALGOL

K. File References:

NAM --- FILE
user SAVE

3.3.3.2 CHKSAV

Purpose

To check the Save File directory

Description

This procedure is used either to find a file name in the Save File directory, or to add a new name tc it. The SAVE procedure described in the previous section calls CHKSAV to add new file names. All other procedures using CHKSAV do so only to see if a file already exists in the directory.

A. Operation: R = CHKSAV(FN)

Whether CHKSAV is being called by SAVE for adding a new file name or from some other procedure can be determined by comparing the name of the file passed in the argument FN to the name stored in the parameter used by SAVE for holding the file name. This parameter is accessible to CHKSAV since it is compiled with SAVE in the same source file. Calls to CHKSAV from outside this source file will mean that these file names will not match. In both cases, CHKSAV looks for the file name in the directory of Save File names.

If a Saved File directory exists, it is opened for reading. Each word is then read and compared to the name passed to CHKSAV in FN. If a match is found, the directory file position of this name is returned to the calling program.

If no similar name is found in the directory, the directory is closed. The determination is then made as to whether the call to CHKSAV came from SAVE or elsewhere. If it came from elsewhere (such as USE in RESLIS), then CHKSAV is done and returns.

When the call is from SAVE, the name is to be added to the directory. The directory file is opened for writing, the single word containing the name is entered via WRWAIT, and the directory file is closed.

Before returning to the calling program, the return value is set to zerc (if a name was added to the directory) or to a file address (if a name was found already in it).

B. Procedures Calling CHKSAV:

SAVE, USE, DROP, LIST, LISTSL

C. Procedures Called By CHKSAV:

FILCNT, OPEN, BUFFER, RDWAIT, WRWAIT, CLOSE

D. COMMON References:

Name		Interrogated?	Changed?
COMBF6(POT.)	directory I/O buffer	x	

E. Arguments:

FN: BCD coded file name

F. Values:

R = 0 (zero)
R = file address

G. Error Codes:

None

H. Messages:

None

I. Length:

137_8 or 95_{10} words

J. Source:

SAVLIS ALGOL

K. File References:

SAVED DIRECT

3.3.3.3 USE

Purpose

To use a Save File

Description

USE provides the user with the facility for reactivating a Save File previously created. It is called by CLP in response to the command, USE.

A. Operation: Code = USE(FN)

USE accepts an argument containing an ASCII pointer to the rest of the command line.

A call to NEXITM takes the next word from the command line. If none is found, an error message (1) is printed and USE returns. CHKNAM is called to convert the word to BCD. If this word is ADD, the name of a Save File should be found next on the command line and the mode of re-activation will be somewhat different. In the "add" mode the Save File specified by the command will become the user's "current Save File", so that NAMEd lists may be SAVEd in it, but the list pointers already in the Save File will not be read into core and made active NAMEd lists. When the word ADD is seen, therefore, USE sets an indicator and goes back to recall NEXITM to get the name of the Save File to be activated. If the word is FILE, it is ignored (USE and USE FILE are equivalent) and NEXITM is called again.

In either mode, when a file name has been found and converted to BCD, it is passed to CHKSAV to see if it is actually a Save File recorded in the directory. If CHKSAV returns a zero value, it failed to find the file name and an error message (1) results. If the file name is found, a call to FSTATE is made to see if the file really exists on the disk, and, if so, to obtain its length. If the file is not on disk, another error message (2) is printed and the name of the file is deleted from the directory by calling CONDIR (Section 3.3.3.5.

At this point, if the "add" mode indicator is on, the procedure skips ahead to the final few operations of activating the file. If the regular mode is being executed, NFN1(POT.) is examined. If it does not contain a Name File name, one is constructed using the same numeric last three characters contained in the user's Dump File. A call to DELFIL is made to delete any Name File by this name which might already exist from a previous Intrex session.

Now, the entire Save File is copied into the new Name File by a small routine called, MOVEIT (Section 3.3.3.4). The length of the new Name File (in records) is inserted into TOTNAM(POT.). This length is also placed in TOTSAV(POT.) and the name of the Save File is inserted into SFN1(POT.) to make the Save File current and receptive to new NAMEd lists.

To complete the activation of the lists in the Save File and new Name File, the list table in the first record of the Name File is read into the in-core list-pointer table. This of course, destroys any list pointers already there, including both NAMEd lists and the current list from a search, if any. (This is the reason for the ADD option.) With no current list, CRL.(CT) is made 0. Once the pointers are read into core, the Name File is closed and a verification message (3) informs the user what the command has done. This message ends with a list of the NAMEd lists just activated, produced by calling the LIST procedure with an argument of 0 (zero) (Section 3.3.2.6).

In the "add" mode, TOTSAV(POT.) and SFN1(POT.) are filled as in the regular mode, but a somewhat different message(4) is printed to verify the execution of the user's command. The list of NAMEd lists which completes the message is produced by calling LIST with an argument consisting of a pointer to the name of the Save File. This prompts LIST to display the names of the lists in that re-activated Save File.

Finally, in both modes the index of the Save File's list-pointer table is set in the decrement of TOTSAV(POT.), using the count of lists just produced by the call to the LIST routine. In all cases USE returns a value of zero to CLP.

B. Procedures Calling USE:

CLP (via CALLIT)

C. Procedures Called By USE:

NEXITM, CHKNAM, CALLIT(CHKSAV, MOVEIT, CONDIR), FSTATE, DELFIL, OPEN, BUFFER, RDWAIT, CLOSE, TYPEIT, LIST

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
NFN1(POT.)	Name File name one	x	x
DFN1(POT.)	Dump File name one	x	
SFN1(POT.)	Save File name one		x
COMBF6(POT.)	Save File directory read buffer	x	

E. Arguments:

FN: ASCII pointer to file name

F. Values:

Code = 0 (zero)

G. Error Codes:

None

H. Messages:

1. "You have not provided the name of a SAVE file." ($USER1$)
2. "SFILE[1] is not found to be stored on disk and is being deleted from the SAVE file directory." ($SAV7$)
3. "SFILE[1] has become your current NAMEd list file and SAVE file. The list names in this file may now be restored to active status by typing their names, which are:" ($USEM1$)

   (List of List-names)
4. SFILE[1] has become your current SAVE file and will accept SAVEd lists. Your current NAMEd lists, if any, are retained and may now be SAVEd, if and when desired." ($USEM2$)

   (List of List-names).

   NOTES:

   1. The file name supplied by the user is given here

I. Length:

724_8 or 468_{10}

J. Source:

RESLIS ALGOL

K. File References:

NAM---FILE

3.3.3.4 MOVEIT

Purpose

To copy a disk file

Description

A. Operation: MOVEIT(IN1, IN2, OUT1, OUT2)

This procedure accepts as arguments the names of two disk files. The first two names are those of a file to be copied, such as a Save File to be reactivated as a Name File. The second two names are those to be given to the new file. The files are opened for reading and writing, respectively, with Common Buffer 6 being used for buffering the writing of the new file and Common Buffer 2 used for transferring the data in and out of core.

Both files are closed upon completion of the transfer.

B. Procedures Calling MOVEIT:

USE

C. Procedures Called By MOVEIT:

OPEN, BUFFER, RDWAIT, WRWAIT, CLOSE

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBF2(POT.)	Core-transfer buffer	x	
COMBF6(POT.)	Write buffer	x	

E. Arguments:

IN1, IN2:	Input File names
OUT1, OUT2:	Output File names

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

Approx. 100_8 or 64_{10} words

J. Source:

SAVLIS ALGOL

K. File References:

NAM---FILE
user SAVE

3.3.3.5 CONDIR

Purpose

To condense the Save File Directory

Description

A. Operation: CONDIR(FLOC)

This procedure is used to drop file names from the Save File directory whose position is given in the argument of the call. The procedure would normally be called from the DROP routine in response to a DROP FILE command. It might also be called from the previously described USE procedure, if the file to be USEd was not found on the disk.

CONDIR first calls FILCNT to see if a Saved File directory exists. If not, the procedure simply returns immediately to the calling program. Otherwise, the length of the file is established and used to control a loop which reads one word of the file at a time. The directory is opened for both reading and writing. Every word except the one found in the argument position is written back into the file. When all reading and writing is finished, the new file is truncated to one word less than its original length by a call to the CTSS routine, TRFILE (Section 3.5.1.8).

Finally, the directory file is closed and CONDIR returns to the calling routine.

B. Procedures Calling CONDIR:

DROP, USE

C. Procedures Called By CONDIR:

FILCNT, OPEN, BUFFER, RDWAIT, WRWAIT, TRFILE, CLOSE

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBF6(POT.)	I/O Buffer	x	

E. Arguments:

FLOC:	integer (file index)

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
113_8 or 75_{10} words

J. Source:
SAVLIS ALGOL

K. File References:
SAVED DIRECT

3.3.3.6 LISTSL

Purpose

To list names of lists in a Save File

Description

A. Operation: CNT = LISTSL(SLN)

This procedure is called by LIST when the user follows the LIST command with the name of a Save File whose SAVEd list names he wants displayed.

LISTSL accepts an argument containing a BCD-Coded Save File name. A call to FILCNT on his file name is made to determine if the file actually exists on the disk. If a negative return from FILCNT shows that no such file exists, an error message is issued (1) and the file name is deleted from the directory (if it exists there) by a call to CONDIR (described earlier).

If the file is found on the disk, TYPEIT is used to produce an introductory message (2). Then the file is opened for reading, buffered, and the names in the table in the first record are read, one word at a time. Each list name is printed by TYPEIT after conversion to ASCII by BCDASC. An empty word signals the end of the list-pointer table and the loop is terminated. The names are counted as they are read and printed. This count is tested for the possibility of zero lists in the table, in which case the user is given the word "none" for a result.

In all cases the count is returned as a value to LIST which uses it to add a summary message (see LIST in Section 3.3.2.6.

B. Procedures Calling LISTSL:

LIST

C. Procedures Called By LISTSL:

FILCNT, TYPEIT, BCDASC, RDWAIT, CLOSE, LOCMES, CONDIR, CHKSAV

D. COMMON References:

Name	Meaning	Interrogated	Changed?
COMBF6(POT.)	I/O buffer		

E. Arguments:

SLN: BCD file name

F. Values:

CNT=count of List names in Save File

G. Error Codes:

None

H. Messages:

1. "SFILE[1] is not found to be stored [illegible] disk and is being deleted from the SAVE File direc[illegible]." ($SAV7$)

2. "Lists in file SFILE[1]" ($SAV6$)

(List of List Names)

NOTES:

1. The name of the file specified by the user is given here.

I. Length:

141_8 or 97_{10} words

J. Source:

SAVLIS ALGOL

K. File References:

user [illegible]AVE

3.3.3.7 LISFIL

Purpose

To list Save File names

A. Operation: LISFIL( )

This procedure is called by the LIST procedure (see Section 3.3.2.6) when a LIST FILE command is given by a user desiring to see what Save Files are currently in the Save File directory. LISFIL first checks to see if a directory exists by calling the procedure FILCNT (Section 3.4.3.1). If FILCNT returns a negative value, no directory exists and the user is informed that no Save Files are currently on the disk(1).

A positive value returned from FILCNT will be the length of the directory, which is then used to control a loop in which the directory entries are read and listed one-by-one. RDWAIT is used to read each directory word, while BCDASC converts the file name from BCD to ASCII so that TYPEIT can display it to the user.

When all the file names in the directory have thus been listed, another TYPEIT call is made to give the user a summary message (1) as to the number of Save Files currently held.

B. Procedures Calling LISFIL:

LIST

C. Procedures Called By LISFIL:

FILCNT, OPEN, RDWAIT, TYPEIT, BCDASC, CLOSE

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
COMBF6(POT.)	I/O buffer	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "N^{I} SAVE Files currently being held on File." ($SAV5$)

<u>NOTES:</u>

1. The number of file names listed by the LISFIL procedure is given here.

I. Length:

102_8 or 66_{10} words

J. Source:

SAVLIS ALGOL

K. File References:

SAVED DIRECT

3.3.4 Boolean Operations

3.3.4.1 AND.

Purpose

To control Boolean commands

Description

The process of performing a Boolean intersection on two lists of references produces a resulting list containing only those references which share certain common specifications or attributes (namely, document number and/or subject term number). When such matching criteria are found, one or both of the similar references is taken into the new list and the comparison continues. This process generally means that the resulting list will be smaller than the length of the shorter of the two original lists.

If the user wishes to intersect two lists on their document numbers only, he issues the command, "LIST1, AND LIST2" where LIST1 and LIST2 are names he previously assigned via the NAME command. (LIST1 may be omitted in which case the current list is taken as LIST1). The procedure AND., called by CLP in processing the AND command, will read the second list name, find it in the list pointer table, and use its pointer and the current list pointer as arguments to the main Boolean procedure ANDER (described in Section 3.3.4.5).

Since the other Boolean commands and their corresponding procedures all necessitate basically the same preparations, subroutine calls, message construction, etc., duplicate coding was obviated by making AND. a general purpose set up routine used also by WITH., OR., and NOT.

A. Operation Code = AND.(SPA)

The POT location MODEG(POT.) contains an "anding mode" indicator (normally set by CLP) which is used by the procedure ANDER to determine what course of action it should follow while comparing references. This mode is extracted from the POT upon entering AND. to be used as an argument in the ensuing call to ANDER. It will have been set to 3 (document number intersection only) by CLP in anticipation of an AND command.

Next, the delimiters and trim table pointers are extracted from the POT and set up for disecting the command line with calls to NEXITM. Flags used for indicating the use of AND. by the NOT command or the OR command are set to false, and the argument (the pointer to the remainder of the command line) is saved in a local variable.

The preceding operations are pertinent only to the AND command. The steps described below are used also by WITH, OR, and NOT commands.

The CTSS blip feature is used during all Boolean operations as well as during searching. SETBLP is called specifying the blip character to be that contained in BLIP(POT.) and setting a time period of one second between blips.

NEXITM is used to obtain the next word on the command line, assumed to be a NAMEd list. If NEXITM returns an error code, a message (1) is printed telling the user he has not given a list name.

The ASCII word returned by NEXITM is now converted to BCD by the basic conversion procedure CTSIT6 (see Section 3.4.2.8) and tested to see if it is the word NOT. If it <u>is</u> NOT, then the user has issued an AND NOT command, which is quite different, of course, from an AND command. In this case, NOT. is called by AND. (see description of NOT. in Section 3.3.4.4), which forces another cycle through AND. with different conditions. In this case two exits will be taken, first from NOT. and then from AND. to return to CLP. This bears some resemblance to the exit from a recursive procedure.

Any word other than NOT is passed to the procedure CHKNAM (see Section 3.3.2.5) to see if the word is the name of a NAMEd list. If CHKNAM returns a negative value, the name was not found in the list table and an error message to this effect (2) is displayed to the user.

Next, RRL.(CL.) is examined to make sure there is a current list on which to perform the required Boolean operation. If it is empty, a message is typed (3) stating that the user has no current list.

Now the procedure ANDER is called with the pointer to the NAMEd list (returned by CHKNAM), the current list pointer, the anding mode, and a zero (indicating "no attribute screening") passed as arguments. The scanning of the two reference lists, as specified by the anding mode, takes place in ANDER. The resulting list is written into the Dump File and a pointer to its augmented (three-part) pointer is returned to AND..

If the value returned is not a pointer but an error code, AND. immediately aborts and returns to CLP. No error message is given by AND. here because it will have been given by the routine causing the error, usually TABENT (Section 3.3.2.3) complaining that the list table is full.

If the value is a zero, the Boolean comparison produced no satisfying references so the following pointer preparations are skipped.

The pointer returned by a successful call to ANDER is made the current list pointer by inserting it into RRL.(CL.). First, however, the old list pointer in this component of CL must be deleted by calling DRPPTR (Section 3.2.4.8), unless that pointer relates to a NAMEd list, which must be retained.

Having completed its task, AND. must then tell the user about the result. A complex call to TYPEIT is prepared, one of whose arguments is determined by the type of Boolean command being performed as indicated by the anding mode. The result message (4) will specify that ANDing, WITHing, ORing, or NOTing has taken place according to whether the mode is 3, 0, 4, or less than 0, respectively. The message will also state the number of documents involved in the resulting list (from the decrement of RRL.). If that number is zero, the user is told (5) that his most recent list is being retained (the contents of RRL. have not changed).

Before assuming that its job is completed, AND. examines the delimiter found by NEXITM to determine if the command terminator has been reached. If not, NEXITM is called again and its returned word converted to BCD. This word is compared to the words AND, OR, NOT, and WITH in case the user is stringing Boolean commands together. If the word is any of these, the proper corresponding mode is set and AND. loops back to the first call to NEXITM to read the name of the next list on which to operate.

If the word is none of these, AND. goes back to the place where CHKNAM is called to see if the user is chaining lists together for the same Boolean operation (e.g. OR L1 L2 L3).

Once a command terminator is found the blip feature is turned off by calling SETBLP with zero arguments, and AND. prepares to exit.

Here, the OR command flag (set by OR.) and the NOT command flag (set by NOT.) are tested. If either flag is on, transfer is made to the exit location of the corresponding routine and, hence, back to CLP.

If both flags are off, the argument pointer to the command line is updated and AND. returns a zero value to the calling routine.

B. Procedures Calling AND.:

CLP, WITH. (both via CALLIT)

C. Procedures Called By AND.:

SETBLP, NEXITM, CTSIT6, CHKNAM, ANDER, DRPPTR, TYPEIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
MODEG(POT.)	anding mode	x	
CD(POT.)	command delimiters	x	
CET(POT.)	front-trim table	x	
CET(POT.)	end-trim table	x	
BLIP(POT.)	blip character	x	
RRL.(CL.)	resultant ref. list ptr.	x	x

E. Arguments:

SPA: ASCII pointer

F. Values:

Code = 0 (zero)

G. Error Codes:

None

H. Messages:

1. "You have not provided the name of a NAMEd list" ($nam3$)
2. "xlist [1] has not been NAMEd." ($nam3$)
3. "You have no current list upon which to perform Boolean operations." ($anerr1$)

4. "The list resulting from ANDing [2] the current active list with lname [3] contains N [4] documents." ($anor1$, $anor2$, $anor4$, $anor4a$, /op3a/ or /op3b/)

5. "Your last active list has been retained." ($fso6$)

NOTES:

1. The user-supplied word is given here.
2. This may be WITHing ($anor0$), ORing ($anor3$) or NOTing ($anor5$) if the anding mode so dictates.
3. The user-supplied list name is given here.
4. The number of documents in the resulting list is given here.

I. Length:

351_8 or 233_{10}

J. Source:

ANDOR ALGOL

K. File References:

None

3.3.4.2 WITH.

Purpose

To control the WITH (strong AND) command

Description

If the user wants to intersect subject term numbers as well as document numbers, then he types, "LIST1 WITH LIST2". (Here again LIST1 may be taken as the current list.) If LIST1 contains the reference list for the word ZINC and LIST2 contains the reference list for the word OXIDE, then the result of the above command would be the same as if the user had done a subject search on the phrase ZINC OXIDE.

The same comparison procedure (ANDER, Section 3.3.4.5) is used by all the Boolean operations. A mode argument to ANDER controls the type of action taken during and after comparison of the lists. The subject-search logic and the WITH command both call ANDER with a mode of 0, which causes term numbers to be used in the intersection process. Combination subject/title searches, subject/author searches, subject/author searches and the AND command call ANDER with modes of 1, 2 and 3 respectively, which prevent ANDER from considering term numbers. The resulting list from this kind of intersection is usually larger than that produced by the more restrictive WITH command intersection.

A. Operation: Code = WITH.(SPW)

WITH. is called by the Command Language Processor (CLP) with an argument containing a pointer to the rest of the command line. The procedure itself performs only two steps, since most of the processing is actually done by AND. and ANDER.

First, WITH. inserts a mode of zero into the MODEG component of POT., which will cause ANDER to consider term numbers in the more demanding type of intersection.

Then WITH. calls AND., passing along the same commmand line pointer as its argument. AND. will, of course, return to WITH. in the normal procedural manner, and hence back to CLP. with the same zero value which is standard for command routines.

B. Procedures Calling WITH.:
CLP (via CALLIT)

C. Procedures Called By WITH.:
AND.

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
MODEG(POT.)*	anding mode		x

E. Arguments:
SPW: ASCII pointer

F. Values:
Code = 0 (zero)

G. Error Codes:
None

H. Messages:
None*

I. Length:
17_8 or 15_{10}

J. Source:
ANDOR ALGOL

K. File References:
None

*Directly from WITH. for indirect results, see AND..

3.3.4.3 OR.

Purpose

To control the OR command

Description

A Boolean OR on two lists of references (L1 OR L2) produces a resulting list containing references found in either or both of the two lists. It merges the two lists into one, maintaining the descending order of document numbers and ascending order of term numbers within each document. Redundant references are discarded so that it is possible that the length of the resulting list will be less than the sum of the lengths of the two original lists. The OR procedure uses AND. logic and causes a call to ANDER with a mode argument of 4 which sets the ANDER switching mechanism to merge rather than exclude. The OR command would generally be used to combine the lists of search words or phrases having similar meanings so that all the references pertaining to a given area will be included in a group which would be unattainable by ordinary search techniques.

A. Operation: Code = OR.(SPO)

OR. is called by CLP and accepts an argument containing a pointer to the rest of the command line. This pointer is placed in a local variable which is common to the AND. procedure. Then a similarly common variable used for holding the anding mode is set to 4, the number which indicates ORing to both AND. and ANDER. Since this mode may be changed later if AND. finds subsequent Boolean commands on the same line, a flag is set to serve as a reminder that the final exit back to CLP must come from OR.. This flag will be examined at the end of AND. and, if set, will cause AND. to branch back to the exit point of OR.

After setting the "OR" flag, transfer is made from OR. to AND. at the point in AND. where the blip feature is activated and the command line is interrogated by NEXITM (see earlier description of AND.) AND. then (using ANDER) performs the ORing operation. When it finishes and returns to OR., the advanced command line pointer is deposited into the argument address and a zero value is returned to CLP.

B. Procedures Calling OR.:
CLP (via CALLIT)

C. Procedures Called By OR.:
None*

D. COMMON References:
None*

E. Arguments:
SPO: ASCII pointer

F. Values:
Code = 0 (zero)

G. Error Codes:
None

H. Messages:
None*

I. Length:
31_8 or 25_{10}

J. Source:
ANDOR ALGOL

K. File References:
None

*Directly from OR. For indirect results, see AND .

3.3.4.4 NOT.

Purpose

To control the NOT command

Description

The Intrex command, LIST1 NOT LIST2, means "produce a list which contains only those references from LIST1 which are not also in LIST2". The resulting list length must be less than or equal to LIST1.

Like Intersection, Negation may be performed either with or without regard to term numbers. If only the word NOT is used between list names, then term numbers will be considered and only those references having the same term and document numbers as the LIST2 reference will be discarded (ANDER mode = -1). If the two words AND NOT appear between list names, then any reference of LIST1 having the same document number as a LIST2 reference will be dropped (ANDER mode = -3). Thus, the result of

LIST1 NOT LIST2

will, in general, contain more references than

LIST1 AND NOT LIST2

since there will usually be fewer discarded references.

A. Operation: Code = NOT.(SPN)

NOT. is called by CLP or AND. with an argument containing a pointer to the rest of the command line. This pointer is saved in a local variable for use by AND. as described above under OR..

Since there are two kinds of NOT operations and, therefore, two modes which might be passed to AND. and from there to ANDER, NOT. must examine the mode indicator in MODEG(POT.) to determine which NOT mode is to be used. A zero in this component means that the word AND has already been encountered on the command line and the local mode is then set to -1.

A three (3) in the mode slot means that NOT. was called directly from CLP (no AND), so the mode in the local parameter is set to -3. A flag is set to indicate that return must be made via NOT. and the control branches to AND. as described above for OR..

AND. and ANDER then perform the actual processing of the command. At the end of AND., the NOT flag sends control back to NOT. which replaces the old argument command pointer with the now advanced local version and returns a zero value to the calling program.

B. Procedures Calling NOT.

AND, CLP (via CALLIT)

C. Procedures Called By NOT.

None *

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
MODEG(POT)*	anding mode	x	

E. Arguments:

APN: ASCII pointer

F. Values:

Code = 0 (zero)

G. Error Codes:

None

H. Messages:

None*

I. Length:

37_8 or 31_{10}

J. Source:

ANDOR ALGOL

K. File References:

None

*Directly from NOT.. For indirect results, see AND..

3.3.4.5 ANDER, GETBUF

Purpose

To perform Boolean manipulations on two lists

Description

ANDER is a very complex, general purpose Boolean operation procedure called by the search module (when multi-word search terms must be intersected) or by the Boolean command procedures described in Sections 3.3.4.1-4. ANDER takes two reference lists and compares their document numbers (and sometimes term numbers). A new list is produced from the two input lists, the contents of which depends upon the Boolean operation being performed. The new list is deposited into the user's Dump File and a new augmented list pointer is added to the list pointer table.

A. Operation: NP = ANDER(P1, P2, MAND, S)

ANDER accepts four arguments passed from the calling routine. The first and second are pointers to augmented list pointers. These in turn point to two reference lists which are to be processed in some Boolean operation. The third argument contains a mode code which determines which Boolean operation is to be performed. The fourth argument is a flag stating whether or not attribute screening is necessary on the references before they are compared by ANDER.

The first step taken by ANDER is to extract from the two augmented list pointers (and place into local variables, BLK1 and BLK2) the block (common buffer) numbers employed to hold the core-stored portion, if any, of the list. These are used later by the procedure GETLIS (Section 3.3.4.6) to select the buffers to be used for reading.

Then, counters and indices are reset to zero, and the current disk block counts of both the Dump File and the Name File are obtained from the POT.

The flags used by ANDER are reset and the list pointed to by argument pointer one is prepared for processing. This is done by calling GETLIS with this pointer in argument one. GETLIS will read the first block of list one into a common buffer determined by the last two arguments, BLK1 and BLK2, and fill the several other argument words of the call with data pertaining to the address, type, file names, etc. of the list.

One of the items of data supplied by GETLIS is the number of disk-stored references left to be read. If this number is zero, then the end-of-file flag (EOF1) is set to indicate that no further reading of list-one is necessary.

Preparation for list-one examination is completed by setting the core index of list-one (IX1) to zero.

Next, the second argument pointer is examined. If it is a zero, this indicates that ANDER is being called by the procedure NAME (Section 3.3.2.2) for the sole purpose of writing a list into the Name File. This simpler use of ANDER is controlled by setting a flag called FAKE, which prevents the reference-comparison apparatus from being used.

When the second pointer is not zero, the second list is prepared as was the first-by calling GETLIS with th[illegible]r as the first argument--but this time the last two arguments are [illegible]BLK2, BLK1). Again, if the number of disk-stored references re[illegible] to be processed after GE[illegible]LIS is zero, the end-of-file flag (EOF2) is set.

Now the name of the output file, into which will be written the resulting list, is selected. If the FAKE flag is not set, the output file name parameter is set to the name of the Dump File (found in the POT). Otherwise, the output file is the Name File and all the parameters relating to list-two are set to zero.

In either case, the core-address index of list-two (IX2) and the output buffer index (OX) are reset to zero, completing the preparations for comparing the references of the two lists.

GETLIS will normally put the first blocks of references from the lists into common buffers one and three. The next block of references to be read (from one list or the other) will go into common buffer two. This is forced by putting that buffer address into the parameter which controls the reading of the next block (named NEXTB). A corresponding control parameter (named NB) is set to the next buffer number, i.e., 2.

The selection of an output buffer is dependent upon the mode. A title or author search list being combined with a previous search list (mode 1 or 2, respectively) will cause the selection of common buffer six. All other modes will cause buffer four to be chosen for storing output. This distinction is made because title and author lists may be themselves occupying common buffer four.

Main Loop A

Here, ANDER sets an output index parameter (AX) to be used, possibly, by the attribute screening routine BUFSCN (Section 3.3.4.7) to a starting postion of zero.

An important factor concerning the reading of disk files in processing the two reference lists is that time is saved by overlapping the reading time with the processing time. This is accomplished by using RDFILE instead of RDWAIT to read the files. This simply initiates the read and proceeds on to the next instruction. This necessitates keeping the data block being read one step ahead of the block being processed. It also means that, before any new read, a call to FWAIT must be used to ensure that the previous read is finished.

A call to FWAIT is made at this point in ANDER if file names have been set by GETLIS. Then the fourth argument to ANDER is tested to see if attribute screening was requested by the search routine. If so, BUFSCN is called with the following four arguments: 1. the core address of the current block of references of list-one; 2. the number of reference words in this block; 3. the same core address as 1, to be used as the output area by BUFSCN; and 4. the output area index set up especially for BUFSCN(AX). The group of in-core references is reduced by BUFSCN to contain only those which contain the qualifying attributes specified by the user's search request (RANGE, TITLE, and AUTHOR with initials are the only possible attribute specifiers at present). The value returned by BUFSCN is the new reduced length. (Note that this is only the length of the core-stored block, not of the entire list.) If this length is zero, meaning all references in the block were rejected, or if the FAKE flag is on, then the following test for determining which list's block will be used up first is skipped by transferring directly to the point where list-one is set up for the next read ("List-One Loop").

Main Loop B

A call to FWAIT must also be made at this point since the exhaustion of list-two core references will transfer control back to this point rather than to Main Loop A.

Here a decision is made as to which list block will be exhausted first and, therefore, which list file will need to be read from next. This decision is made by examining the document numbers of the last reference in each core block. Since the lists are in descending numerical order of document numbers, the one which ends with the highest number will be exhausted first.

List-Two Loop

If list-two is due to be emptied first, an indicator (FST) is set to If the end-of-file flag is already set for list-two, control then branches t ie point where references are compared ("Compare Documents").

Otherwise, the file names of list-two (as established by GETLIS) are prepared as the next file to be read, and its disk address (RELLOC) is updated.

List-One Loop

If list-one is due to be emptied first, the FST indicator is set to 1. If list-one's end-of-file flag is already set, control branches to the reference comparison point ("Compare Documents").

Otherwise, the file names of list-one (as established by GETLIS) are prepared as the next file to be read, and its disk address (RELLOC) is updated.

Then the file whose name was prepared to be read next, if any, is read (via a call to RDFILE) into the next selected common buffer (up to 432 words). (If the file being read is the Dump File and if output has previously been written into the Dump File beyond the second 432-word record, then a rare and insidious CTSS file handling bug, involving alternate reads and writes of dually opened files, must be avoided by issuing a special call to the RDWAIT procedure. This read must be into the first half of the file and not on a record boundary. It serves to avoid confusing the chaining mechanism of CTSS file handling logic.)

Compare Documents

Before comparing references from the two lists, the FAKE flag (which indicates a pseudo-anding is being done by the NAME procedure) is checked. If this flag is set, control is at once transferred to a point further on in ANDER where the reference is extracted from list-one and inserted into the output buffer ("Take B").

If FAKE is not set, the length of the core-stored references in list-one is checked. If it is zero (because of an attributed screen which discarded them all), control passes to the point where the list-one index is incremented and checked ("Bump List One"). This will set up another read (if there are more references on the disk for this list) and cycle back to "Main Loop A".

Having established the existence of two lists of references, the document number of the next one pointed to by each list index is extracted.

If the flag (FIN) which indicates that one of the lists has been completely scanned is set, then transfer is made to the point where document numbers are counted prior to including the reference in the output list ("Count Docs").

If FIN is NOT set, the document numbers just extracted are compared. If they are the same, control is transferred ahead to the point where term numbers may be compared ("Compare Terms"). If they are not the same, then the mode will determine which course of action is to be taken.

If the NOT mode is active (0) and if list-one's document number is greater than that of list-two, transfer is made to the list-one index incrementation ("Bump List One") which skips to the next reference on the list governing the NOT screen. If list two's document number is larger, then this reference isn't on the NOT list and must be included in the output by transferring ahead to the point where document numbers are counted and references inserted into the output buffer ("Count Docs").

If the OR mode is active (4), then a reference-selection-control-flag (TX) is set (if list one's document is larger) or reset (if smaller). Again, ANDER transfers to the document counting area, Count Docs.

If ANDER reaches this point, then the mode must be either 0 or 3 (one of the modes of Boolean AND). The document numbers of the two lists are compared to see which is the larger.

If the mode is 3 (the weaker AND), then the document number which is larger must be compared with the immediately preceding one on the list to determine if the present one is a continuation of the same document number group. If so, that reference must be added to the output list. This is accomplished by setting (list-one document number greater) or resetting (list-two document number greater) the reference-selection-control-flag TX and branching ahead to "Take B".

If this document number is different from the previous one, or if the mode is 0 (the stronger AND), then ANDER transfers ahead to the appropriate list incrementation section, depending upon which document number is larger.

Bump List-One

The next reference word in list-one is made accessible by adding one to its index IX1. If the index is then not beyond the last reference in core, transfer is made back to the start of the compare loop ("Compare Documents").

When the index reaches the end of the core-stored references, the end-of-file flag is tested to determine if any more references are waiting to be read from a list-one disk file.

If EOF1 is set, all references in this list have been processed. In either mode of ANDing, this means that the comparison is completed and ANDER transfers to the wrap-up section ("Finish"). If the mode indicates ORing or NOTing is in progress, however, then the other list must be completed and added to the output list. A check is made to determine if, by coincidence, list two is already exhausted. If so, transfer is made to "Finish". If not, the reference-selection-flag, TX, is reset to false to force the selection of list-two references to completion. Then the flag FIN, which indicates that one list is done is set. Finally, the last core-reference address of list-one is zeroed out to force the main loop to follow the path which selects and processes list-two. The address immediately following the last reference is also zeroed to prevent undesirable document number matching. ANDER then loops back to get the next block of list-two references ("Main Loop·B").

If, upon exhausting the current in-core block, the e -of-file flag has not been set, preparation is made for the reading and processing of the next block of references from the disk file. The parameter (RL1) which holds the number of references left to be read is examined to see if at least one more full block (432 words) remains. If not, the new core-block length (CL1) is set to the number of disk references remaining, and the disk reference count is set to zero. Otherwise, the core-block length is set to the full block size of 432 and the disk reference count is reduced by that amount.

Now the core-block address is set to the location contained in the next-block parameter, NEXTB, and a new value for NEXTB is obtained from GETBUF, a sub-procedure of ANDER which accepts an argument containing the common buffer number just exhausted and returns the address of the corresponding buffer. This action ensures that the next buffer to be used will be the one just emptied.

Next, the core index of list-one is reset to zero. Adjustments are made to the core address, core length, and disk reference count if the list is an Inverted File-stored list. These adjustments are necessary to compensate for the presence of a section header at the top of each block of Inverted

File references.

With preparations completed, ANDER now loops back to "Main Loop A" to screen attributes on the new block (if necessary) and start a read of the next one.

Bump List-Two

This section of ANDER coding is almost identical to the one just described under "Bump List-One". Of course, the list indices, counts, flags, etc. are those related to list-two in this case. If the entire list has been exhausted and the mode is OR, necessitating the continued processing of list-one, then the reference-selection-flag TX is set to TRUE to force the selection of list-one. (Note that, unlike "Bump List-One," the NOT mode does not require further processing of the other list, since list-two is the only one containing potentially useable references). Transfer is made from here to "Main Loop B" (as it was in "Bump List-One") to start a read of the next block of list-one references.

On the other hand, if more list-two references remain to be read from the disk, the buffer and core-address settings (described above) control the reading of the next block of list-two when transfer is made back to "Main Loop B".

Compare Terms

Certain modes require comparison of term numbers when document numbers have been found to match. One which does not is that used for intersecting title references with subject references. This is indicated by a mode of 1 sent from the calling procedure SEARCH, (Sections 3.2.4.1 and 2). When this mode is detected at "Compare Terms", transfer is immediately made to "Count Docs" without any term comparison being done.

Another mode requiring no term matching is the "strong" NOT (with the AND before it) whose mode is -3. This mode requires that all references with matching document numbers be dropped from the output list regardless of term numbers, so ANDER transfer to "Bump List Two" to get the next reference.

Any other mode will cause the extraction and comparison of the two term numbers. The action taken after comparison depends again upon the specific mode. The "weak" NOT (-1) and the "strong" AND (0) will cause ANDER to skip the reference by transferring to "Bump List-One" if the list-one term number is less (earlier) than the list-two term. If the list-one term is greater (later) than the list-two term, one of two actions may be taken. If the mode is 0, then the list-two reference is skipped by transferring to "Bump List Two". If the mode is -1, the "weak" NOT allows the different term from the same document by transferring to "Count Docs". If the term numbers are equal, the "weak" NOT mode will cause transfer to "Bump List-Two" which omits this reference from the output list, while "strong" AND mode will accept the reference in the output list by transferring to "Count Docs".

In the OR (4) mode, the term numbers are compared and the reference-selection-flag TX is set or reset depending upon whether the list-one term is less-than-or-equal-to or greater-than the list-two term, respectively.

Count Docs.-

The number of different document numbers involved in the output list is tabulated by "remembering" the previous number added to the list and comparing it here to either list-one's or list-two's document number, depending upon the setting of the indicator TX. When the number is the same as the previous one, transfer is made ahead to "Take A". If they differ, the document count is incremented by one and the present document number becomes the new "previous" one.

Take A

Before the current reference is added to the output list, one more mode test is made. If the mode is OR (4) and both the document and term numbers of the two list's current references are identical, then neither reference is chosen for insertion into the output list on this cycle through the comparison loop. Omission is caused by transferring ahead to "See Mode." After one list index is advanced, then if the two current references differ, the remaining similar reference is selected. This avoids duplicity of references.

Take B

The decision as to which list will provide the current reference to be added to the output list is determined by either the "pseudo-AND" flag FAKE or the reference-selection-flag TX. If either of these flags is set, the reference is extracted from list-one. If both are reset, the list-two reference is taken. In either case, as soon as the transfer of the reference into the output buffer has been made, the output index (OX) is incremented by one and a test is made to see if the buffer (432 words) is now full.

When the buffer is full, the CTSS utility procedure WRWAIT is called to write the contents of the buffer into the output file. Then the written-reference counter OUTCNT is incremented by 432 and the output buffer index OX is reset to zero.

See Mode

Having completed the comparison and selection or rejection of one reference, ANDER determines which list index should be incremented by again examining the mode.

A mode of 4 (OR) will cause ANDER to use the reference-selection-flag TX to determine which of the two references was just used and, therefore, which list index should be next incremented.

All the other modes except 0 (strong AND) will cause transfer to "Bump List-Two" to get the next list-two reference.

The zero mode increments both of the list indices. Rather than test for the end-of-buffer in both incrementations, only the one which is scheduled to run out first, as determined back at the beginning of "Main Loop B", must be tested. This is detected here by examining the earlier set flag, FST. If FST is 1, then list-one will be exhausted first so the list-two index is incremented here, followed by a transfer to "Bump List-One". If FST is 2, then list-two will run out of reference first and the reverse procedure will be followed.

Finish

When all relevant references have been processed, the output list must be completed and the new list pointer must be constructed and entered into the list pointer table. First, the current output buffer index, OX is added to the written-reference counter OUTCNT to update the total number of references in the output list.

Next, the disk files, if any, holding the lists may need to be closed, depending upon the type of list involved and the mode of ANDER.

If OUTCNT, the number of output references selected, is zero, then ANDER immediately returns to the calling program with a zero for a return value. Otherwise, the current output block is written, using WRWAIT, into the output file. (Note that the entire 432-word block is written, even if only partially filled.)

Now, the block count in the POT, which indicates the size of the user's Dump File, is updated by adding the number of blocks written by this call to ANDER. (If the pseudo-anding mode called by the Name procedure is in use, the Name File block count is updated instead.)

An augmented list pointer is now constructed within a local array containing all the relevant data pertaining to the size and location of the new list. This pointer is then added to the table by calling the procedure TABENT (see Section 3.3.2.3), which returns a pointer to the location in the table where it inserted the augmented pointer. The number of documents tabulated by the "Count Docs" section is inserted into the decrement of the new pointer making it a complete list pointer, which is then returned as a value to the calling program.

B. Procedures Calling ANDER:
 STRCH, ASRCH, AND., NAME

C. Procedures Called By ANDER
 GETLIS, FWAIT, BUFSCN (via CALLIT), RDWAIT, RDFILE, GETBUF, WRWAIT, CLOSE, TABENT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
TOTBLK(POT.)	Dump File block count	x	x
TOTNAM(POT.)	Name File block count	x	x
DFN1(POT.)	Dump File name-one	x	
NFN1(POT.)	Name File name-one	x	
COMBF2(POT.)	common buffer two	x	
COMBF4(POT.)	common buffer four	x	
COMBF6(POT.)	common buffer six	x	

E. Arguments:

P1:	address pointer
P2:	address pointer
MAND:	integer
S:	Boolean

F. Values:

NP = pointer to new list

G. Error Codes:

NP = -1 (List table full)

H. Messages:

None (see GETLIS below)

I. Length:

1655_8 or 941_{10} words (excluding GETLIS)

J. Source:

BOOL ALGOL

3.4.6 GETLIS

ırpose

› provide access to a reference list

escription

The variety of list types, formats, sizes, etc. greatly complicates e problem of accessing lists for output or comparison by the ANDER ocedure. GETLIS is designed to modularize the operations involved in itiating a read of a list (if disk-stored) and supplying the relevant data the location and size of the list. Although GETLIS is internal to ANDER, is used also by FSO in processing a user's OUTPUT command.

Operation:

GETLIS (P1, T1, CA1, CL1, DA1, RL1, XNAM1, XNAM2,
BLKA, BLKB)

The current length of the Dump File is obtained by multiplying the ock count in the POT by 432, the number of computer words in a disk cord. This will be used to determine the necessity of avoiding a CTSS e reading bug as described below.

The only argument (of the ten employed) which passes informa- on to GETLIS rather than accepting information from it, is P1. This argu- ent contains a pointer to a reference list augmented pointer. The tag of the cond word of this augmented pointer, containing the list type, is extracted d placed in the second argument of GETLIS, T1.

If the type is 0, then GETLIS is dealing with a completely core- ored reference list and there is no disk file to be read. The seventh argu- ent XNAM1 (file name one), and the corresponding ANDER parameter hich holds the current, to-be-read file name (LN1) are set to zero. Also t to zero is the sixth argument RL1, which holds the number of disk- ored references left to be read. In this case, GETLIS jumps ahead to the d of the procedure where the third and fourth arguments, core address list and core length of list, respectively, are filled from the augmented st pointer.

Any other type of list involves at least some disk-stored references. The disk index (RELLOC) and the number of disk-stored references are extracted from the augmented pointer and inserted into arguments five and six, respectively.

If the list type is 4 (a NAMEd list), then the Name File's first name is copied from the POT into the seventh argument and into the local parameter LN1. The eight argument and local LN2 are set to contain the word FILE, which is always the last name of a Name File (or Dump File). A call to FSTATE is made to determine if this file is already open for reading. If not, the procedure OPEN is called.

If the list type is 2 (stored on the Dump File), then the Dump File name is extracted from the POT and inserted into the seventh argument and LN1. No FSTATE call is needed in this case because the Dump File is always open for both reading and writing.

In both cases of completely disk-stored lists, the first block of references is read into core and occupies one of the common buffers not already being used. A fairly elaborate system of common buffer bookkeeping is carried on in IFSRCH (Section 3.2.5.1), ANDER (previous Section), and GETLIS, using the common buffer numbers stored in the augmented list pointers. These block numbers are extracted by ANDER (called BLK1 for list-one and BLK2 for list-two) and both are passed to GETLIS as arguments nine and ten (BLKA and BLKB). When list-one is to be read, BLK1 is first and becomes BLKA in GETLIS. When list-two is to be read, BLK2 is first and becomes BLKA in GETLIS. When GETLIS is called from FSO or NUMBER, buffer four or five, respectively, is passed via argument nine to become BLKA. Argument ten ,BLKB, is zero.

GETLIS now examines argument nine, BLKA and, if it is non-zero, leaves both block numbers as they are. If, however, BLKA is a zero, it indicates that the call to GETLIS is from ANDER, rather than FSO or NUMBER, and that there are no references from this list in core. Therefore a common buffer must be selected by placing a buffer or block number in BLKA. The object is to read list-one into buffer one and list-two into buffer three. On the first call to GETLIS, BLK2 is BLKB and is also empty. This prompts GETLIS to put a 1 into BLKA (i.e., BLK1). On the second call to GETLIS, BLK1 is BLKB and will not be empty. This informs GETLIS to put a 3 into BLKA (i.e., BLK2).

Now BLKA is used to compute the actual core address of the selected common buffer, which is then placed in argument three, CA1.

If the list type is 2 (in Dump File), action must be taken to avoid the CTSS file chaining bug, involving files which are opened simultaneously for reading and writing, by making an extra read into the first half of the file at a position other than on a record boundary.

For either list type (2 or 4), RDWAIT is now called to put the first block of disk-stored references into core in the selected common buffer. The disk index, argument five, is incremented by the size of the block(432) and the number of disk-stored references remaining, RL1, is reduced. If the resulting number is less than the size of a block, then that number is used to fill the fourth argument (number of core-stored references) and the disk reference count is reduced to zero.

If the number of disk references is presently greater than or equal to the size of a block, then the fourth argument, CL1, is set to 432 and the disk reference count is reduced by that same amount.

In the case of list types 2 and 4, GETLIS is now finished and returns to the calling program.

If the list type is 1, indicating a list which is partially in core, with the rest of the references in an Inverted File segment, then the first name of that segment must be extracted from word one of the augmented list pointer. This name is deposited into argument seven and the local parameter LN1 (for use by ANDER). The second name of the segment is obtained from the POT and placed in argument eight and LN2. A call to FSTATE is made to determine if this file is currently open for reading. If not, it is opened by calling the CTSS procedure OPEN.

At this point the address and length of the core-stored portion of the list is extracted from the third word of the augmented pointer and deposited in the third and fourth arguments of GETLIS. (These steps also serve in processing type 0 lists as mentioned earlier.) This completes the processing of all legitimate list types and GETLIS returns to the calling program.

If the type code taken from the augmented list pointer is other than those described above, GETLIS transfers to an error exit where a message (1) informs the user of the error, and processing is terminated by a call to DORMNT.

B. Procedures Calling GETLIS:

ANDER, FSO, NUMBER

C. Procedures Called By GETLIS:

FSTATE, OPEN, RDWAIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
TOTBLK(POT.)	Dump File block count	x	
NFN1(POT.)	Name File name one	x	
DFN1(POT.)	Dump File name one	x	
COMBF1(POT.)	Common buffer one	x	
IFS2(POT.)	Inverted File name two	x	

E. Arguments: (→ means argument gives data to GETLIS. ← means argument accepts data from GETLIS.)

1.	P1:	→	address pointer
2.	T1:	←	list type
3:	CA1:	←	core reference address
4.	CL1:	←	core reference length
5.	DA1:	←	disk file index
6.	RL1:	←	disk references left
7.	XNAM1:	←	list file name one
8.	XNAM2:	←	list file name two
9.	BLKA:	←	this list's core buffer
10.	BLKB:	←	other list's core buffer

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Invalid List type in pointer" (LOCMES)

I. Length:

374_8 or 252_{10} words

J. Source:

BOOL ALGOL

K. Files Referenced:

DUMnnn	FILE
NAMnnn	FILE
SInnn	date
AInnn	date

3.3.4.7 BUFSCN

Purpose

To screen list by attributes

Description

When the user specifies a RANGE in a SUBJECT command, or when he issues a TITLE command (which is really like a SUBJECT with range 5), or when he issues an AUTHOR command qualified by initials, the specified attributes are set during the interpretation phase to be used in screening the resulting reference list. This screening is done either during execution of the SEARCH procedure via ATSCRN (which in turn, calls BUFSCN to perform the actual screen - see Section 3.2.4.3) or during execution of ANDER, which calls BUFSCN directly). BUFSCN examines one core array of references and compares the appropriate component of the reference word to the specified attribute. Only those references with the matching attributes are kept in the array.

A. Operation: NWDS = BUFSCN (INAD, WDS, OUTTAD, OX)

BUFSCN consists of one large processing loop within which is another loop almost as large. The first, or outer loop steps one-by-one through the list of references supplied by the first two arguments of BUFSCN. The first argument specifies where the reference group begins and the second argument tells how many there are to be examined, thus controlling the duration of this outer loop.

The second, or inner loop steps through the attribute list as set up during the interpretation of the user's search request and prepared locally by IFSRCH (which is the reason for compiling BUFSCN and ATSCRN in the same file as IFSRCH).

Each reference word is compared to each attribute word with the irrelevant parts of the reference words masked off. Further action is taken only if the two remaining parts are equal.

A reference which contains the required attributes is stored in the array designated by the third argument of BUFSCN at the depth specified by the index in the fourth argument. When BUFSCN is called by ANDER, this index is used merely to control the output storage (which is the same area as the input) and to record the length of the output list. When called from ATSCRN, however, the index is carried over from one call of BUFSCN to the next until the output buffer is full (432 words). As the references are stored in the output area, the document numbers are counted by comparing each number to the previous one. Whenever a different number is encountered, the counter is incremented. This count is used only by ATSCRN and is not communicated to ANDER.

If the output buffer is filled, its contents are written onto the Dump File and the block counter stored in the POT is incremented. At this point, the output index (fourth argument) is reset to zero and the parameter containing the number of references written is incremented by 432 for the benefit of ATSCRN.

After the outer loop has stepped through all the references in the input block, the output buffer index is returned to the calling routine.

B. Procedures Calling BUFSCN:

ANDER, ATSCRN

C. Procedure Called By BUFSCN:

WRWAIT

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
DFN1(POT.)	Dump File name one	x	
TOTBLK(POT.)	Dump File block count		x

E. Arguments:

INAD:	address pointer
WDS:	integer
OUTAD:	address pointer
OX:	integer

F. Values:

NWDS = length of screened list

G. Error Codes:

None

H. Messages:

None

I. Length:

170_8 or 120_{10} words

J. Source:

IFSRCH ALGOL

K. Files Referenced:

DUMnnn FILE

3.4 Intrex Utilities

3.4.1 Free Storage Controls

3.4.1.1 FREE

Purpose

To obtain memory block from free storage

Description

FREE locates a block of contiguous core locations of the size requested, removes the block from the free storage chain, and returns a pointer to it.

When the first call is made to FREE, the memory bound is raised to the top of memory, creating a free storage area. As storage is requested, blocks are taken from progressively higher locations in this area. When storage is returned to this area, its location is entered in the first element in the chain, a word in the free storage package. At the same time, the length of the area returned is entered in the decrement of the first word of this area. If a second block, higher than the first, is returned, the address portion of the first word of the first block will be filled with the location of the second block and the length of the second block will be inserted in the decrement of the first word of the second block. If the second block is between the pointer in the free storage package and the first block returned, the first word of the second block is modified to contain its length and the location of the first block. The original pointer is modified so as to point to the second block instead of the first block.

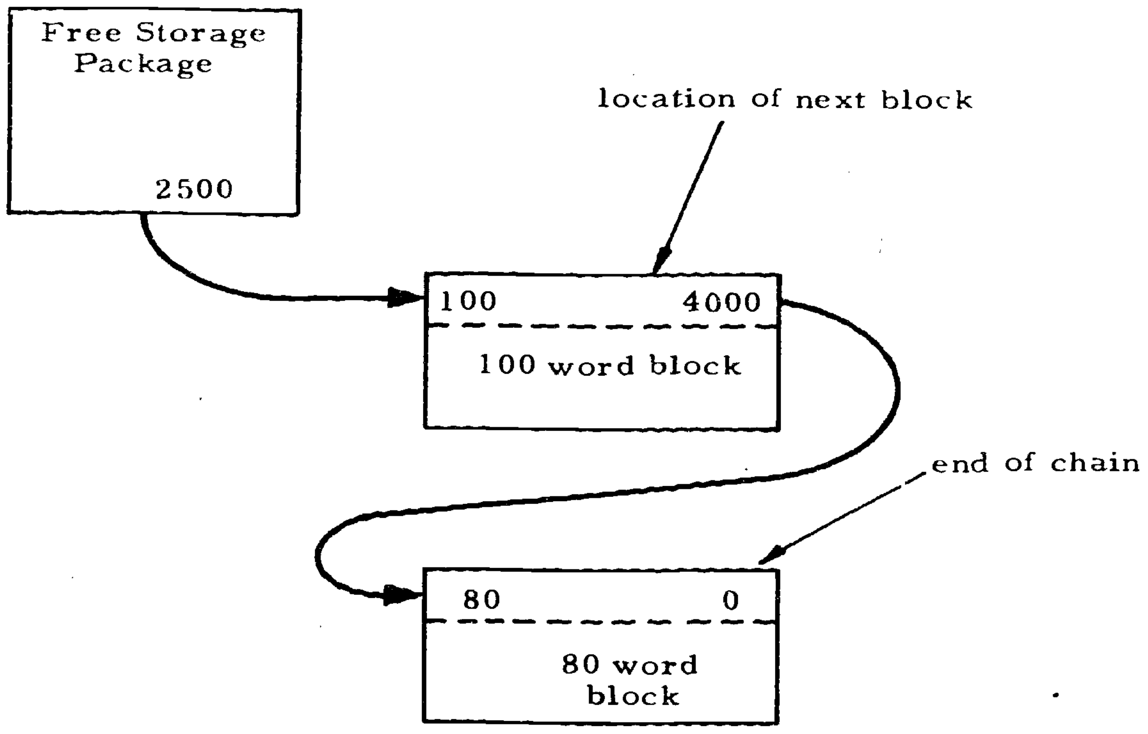


Diagram of Memory

Fig. 3.3

A. Operation Ptr = FREE (Len)

1. If Len is less than or equal to zero, FREE prints an error message by means of a call to CNTLOC and then calls ERRGO (Sect. 3.1.0.1)

2. FREE steps through the linked list of free storage areas, looking for one at least as big as Len. If it finds an area larger than Len, FREE uses the first part of the block to fill the storage request and relocates the pointer in the first word of the block to be just beyond the portion retrieved. This involves adjusting the pointer of the previous block so that it points Len words further and it involves constructing a new pointer for the remnant of the original block.

3. If FREE discovers a block of size Len in its free storage chain, there will not be a remnant left after the request is filled. Therefore, FREE modifies the pointer of the previous block so that it points to the block following the retrieved block.

4. If FREE cannot find a block large enough, it looks for room above the free storage chain. If there is room there, the lower end of this block is allocated. This portion will become part of the free storage chain if it is later returned via FRET.

5. If there is no room available, an error message is printed and FREE returns an error code of -1.

6. If FREE has found a block of length Len, it returns the location of the first word of the block as its value.

B. Procedures Calling FREE:

AUTHOR, CONNAM, GETFLD, IFSINT, INIFIX, INIPOT, INITYP, INIVAR,IN.,'SYSGEN, TABLE, TYPASH

C. Procedures Called By FREE:

CNTLOC, ERRGO, GETMEM, OCTTOI, SETMEM, WFLX, WFLXA.

D. COMMON References:

None

E. Arguments:

Len: binary integer

F. Values:

Ptr = location of first word of block

G. Error Codes:

Ptr = -1: storage not available.

H. Messages:

1. "Illegal use of 'free', count is zero or negative" (preset)
2. "Short $\underline{x}$ words of free storage
   Largest block is $\underline{y}$
   Total storage is $\underline{z}$" (preset)

I. Length:

250_8 or 168_{10} words

J. Source:

FREED ALGOL

K. File References:

None

3.4.1.2 FRET

Purpose

To return memory to free storage.

Description

When a block of storage obtained via the free storage mechanism is no longer needed, it can be returned to the common storage pool by means of a call to FRET.

A. Operation: FRET(Len, Ptr)

A subroutine calls FRET to return a block of storage of length Len located at Ptr.

1. If Len is less than zero, an error message is printed. If Len is zero, FRET returns without doing anything.
2. FRET steps through the free storage chain, looking for a block whose address is greater than the one being returned.
3. If the block being returned is above those already in the chain, the last pointer in the chain is modified so as to point to this new last block. However, if the block which was formerly the last block in the chain overlaps with the block being returned, an error message is printed and nothing else is done. If the block being returned is contiguous with the last block, the two are merged into one.
4. If the block being returned has a lower address than a block already in the chain, it is inserted in the chain. If the block being returned overlaps with the block in front of it, an error message is printed and nothing is done. If it overlaps with the block behind it, a comment is made and it is merged with this block. If it touches neither the one in front of it or behind it, the pointer associated with the block in front is modified to point to the new block, and the new block is made to point to the one following it.

B. Procedures Calling FRET:

ASCITC, CHKNAM, CLP, CONNAM, EVAL, FCLEAN, FRALG, GETFLD, GETINT, GETLIN, IFSINT, IFSRCH, ~~INFO~~, INIFIX, INFO, INITYP, NEXITM, NUMBER, PREP, QUIT, RANGE, REND, SUBJ., S.T, TABLK, TITLE, TYPASH, TYPEIT, .C.ASC

C. Procedures Called By FRET:

CNTLOC, ERRGO, GETMEM, OCTTOI, WFLX, WFLXA

D. COMMON References:

None

E. Arguments:

Len:	length of block to be returned (binary)
Ptr:	location of block to be returned (binary)

F. Values:

None

G. Error Codes:

None

H. Messages:

1. "Illegal use of 'fret', count negative" (preset)
2. "An attempt to 'fret' storage already returned"(preset)

I. Length:

251_8 or 169_{10} words

J. Source:

FREED ALGOL

K. Files Referenced:

None

3.4.1.3 FREZ

Purpose

To obtain a zeroed memory block from free storage

Description

A. Operation: Ptr = FREZ(Len)

FREZ calls FREE to obtain a block of storage of size Len. FREZ uses a Fap-coded loop to zero it out before returning the location of the first word of the block as the value of FREZ.

B. Procedures Calling FREZ:

AUTHOR, ASCIT, ASCITC, ASCIT6, BUFSCN, CLP, CTSIT, CTSIT6, EVAL, GETFLD, GETLIN, IFSRCH, INIS.T, INITDB, MONINT, NEXITM, NUMBER, PREP, RANGE, REND, SEARCH, SUBJ., S.T., TABLE, TABLK, TITLE

C. Procedures Called By FREZ:

CNTLOC, ERRGO, FREE

D. COMMON References:

None

E. Arguments:

Len: binary integer

F. Values:

Ptr = core location in binary

G. Error Codes:

Ptr = -1: not enough storage

H. Messages:

1. "Illegal use of 'FREZ'. Count is zero or negative" (preset)

I. Length:

43_{10} or 53_8 words

J. Source:

FREED ALGOL

K. Files Referenced:

None

3.4.1.4 CNTLOC

Purpose

To print count and location of call to erring free storage procedure

Description

CNTLOC is called by FREE and FRET to print a message giving the origin of a call to free storage and the size of the block in question.

A. Operation: CNTLOC( )

1. If the high end of a block being returned via FRET overlaps, CNTLOC prints:

   Count = x Called from location y
   Count adjusted to z

   If the beginning, or low end, of the block overlaps, CNTLOC prints:

   Count = x Called from location y
   FRET call ignored

2. If FREE calls CNTLOC (which is only when Len is less than or equal to zero), only the first line of the message above is printed.

B. Procedures Calling CNTLOC:
   FREE, FREZ, FRET

C. Procedures Called By CNTLOC:
   OCTT0I, WFLX, WFLXA

D. COMMON References:
   None

E. Arguments:
   None

F. Values:
   None

G. Error Codes:
   None

I. Messages:

See Item A in this Section

. Length:

51_8 or 41_{10} words

. Source:

FREED ALGOL

. Files Referenced:

None

3.4.1.5 SIZE

Purpose

To find size of largest block of free storage.

Description

A. Operation: Val = SIZE( )

SIZE steps through the free storage chain, looking for the largest block. The size of the largest block is returned as the value of SIZE.

B. Procedures Calling SIZE:

INIFIX, SUPER

C. Procedures Called By SIZE:

GETMEM

D. COMMON References:

None

E. Arguments:

None

F. Value:

Val = size of largest block in binary

G. Error Codes:

None

H. Messages:

None

I. Length:

101_8 or 65_{10} words

J. Source:

FREED ALGOL

K. Files Referenced:

None

3.4.1.6 FRER

Purpose

To check free storage

Description

FRER is used to determine if a block of free storage of the size desired is available.

A. Operation: Code = FRER (Len)

FRER calls SIZE to find the largest block of free storage available. If this block is smaller than Len, FRER returns a zero; otherwise, it returns a 1.

B. Procedures Calling FRER:

CONNAM, GETFLD, TYPASH

C. Procedures Called By FRER:

SIZE

D. COMMON References:

None

E. Arguments:

Len: size of storage block needed in binary.

F. Values:

Code = 1 - not enough storage

0 - storage is available

G. Error Codes:

None

H. Messages:

None

I. Length:

11_8 or 9_{10} words

J. Source:

FRCED ALGOL

K. Files Referenced:

None

3.4.1.7 FSIZE

Purpose

To count the total number of words of free storage that are available.

Description

A. Operation: Val = FSIZE( )

FSIZE calls SIZE, which, in the course of finding the largest block of storage, also totals all of the blocks together. FSIZE returns this total as its value.

B. Procedures Calling FSIZE:

SUPER

C. Procedures Called By FSIZE:

SIZE

D. COMMON References:

None

E. Arguments:

None

F. Values:

Val = free storage total, in binary

G. Error Codes:

None

H. Messages:

None

I. Length:

14_8 or 12_{10} words

J. Source:

FREED ALGOL

K. Files Referenced·

None

3.4.1.8 FRESET

Purpose

To reset free storage package.

Description

FRESET zeroes out the variables in the free storage mechanism. A subsequent call to FREE will result in storage being allocated from above the current memory bound.

A. Operation: FRESET( )

FRESET zeroes out the first word of the free storage chain, along with 3 other variables.

B. Procedures Calling FRESET:

SYSGEN

C. Procedures Called By FRESET:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

12_8 or 10_{10} words

J. Source:

FREED ALGOL

K. Files Referenced:

None

3.4.1.9 FTRACE

Purpose

To trace use of free storage.

Description

A. Operation: FTRACE(Swt)

If FTRACE is called with a value of 1 for Swt, the tracing mechanism is turned on. When FREZ or FREE is called, the system will print the location of the storage block, its size, and the location of the call to the free storage package, each on a separate line. When FRET is called, the size, the location of the block, and the location of the call are printed, in that order.

B. Procedures Calling FTRACE:

None

C. Procedures Called By FTRACE:

None

D. COMMON References:

None

E. Arguments:

Swt:	1:	turn on trace
	0:	turn off trace

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

6 words

J. Source:

FREED ALGOL

K. Files Referenced:

None

3.4.1.10 FRALG

Purpose

To return a procedure to free storage when it is no longer needed.

Description

Certain procedures, such as initialization routines, are only executed once. FRALG makes available as data areas the memory used by such routines.

A. Operation: FRALG(Label 1, Label 2)

FRALG calls MAINBD to find the boundary between the core-resident section of the retrieval system and the overlay sector. If FRALG is being called by a routine in the overlay sector, FRALG returns without doing anything.

Otherwise, FRALG makes a call to FRET of the form: FRET (Label 2-Label 1, Label 1), where Label 1, Label 2 are labels in the calling procedure defining the first and last locations of the area to be returned.

B. Procedures Calling FRALG:

INIAUT, INICON, INIEVL, INIFIX, INIFLD, INIOUT, INIRNG, INIS.T, MONINT, PREP, SIGNIN, SYSGEN, TABLE

C. Procedures Called By FRALG:

FRET, MAINBD

D. COMMON References:

None

E. Arguments:

Label 1:	beginning location of area to be returned
Label 2:	ending location of area to be returned

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

22_{10} or 26_8 words

J. Source:

FRALG FAP

K. Files Referenced:

None

3.4.2 Code Conversion

3.4.2.1 .C.ASC

Purpose

To generate an ASCII string.

Description

The procedure .C.ASC converts the BCD string defined by the AED language .C. specification into an ASCII string with pointer Ascptr.

A. Operation: Ascptr = .C.ASC(.C./string/, 1)

The .C. form of a string pointer is converted to the .BCI. format and ASCITC is called to convert the string to ASCII. ASCITC will obtain memory for the converted string by a call to FREE. If .C.ASC has a second argument, all of the characters in the string will be converted to upper case. If the original BCD string is not in an overlay segment, it will be returned to free storage via a call to FRET.

B. Procedures Calling .C.ASC:

INIAUT, INICON, INIEVL, INIFLD, INIOUT, INIRNG, INIRES, INIS.T, INIVRB, MONINT, SAVE

C. Procedures Called By .C.ASC:

ASCITC, FRET, GET, INC1, ISARGV, PUT

D. COMMON References:

None

E. Arguments:

Arg: Arg contains the address of the beginning of the BCD string. The word before the beginning of the string contains the length of the string in its decrement.

If lower case ASCII is desired, the second argument should not appear.

F. Values:

Ascptr = ASCII pointer to string

G. Error Codes:

None

H. Messages:

None

I. Length:

146_8 or 102_{10} words

J. Source:

ASCON ALGOL

3.4.2.2 INI.C.

Purpose

To initialize .C.ASC .

Description

A. Operation: INI.C.( )

INI.C. initializes .C.ASC by providing it with the beginning address of the overlay area. .C.ASC will not return the original BCD string to free storage if it lies in the overlay area.

B. Procedures Calling INI.C.:

INIFIX

C. Procedures Called By INI.C.:

MAINBD

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

7 words

J. Source:

ASCON ALGOL

K. Files Referenced:

None

3.4.2.3 INTASC

Purpose

To convert integer to ASCII.

Description

A. Operation: Ascptr = INTASC (Integer)

INTASC converts the binary number Integer to its ASCII equivalent and returns an ASCII pointer Ascptr. The ASCII codes are stored in an array in the subroutine. INTASC handles positive or negative numbers. The magnitude of integer must be less than 100 million.

B. Procedures Calling INTASC:

GETFLD, SEEMAT, TRANS, TYPEIT

C. Procedures Called By INTASC:

INC, PUT

D. COMMON References:

None

E. Arguments:

Integer: binary number to be converted

F. Values:

Ascptr = ASCII pointer to converted number

G. Error Codes:

None

H. Messages:

None

I. Length:

133_8 or 91_{10} words

J. Source:

ASCON ALGOL

K. Files Referenced:

None

3.4.2.4 OCTASC

Purpose

To convert integer to an octal ASCII number.

Description

A. Operation: Ascptr = OCTASC(Int)

OCTASC will convert the binary integer Int into its octal equivalent as expressed in ASCII digits. The ASCII string, resides in an array within OCTASC and is pointed to by Ascptr.

B. Procedures Calling OCTASC:

ERRGO, LISTEN, TYPEIT

C. Procedures Called By OCTASC:

INC, PUT

D. COMMON References:

None

E. Arguments:

Int: Positive or negative integer

F. Values:

Ascptr = ASCII pointer to octal number

G. Error Codes:

None

H. Messages:

None

I. Length:

112_8 or 74_{10} words

J. Source:

ASCON ALGOL

K. Files Referenced:

None

3.4.2.5 TSSASC

Purpose

To convert 12-bit BCD to ASCII.

Description

A. Operation: Achar = TSSASC (Char)

TSSASC converts the BCD character that it finds in the right most 12 bits of Char to ASCII and returns this value to Achar. Char may contain either a 6-bit or a 12-bit BCD character. If Char is not a BCD character, or if it cannot be mapped into ASCII, TSSASC returns a zero.

B. Procedures Calling TSSASC:

ASCIT6, ASCITC, GETLIN

C. Procedures Called By TSSASC:

None

E. Arguments:

Char: 6-bit or 12-bit BCD character, right adjusted

F. Values:

Achar = an ASCII character

G. Error Codes:

None

H. Messages:

None

I. Length:

212_8 or 138_{10} words

J. Source:

CTSS

K. Files Referenced:

None

3.4.2.6 ASCTSS

Purpose

To convert ASCII to 12-bit BCD.

Description:

A. Operation: Char = ASCTSS(Achar)

ASCTSS converts the ASCII character that it finds in the rightmost 9 bits of Achar to its 12-bit BCD equivalent.

B. Procedures Calling ASCTSS:

CTSIT6, TRASH

C. Procedures Called By ASCTSS:

None

D. COMMON References:

None

E. Arguments:

Achar: 9-bit, right adjusted, ASCII character

F. Values:

Char = 12-bit BCD character

G. Error Codes:

None

H. Messages:

None

I. Length:

7 words

J. Source:

CTSS

K. Files Referenced:

None

3.4.2.7 ASCIT6

Purpose

To convert string of 6-bit characters to ASCII.

Description

A. Operation: Ascptr = ASCIT6(Bcdptr)

ASCIT6 converts the string of 6-bit characters pointed to by Bcdptr into a string of upper case ASCII characters pointed to by Ascptr. ASCIT6 allocates storage for the ASCII string by a call to FREE. The characters pointed to by Bcdptr are extracted one by one (using GET6), converted to ASCII (using TSSASC) and added to the new string by means of PUT.

B. Procedures Calling ASCIT6:

MONTOR

C. Procedures Called By ASCIT6:

FREE, INC, INC6, PUT, TSSASC

D. COMMON References:

None

E. Arguments:

Bcdptr: Address portion contains beginning location of string and decrement portion contains number of 6-bit bytes.

F. Values:

Ascptr = ASCII pointer

G. Error Codes:

None

H. Messages:

None

I. Length:

104_8 or 68_{10} words

J. Source:

STRAND ALGOL

K. Files Referenced:

None

3.4.2.8 CTSIT6

Purpose

To convert ASCII string into 6-bit BCD strings

Description

A. Operation: Bcdptr = CTSIT6 (Ascptr)

CTSIT6 converts the string of ASCII characters pointed to by Ascptr to a string of 6-bit characters pointed to by Bcdptr. CTSIT6 allocates storage for the BCD string by a call to FREE. Each ASCII character is converted by a call to ASCTSS. The pointer returned contains the number of characters in its decrement.

B. Procedures Calling CTSIT6:

AND., CHKNUM, INFO, QUIT, SIGNIN

C. Procedures Called By CTSIT6:

ASCTSS, FREE, GET, INC, INC6, PUT

D. COMMON References:

None

E. Arguments:

Ascptr: ASCII pointer

F. Values:

Bcdptr = 6-bit BCD string pointer

G. Error Codes:

None

H. Messages:

None

I. Length:

113_8 or 75_{10} words

J. Source:

STRAND ALGOL

K. Files Referenced:

None

3.4.2.9 ASCITC

Purpose

To convert 6-bit BCD string to lower-case ASCII.

Description

A. Operations: Ascptr = ASCITC (Bcdptr)

ASCITC functions the same as ASCIT6, (see Section 3.4.2.7), except that the converted ASCII characters are usually lower case instead of upper. However, if a 6-bit BCD character in the string pointed to by Bcdptr is preceded by a $, the character is converted to upper case.

B. Procedures Calling ASCITC:
BCDASC, LOCMES, PREP, .C.ASC

C. Procedures Called By ASCITC:
FREE, FRET, GET6, INC, INC6, PUT, TSSASC

D. COMMON References:
None

E. Arguments:
Bcdptr: 6-bit BCD string pointer

F. Values:
Ascptr = ASCII pointer

G. Error Codes:
None

H. Messages:
None

I. Length:
154_8 or 108_{10} words

J. Source:
STRAND ALGOL

K. Files Referenced:
None

3.4.2.10 ASCINT

Purpose

To convert ASCII number to binary.

Description

A. Operation: Val = ASCINT (Ptr)

ASCINT converts the ASCII-coded number pointed to by Ptr to a binary number. ASCINT calls CHKNUM to check for the letters ℓl and O. If ASCINT finds a non-numerical character, it returns a value of -1.

B. Procedures Calling ASCINT:

CALLIT, IN., NUMBER, OUT., RANGE, TABLE

C. Procedures Called By ASCINT:

CHKNUM, GET, INC

D. COMMON References:

None

E. Arguments:

Ptr: pointer to number expressed in ASCII

F. Values:

Val = binary number

G. Error Codes:

Val =-1: bad argument

H. Messages:

None

I. Length:

171_8 or 121_{10} words

J. Source:

ASCINT ALGOL

K. Files Referenced:

None

3.4.2.11 CHKNUM

Purpose

To check for ℓ's and O's.

Description

A. Operation: CHKNUM(Ptr)

CHKNUM checks the ASCII-coded number pointed to by Ptr for the presence of non-numerical characters. If the character is the letter O or ℓ it prints an error message. If the character is some other letter, CHKNUM returns to the calling program.

B. Procedures Calling CHKNUM:
ASCINT, CALLIT, INFO

C. Procedures Called by CHKNUM:
GET, INC1, PUT, TYPEIT

D. COMMON References:
None

E. Arguments:

Ptr: ASCII pointer to number

F. Values:
None

G. Error Codes:
None

H. Messages:

1. "You have typed the letter ℓ in place of a one (1) in the command argument A. Your command will be processed as if a 1 had been typed. Please observe this distinction. Failure to do so may cause matching difficulties." ($loerr2$, $loerr0/1$, $loerr3$, $loerr4$, $loerr5$, $loerr6$)

I. Length:
166_8 or 118_{10} words

J. Source:
CHKNUM ALGOL

K. Files Referenced:
None

3.4.3 File Manipulation

3.4.3.1 FILCNT

Purpose

To give length of file.

Description

A. Operation: Len = FILCNT(Name1, Name2)

FILCNT calls FSTATE to find the length of a file. If the file cannot be found, FILCNT returns a -1 instead of the length.

B. Procedures Calling FILCNT:

CHKSAV, CLEANP, CONDIR, CONNAM, GETLIN, IFSINT, INIRES, LISFIL, LISTSL, PREP, REND, TABLE

C. Procedures Called By FILCNT:

FSTATE

D. COMMON References:

None

E. Arguments:

Name1, Name2: name of file (6-bit BCD)

F. Values:

Len = length of file in words

G. Error Codes:

Len = -1: file not found

H. Messages:

None

I. Length:

52_8 or 42_{10} words

J. Source:

AEDLBJ BSS

K. Files Referenced:

COMAND TABLE
FIELDS TABLE
ENDING TEST2
HOLD UP
HOLD IT
IFDS date
IFDA date
DUMnnn FILE
NAMnnn FILE
PASnnn FILE
name SAVED
SAVED DIRECT

3.4.4 Character-String Manipulation

3.4.4.1 GET

Purpose

To get ASCII character.

Description

A. Operation: Char = GET(Ascptr)

GET will extract the 9-bit ASCII character pointed to by Ascptr and set the contents of Char to this value.

B. Procedures Calling GET:

ASCINT, AUTHOR, CHKNUM, COMPAR, COMPUL, CTSIT, CTSITC, FIND, GETLIN, IFSRCH, IN., NAM5, NEXITM, OUT., SIGNIN, STEM, TABLK, TYPEIT, .C.ASC

C. Procedures Called By GET:

D. COMMON References:

None

E. Arguments:

Ascptr: ASCII pointer to character string

F. Values:

Char = right-adjusted 9-bit character pointed to by Ascptr

G. Error Codes:

None

H. Messages:

None

I. Length:

20_{10} or 24_8 words

J. Source:

GET FAP

K. Files Referenced:

None

3.4.4.2 INC

Purpose

To increment pointer.

Description

A. Operation: INC(Ascptr, Shift)

INC adjusts the ASCII pointer Ascptr by the amount (+ or -) specified by Shift. The decrement of Ascptr is left unchanged.

B. Procedures Calling INC:

ASCINT, ASCIT, ASCIT6, ASCITC, CTSIT, CTSIT6, EVAL, FSO, GETFLD, GETLIN, INTASC, MATCH, NEXITM, OCTASC, SEEMAT, SPCTRN, STEM, TABLE, TABLK, TRANS, TRASH, TYPEIT.

C. Procedures Called By INC:

None

D. COMMON References:

None

E. Arguments:

Ascptr:	ASCII pointer to character string
Shift:	number of bytes by which Ascptr is to be adjusted (+ or -, binary)

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

27_{10} or 33_{8} words

J. Source:

INC FAP

K. Files Referenced:

None

3.4.4.3 PUT

Purpose

To insert ASCII character.

Description

A. Operation: PUT(Char, Ascptr)

PUT will insert the rightmost nine bits of Char in the byte location indicated by Ascptr.

B. Procedures Calling PUT:

ASCIT, ASCIT6, CAPASC, CHKNUM, EVAL, GETFLD, GETLIN, INTASC, IN., NEXITM, OCTASC, OUT., SEEMAT, SIGNIN, TABLE, TRANS, .C.ASC

C. Procedures Called By PUT:

None

D. COMMØN References:

None

E. Arguments:

Char:	9-bit ASCII character to be inserted
Ascptr:	ASCII pointer to byte position to be filled by "Char".

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

25_{10} or 31_8 words

J. Source:

PUT FAP

K. Files Referenced:

None

3.4.4.4 GET6

Purpose

To get 6-bit character.

Description

A. Operation: Char = GET6(Ptr)

GET6 extracts a 6-bit BCD character pointer to by the BCD pointer Ptr and return it as the value of Char.

B. Procedures Calling GET6:

ASCITC, ASCIT6, PREP

C. Procedures Called By GET6:

None

D. COMMON References:

None

E. Arguments:

Ptr:	address =	location of beginning of BCD string
	tag =	first byte position (0-5)
	decrement =	length of string in bytes

F. Values:

Char = 6-bit character, right adjusted

G. Error Codes:

None

H. Messages:

None

I. Length:

22_{10} or 26_8 words

J. Source:

GET6 FAP

K. Files Referenced:

None

3.4.4.5 INC6

Purpose

To increment BCD pointer.

Description

A. Operation: INC6(Ptr, Count)

INC6 increments Ptr by the number of bytes specified in Count. If Count is negative, the pointer is backed up.

B. Procedures Calling INC6:

ASCITC, ASCIT6, PREP, TRASH

C. Procedures Called By INC6:

None

D. COMMON References:

None

E. Arguments:

Ptr:	6-bit BCD pointer
Count:	number of 6-bit bytes by which Ptr is modified

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

72_8 or 58_{10} words

J. Source:

INC6 FAP

K. Files Referenced

None

3.4.4.6 PUT6

Purpose

To insert 6-bit character.

Description

A. Operation: PUT6 (Char, Ptr)

PUT6 puts the right-m[illegible] 6 bits of Char in the byte position indicated by Ptr.

B. Procedures Calling PUT6:

CTSIT6, PREP, TRASH

C. Procedures Called By PUT6

None

D. COMMON References:

None

E. Arguments:

Char:	6-bit data element, right adjusted
Ptr:	pointer to BCD string

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

29_{10} or 35_8 words

J. Source:

PUT6 FAP

K. Files Referenced:

None

3.4.4.7 GET12

Purpose

To get 12-bit byte.

Description

A. Operation: Char = GET12(Ptr)

GET12 extracts the 12-bit BCD character pointed to by Ptr and returns it as the value of Char.

B. Procedures Calling GET12:

GETLIN

C. Procedures Called By GET12:

None

D. COMMON References:

None

E. Arguments:

Ptr: pointer to 12-bit character string

F. Values:

Char = 12-bit right adjusted character

G. Error Codes:

None

H. Messages:

None

I. Length:

50_8 or 40_{10} words

J. Source:

STRIX2 ALGOL

K. Files Referenced:

None

3.4.4.8 INC12

Purpose

To adjust 12-bit BCD pointer.

Description

A. Operation: INC12(Ptr, Shift)

INC12 adjusts the pointer Ptr by the positive or negative quantity specified by Shift. The decrement of Ptr is left unchanged.

B. Procedures Calling INC12:

GETLIN

C. Procedures Called By INC12:

None

D. COMMON References:

None

E. Arguments:

Ptr: 12-bit BCD pointer ($0 \leq$ tag ≤ 2)
Shift: positive or negative integer

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

175_8 or 125_{10} words

J. Source:

STRIX2 ALGOL

K. Files Referenced:

None

3.4.4.9 COPY

Purpose

To copy ASCII string.

Description

A. Operation: COPY(Ascptr1, Ascptr2)

COPY takes the ASCII string defined by ASCPTR1 and copies it into the area which begins at the locations pointed to by Ascptr2. It is not necessary to specify a length in Ascptr2 (in other words, you could overflow your target area).

B. Procedures Calling COPY:

A. Called by: AUTHOR, EVAL, GETEND, GETFLD, IN., NEXITM, SEEMAT, STEM, S.T., TABLE, TRANS, TYPEIT

C. Procedures Called By COPY:

None

D. COMMON References:

None

E. Arguments:

Ascptr1:	ASCII pointer to string to be copied
Ascptr2:	ASCII pointer to area which will receive copy

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

114_{10} or 162_8

J. Source:

COPY FAP

K. Files Referenced:

None

3.4.4.10 DIST

Purpose

To compute length of string in bytes.

Description

A. Operation: Count = DIST(Ascptrl, Ascptr2)

DIST counts the number of characters in an ASCII string and returns this value to Count. The string is defined by Ascptrl, which points to the first character, and Ascptr2, which points to the last character. The count includes the first and last characters.

B. Procedures Calling DIST:

EVAL, FSO, SPCTRN

C. Procedures Called By DIST:

None

D. COMMON References:

None

E. Arguments:

Ascptrl, Ascptr2: ASCII pointers

F. Values:

Count = inclusive count of bytes between the two pointers

G. Error Codes:

None

H. Messages:

None

I. Length:

30_{10} or 36_8 words

J. Source:

DIST FAP

K. Files Referenced:

None

3.4.4.11 FIND

Purpose

To scan for delimiter

Description

A. Operation: Ptr = FIND(Delim, Strptr)

FIND looks for delimiters pointed to by the ASCII pointer Delim by scanning the ASCII string defined by Strptr. If FIND does not find any of the specified delimiters in the string, it returns a zero. Otherwise FIND returns a pointer to the first delimiter that it encounters. The decrement of this pointer contains the length of the remainder of the string from this point.

B. Procedures Calling FIND:
NEXITM, SPCTRN

C. Procedures Called By FIND:
GET, INC1

D. COMMON References:
None

E. Arguments:

Delim: ASCII pointer to a string of ASCII characters functioning as delimiters.

Strptr: ASCII pointer to a string to be searched for delimiters.

F. Values:

Ptr = pointer to substring of searched string, beginning with first occurance of a delimiter

G. Error Codes:
None

H. Messages:
None

I. Length:
108_8 or 72_{10} words

J. Source:
MATCH ALGOL

K. Files Referenced:
None

3.4.4.12 COMPAR

Purpose

To compare ASCII strings.

Description

A. Operation: COMPAR(Ascptr1, Ascptr2, Chars, Hi, Low, Equal)

COMPAR will compare the string pointed to by Ascptr1, with the string pointed to by Ascptr2. If, after comparing the number of characters specified in Chars, no inequality has been discovered, COMPAR will return by transferring to the label Equal. Otherwise, COMPAR will transfer to Hi or Low, depending on whether the first non-matching character in the string pointed to by Ascptr1 is high or low relative to the corresponding character in the other string.

B. Procedures Calling COMPAR:

SPCTRN, TABLK

C. Procedures Called By COMPAR:

GET, INC1

D. COMMON References:

None

E. Arguments:

Ascptr1:	pointer to the comparison ASCII string. This string is high, low or equal relative to the string pointed to by Ascptr2.
Ascptr2:	pointer to ASCII string to which first string is compared.
Chars:	number of characters from each string to be compared.
Hi:	If the first character in Ascptr1 which is not equal to Ascptr2 is greater, then this exit is taken.
Low:	Exit if first string is low.
Equal:	First Chars characters of both strings are equal if this exit is taken.

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

57_8 or 47_{10} words

J. Source:

MATCH ALGOL

K. Files Referenced:

None

3.4.4.13 COMPUL

Purpose

To compare strings, ignoring case differences.

Description

A. Operation: COMPUL (Ascptr1, Ascptr2, Chars, Hi, Low, Equal)

COMPUL compares two ASCII strings in exactly the same way that COMPAR does, except that it ignores case differences.

B. Procedures Calling COMPUL:
IFSRCH, MONTOR, OUT., SPCTRN, RANGE, TABLK, TIME

C. Procedures Called By COMPUL:
GET, INC1

D. COMMON References:
None

E. Arguments:
See description of COMPAR (Section 3.4.4.12)

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
61_8 or 49_{10} words

J. Source:
MATCH ALGOL

K. Files Referenced:
None

3.4.4.14 MATCH

Purpose

To find substring.

Description

A. Operation: Ptr = MATCH(Short, Long, Mode)

MATCH looks for the ASCII string Short within the ASCII string Long. If it finds a match, it will return a pointer to the first matching character. The decrement of this pointer ~~will~~ indicate how many characters the substring is from the beginning of the string pointed to by Long. If there is no match, or if short is longer than long, a value of 0 is returned. If the Mode is zero case differences will be considered, but not if the Mode is 1.

B. Procedures Calling MATCH:

FSO

C. Procedures Called By MATCH:

COMPARE, COMPUL, INC

D. COMMON References:

None

E. Arguments:

Short:	ASCII pointer to the substring being searched for
Long:	ASCII pointer to the string being searched.
Mode:	If 0, consider case differences, but if 1, ignore cases.

F. Values:

Ptr = ASCII pointer to substring within string Long.
If the decrement of Ptr is n, the substring begins with the nth character of Long.

G. Error Codes:

Code = 0: substring not found

H. Messages:

None

I. Length:

156_8 or 110_{10} words

J. Source:

MATCH ALGOL

K. Files Referenced:

None

3.4.4.15 GETSET

Purpose

To set up pointer for GETINC and SETINC.

Description

The GETINC package provides a means of sequentially processing a string of ASCII characters with slightly greater economy than by using the routines GET, INC and PUT.

A. Operation: GETSET (Ascptr, Newptr)

GETSET takes the ASCII pointer Ascptr and creates a pointer, Newptr, which can be used by GETINC and PUTINC.

B. Procedures Calling GETSET:

GETFLD

C. Procedures Called By GETSET:

None

D. COMMON References:

None

E. Arguments:

Ascptr:	ASCII pointer
Newptr:	address portion = location of beginning of string
Tag Portion:	= 0
Decrement Portion:	= expression of byte position in terms of number of bits from right end of word.

```
byte 0: = 27
byte 1: = 18
byte 2: =  9
byte 3: =  0
```

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

16_{10} or 20_8 words

J. Source:

GETINC FAP

K. Files Referenced:

None

3.4.4.16 GETINC

Purpose

To get character and increment pointer.

Description

A. Operation: Char = GETINC (Newptr)

GETINC extracts the character (9-bit ASCII) pointed to by the special pointer Newptr and returns this character as its value. At the same time, Newptr is incremented by 1 byte.

B. Procedures Calling GETINC:

GETFLD

C. Procedures Called By GETINC:

None

D. COMMON References:

None

E. Arguments:

Newptr: special pointer (see description of GETSET)

F. Values:

Char = ASCII (9-bit) character

G. Error Codes:

None

H. Messages:

None

I. Length:

22_8 or 18_{10} words

J. Source:

GETINC FAP

K. Files Referenced:

None

3.4.4.17 PUTINC

Purpose

To insert character and increment pointer.

Description

A. Operation: PUTINC(Char, Newptr)

PUTINC stores the 9-bit ASCII character on the right-hand side of Char in the byte pointed to by Newptr.

B. Procedures Calling PUTINC:

GETFLD

C. Procedures Called By PUTINC:

None

D. COMMON References:

None

E. Arguments:

Char:	9-bit ASCII character
Newptr:	special pointer (see description of GETSET)

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

34_8 or 28_{10} words

J. Source

PUTINC FAP

K. Files Referenced:

None

3.4.4.18 INC1

Purpose

To increment ASCII pointer one byte.

Description

A. Operation: INC1(Ascptr)

Ascptr is incremented by one byte.

B. Procedures Calling INC1:

CAPASC, CHKNUM, COMPAR, COMPUL, DROP, FIND, FSO, GETFLD, IN., NAME, NEXITM, OUT., SEEMAT, TABLE, TRANS, TYPEIT, .C.ASC

C. Procedures Called By INC1:

None

D. COMMON References:

None

E. Arguments:

Ascptr: ASCII pointer

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

14_{10} or 16_8 word

J. Source:

INC1 FAP

K. Files Referenced:

None

3.4.4.19 DEC1

Purpose

To decrement ASCII pointer by one byte.

Description

A. Operation: DEC1(Ascptr)

Ascptr is backed up one byte.

B. Procedures Calling DEC1:

NEXITM, STEM, TRANS, TYPEIT

C. Procedures Called By DEC1:

None

D. COMMON References:

None

E. Arguments:

Ascptr: ASCII pointer

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

18_{10} or 22_8 words

J. Source:

DEC1 FAP

K. Files Referenced:

None

3.4.5 Miscallaneous Utilities

3.4.5.1 DNSORT

Purpose

To sort in descending order.

Description

DNSORT is a special-purpose version of the general-purpose exchange sort XSORT. DNSORT sorts a list of words in place, using the right-most 15 bits as a key.

A. Operation: Ptr = DNSORT(Ptr)

DNSORT is called by NUMBER to sort in descending order a list of words pointed to by Ptr. These words contain document numbers that users have typed in the address portion of each word. Using these bits as a key, the sort proceeds by finding the highest document number for the first word of the array. Then, it finds the second-highest number for the next position, and so on. If the pointer Ptr has a zero decrement, DNSORT will return a -1. Otherwise, DNSORT returns Ptr as its value.

B. Procedures Calling DNSORT:
NUMBER (via CALLIT)

C. Procedures Called By DNSORT:
None

D. COMMON References:
None

E. Arguments:
Ptr: word pointer to array

F. Values:
Ptr = word pointer to sorted list (identical to Ptr used as argument)

G. Error Codes:
Ptr = -1: zero length list

H. Messages:
None

I. Length:

62_8 or 50_{10} words

J. Source:

DNSORT FAP

K. Files Referenced:

None

3.4.5.2 FAPDBG

Purpose

To debug system.

Description

A. Operation: FAPDBG( )

FAPDBG is an extensive debugging subroutine. It enables a user to examine core locations and to change their values. In addition, it enables a user to set a breakpoint and start execution of his procedure at any point that he chooses. Finally, if the user specifies a symbol table. FAPDBG will reference core locations in terms of their symbolic addresses rather than their absolute core locations.

FAPDBG is evoked by typing the Intrex command "fapdbg" or "dbg". Because the Intrex system is currently fairly stable, FAPDBG is not part of the present system. If extensive changes are made to the system in the future, it will probably be practical to re-include this large (5384 word) package.

B. Procedures Calling FAPDBG:
CLP (via CALLIT)

C. Procedures Called By FAPDBG:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:

$12{,}400_8$ or 5384_{10} words

J. Source:

TSLIB2 BSS

K. Files Referenced:

Name SYMTB

3.4.5.3 NAP

Purpose

To call SLEEP

Description

A. Operation: NAP(Time)

NAP calls SLEEP (Section 3.5.7.2) with the argument (number of seconds to sleep) set up in the accumulator.

B. Procedures Calling NAP:

GETLIN

C. Procedures Called By NAP:

SLEEP

D. COMMON References:

None

E. Arguments:

Time: umber of seconds that program will be asleep

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

13_{10} or 15_8 words

J. Source:

UTILIB

K. Files Referenced:

None

3.4.5.4 TBSRCH

Purpose

To search a table for a key

Description

A. Operation: Ptr = TBSRCH(Key, Table, Length)

The procedure TBSRCH is used to search a table that contains one computer word per entry. If the "Key" is found in "Table", then "Ptr" will point to the word which matched exactly on "Key". If no match is found, "Ptr" will be set to zero.

The last entry in the table must be a fence of binary ones.

B. Procedures Calling TBSRCH:

NEXITM

C. Procedures Called By TBSRCH:

None

D. COMMON References:

None

E. Arguments:

Key:	item (1 word) being searched for.
Table:	beginning location of table.
Length:	length of table in words.

F. Values:

Ptr = pointer to word in table which matched

G. Error Codes:

Ptr = 0: Key not found in table

H. Messages:

None

I. Length:

36_8 or 30_{10} words

J. Source:

SRCH FAP

K. Files Referenced:

None

3.4.5.5 VSRCH

Purpose

To search table with multi-word entries.

Description

A. Operation: Ptr = VSRCH(Key, Table, Length, N)

The procedure VSRCH is used to scan a table pointed to by Table for a word with the value of Key. Each entry in the table may be N words long. However, VSRCH only examines the first word of each entry in searching for the key.

If the first word of an entry matches Key, VSRCH will return a pointer to this word. If the Key is not found, VSRCH will return a value of zero.

B. Procedures Calling VSRCH:

LOOKUP, STEM

C. Procedures Called By VSRCH:

None

D. COMMON References:

None

E. Arguments:

Key:	item for which table is scanned
Table:	beginning location of table
Length:	number of entries in table
N:	number of computer words per entry

F. Values:

Ptr = pointer to word in table which matched with key.

G. Error Codes:

Ptr = 0: Key not found in table

H. Messages:

None

I. Length:

50_8 or 40_{10} words

J. Source:

SRCH FAP

K. Files Referenced: ✓

None

3.4.5.6 SHIFT

Purpose

To shift array left nine bits.

Description

A. Operation: SHIFT (Array)

The procedure SHIFT will shift the contents of the 3 word-array beginning at location Array nine bits to the left.

B. Procedures Calling SHIFT:

STEM

C. Procedures Called By SHIFT:

None

D. COMMON References:

None

E. Arguments:

Array: beginning location of 3-word array

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

13_{10} or 15_8 words

J. Source

SHIFT FAP

K. Files Referenced:

None

3.4.5.7 STRACC, PRSTRA

Purpose

STRACC:	To initialize tracing mechanism.
PRSTRA:	To write out results of trace.

Description

A. Operation: STRACC( ), PRSTRA( )

STRACC is called as part of the system generation process. It examines each word of core, looking for the TTR instructions which is used in the CTSS environment as a transfer vector. If it finds a TTR, and if it find a TSX instruction which points to the TTR and if the TTR points to a reasonable location, the TTR instruction is replaced by an instruction of the form TXH Trac, 0, Proc. The instruction TXH has the same effect as a simple transfer when the tag is zero. "TRAC" is the entry point of a trace routine and Proc is the transfer location of the TTR.

During execution of Intrex, transfers to subroutines will automatically result in transfers to the trace procedure TRAC. TRAC searches and internal table for the value Proc. If Proc is already in the table, the count is incremented by 1 in that slot. If Proc is not in the table, it is added.

At some point the procedure PRSTRA may be called. PRSTRA writes the table used by TRAC in a file called TRACY TABS.

The user may obtain a useful printout of the data in TRACY TABS by using the stand-alone program TROUT.

B. Procedures Calling STRACC, PRSTRA:

STRACC:	SEGINT
PRSTRA:	undetermined

C. Procedures Called By STRACC, PRSTRA:

STRACC:	GETMEM, TSPOT
PRSTRA:	CLOSE, OPEN, WRWAIT

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

STRACC:	250_{10} or 372_8 words
PRSTRA:	15_{10} or 17_8 words

J. Source:

STRACE ALGOL

K. Files Referenced:

STRACC:	None
PRSTRA:	TRACY TABS

3.4.5.8 TESTMO

To test for minus zero

Description

A. Operation: Bool = TESTMO(Val)

If the 36-bit word Val has a one bit in the left-most position and zeroes elsewhere, TESTMO returns a value of TRUE. Otherwise, its value is FALSE.

B. Procedures Calling TESTMO:

TYPEIT

C. Procedures Called By TESTMO:

None

D. COMMON References:

None

E. Arguments:

Val: value to be tested

F. Values:

Bool = TRUE if -0, FALSE otherwise

G. Error Codes:

None

H. Messages:

None

I. Length:

25_8 or 21_{10} words

J. Source:

TESTMO ALGOL

K. Files Referenced:

None

3.4.5.9 TSPOT, TRAC

Purpose

To TRACe subroutine calls.

Description

A. Operation: TRAC( ), Loc = TSPOT( )

TRAC is used to trace subroutine calls within the Intrex system. As a part of system generation STRACC (see Section 3.4.5.7) replaces all of the transfer vectors with an instruction of the form "TXH TRAC,O, PROC." When procedure Proc is called, the TXH instruction transfers control to the entry point TRAC. TRAC looks up entry point Proc in its table. If it finds it, TRAC updates the count field for that entry by one. Otherwise, TRAC creates a new entry in the table for Proc. TRAC then transfers to Proc.

TSPOT is used to communicate between the subroutines TRAC, STRACC, and PRSTRA. TSPOT provides STRACC with the entry point to TRAC, which STRACC uses in generating TXH instructions. The core location immediately above the entry point TRAC contains the location of the table used by TRAC. PRSTRA (see Section 3.4.5.2) uses this address in writing out the table.

B. Procedures Calling TRAC, TSPOT:

TRAC:	None
TSPOT:	STRACC

C. Procedures Called By TRAC, TSPOT:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

TRAC:	None
TSPOT:	Loc = entry point for TRAC

G. Error Codes:

None

H. Messages:

None

I. Length:

TRAC: 60_{10} or 74_8 words, TSPOT = 2 words

J. Source:

TRACE FAP

K. Files Referenced:

None

3.4.5.10 TIMEIN, TOUT, TOTTIM

Purpose

To provide fine timing of Intrex.

Description

A. Operation: TIMEIN( ), TOUT( ), TOTTIM( )

TIMEIN, TOUT and TOTTIM work together to provide a means of timing Intrex modules. Their advantage over MONTIM is that, being more primitive, they have considerably less overhead and can, therefore, be used to measure finer time intervals.

TIMEIN extracts the value of the B-core timer and places it in a save location. TOTTIM extracts the current time and subtracts from it the value obtained by TIMEIN. This value is added to a total. If INCTIM(SST.) is on, the incremental time is printed via WRFLXA. TOTTIM prints the total accumulated time in the same format.

B. Procedures Calling TIMEIN, TOUT, TOTTIM:

Dependent on the procedure being tested

C. Procedures Called by TIMEIN, TOUT, TOTTIM:

TIMEIN:	None
TOUT:	WRFLXA
TOTTIM:	WRFLXA

D. COMMON References:

Name	Meaning	Interrogated?	Changed?
INCTIM(SST.)	Increm. Time Mode	x	

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:
   1. " + 00108+" **

I. Lengths:

TIMEIN:	4 words
TOUT:	44 words
TOTTIM:	6 words

J. Source:

TIMER FAP

K. Files Referenced:

None

**This means 1 second and 8 60th's.

3.4.5.11 WHEN

Purpose

To get time and date

Description

A. Operation: Time = WHEN (Date)

The CTSS utility GETIME will deposit the time of day in the AC register and the date in the MQ. Since the MQ is not accessible through AED, the Fap-coded subroutine WHEN provides a means of obtaining this data. WHEN calls GETIME and stores the MQ in the argument Date.

B. Procedures Calling WHEN:

MONTOR

C. Procedures Called By WHEN:

GETIME

D. COMMON References:

None

E. Arguments:

Date: BCD-coded date in format of MMDDYY (value returned by WHEN).

F. Values:

Time = time of day in 60th's of seconds

G. Error Codes:

None

H. Messages:

None

I. Length:

15_8 or 13_{10} words

J. Source:

TIME ALGOL

K. Files Referenced:

None

3.5 CTSS Utilities

3.5.1 Disk I/O

3.5.1.1 OPEN

Purpose

To prepare the system for reading or writing a file.

Description

A. Operation: OPEN (Status, Name1, Name2, Mode, Device)

OPEN must be called before a file may be read or modified.

B. Procedures Calling OPEN:

BFOPEN, CHKSAV, CONDIR, CONNAM, DYNAMO, GETINT, GETLIS, IFSINT, INIRES, INITYP, LISFIL, LISTSL, LONG, MOVEIT, NAME, OPFILE, SAVE, SHORT, SUMOUT, SYSGEN, TRETRI, USE.

C. Procedures Called By OPEN:

None

D. COMMON References:

None

E. Arguments:

a. Status:

R:	Read
W:	Write
RW:	Read-Write

b. Name1:

first name of file

c. Name2:

second name of file

d. Mode:

0:	permanent
1:	temporary
4:	read-only
100:	protected

e. Device:

2: Disk

F. Values:

None

G. Error Codes:

3:	file is already in active status
4:	more than ten active files
5:	status is illegal
7:	linking depth exceeded
8:	file in private mode under different user
9:	attempt to write a read-only file
10:	attempt to read a write-only file
11:	machine or system error
12:	file not found in UFD
13:	illegal device specified
14:	no space allotted for this device
15:	space exhausted for this device
16:	file currently being restored from tape
17:	I/O error
18:	illegal use of M.F.D.
19:	U.F.D. not found (i.e., OPEN through a link)
20:	Attempt to read secondary mode file

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

SYSNAM SGMT0n ($1 \leq n \leq 4$)
(MOVIE TABLE)
SYSNAM .TBLE.
COMMAND TABLE*
FIELDS TABLE*
ENDING TEST1*
SMFILE DISTXT
SMFILE DIRTAB
LMFILE DISTXT
LMFILE DIRTAB
GUIDEA DISTXT
GUIDEA DIRTAB
MON0nn FILE* ($01 \leq nn \leq 10$)
TIMING SUMARY
NEW SUMARY*
FICHE DIRECT
CATDIR INTREX
CRnnn INTREX ($001 \leq nnn \leq 290$)

IFDA	date	
IFDS	date	
IFTABA	date	
IFTABS	date	
SInnn	date	$(001 \leq nnn \leq 260)$
AInnn	FILE	$(001 \leq nnn \leq 030)$
TEMnnn	FILE	$(001 \leq nnn \leq 010)$
NAMnnn	FILE	$(001 \leq nnn \leq 010)$
PASnnn	FILE	$(001 \leq nnn \leq 010)$
DUMnnn	FILE	$(001 \leq nnn \leq 010)$
Name	SAVE	
SAVED	DIRECT	

*OPEN,BUFFER, RDFILE, WRFILE, CLOSE called by BFOPEN, BFREAD, BFWRIT and BFCLOS respectively (see Section 3.5.2.3).

3.5.1.2 CLOSE

Purpose

To return an active file to inactive status

Description

A. Operation: CLOSE (Name 1, Name2)

Since the system will maintain only ten active files at once, files generally must be closed after being used.

B. Procedures Calling CLOSE:

ANDER, CLEANP, CHKSAV, CONDIR, CONNAM, DYNAMO, FSO, GETINT, IFSINT, IFSRCH, INIRES, INIVAR, LISFIL, LISTSL, LONG, NAME, QUIT, SAVE, SEARCH, SHORT, SUMOUT, SYSGEN, TIME, USE

C. Procedures Called By CLOSE:

None

D. COMMON References:

None

E. Arguments:

Name 1:	first name of file or "ALL"
Name 2:	second name of file

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced

See Section K of 3.5.1.1

3.5.1.3 BUFFER

Purpose

To assign an area of core which the I/O mechanism may use as a buffer.

Description

A. Operation: BUFFER(Name1, Name2, Buf(0) to 432)

BUFFER is called after the file has been initialized by OPEN. It must be used if the file is being written.

B. Procedures Calling BUFFER:

BFOPEN, CHKSAV, CONDIR, CONNAM, INIRES, LISFIL, LISTSL, LONG, MOVEIT, NAME, QUIT, SAVE, SHORT, SUMOUT, SYSGEN, USE

C. Procedures Called By BUFFER:

None

D. COMMON References:

None

E. Arguments:

a. Name1 and Name2: first and second names of the file.

b. Buf(0) to 432: Buf(0) is the beginning location of the core storage area. The expression "to 432" will cause the length of the buffer to be placed in the decrement of the argument of the call.

F. Values:

None

G. Error Codes:

3 - File is not an active file
4 - buffer is above memory bound
5 - buffer is too small

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

See Section K of 3.5.1.1

3.5.1.4 RDFILE

Purpose

To read a file and overlap with other processing

Description

A. Operation: RDFILE (Name1, Name2, Relloc, Area(x) to y, Eof, Eofcnt)

RDFILE initiates the I/O necessary to move y words of data from the file Name1, Name2, starting at word Relloc, to the area of memory starting at Area(x).

B. Procedures Calling RDFILE:

ANDER, BFREAD

C. Procedures Called By RDFILE:

None

D. COMMON References:

None

E. Arguments:

Name1, Name 2:	Name of file
Relloc:	1st word to be read
Area (x):	1st word of read area
y:	number of words to be read
Eof:	location of end-of-file routine
Eofcnt:	number of words read on last read before encountering EOF.

F. Values:

None

G. Error Codes:

3 - file is not active
4 - file is not in read status
5 - no buffer assigned to this file
6 - previous I/O out of bounds
7 - I/O error
8 - U.F.D. has been deleted

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

```
DUMxxx   FILE   (001 ≤ xxx ≤ 010)
NAMxxx   FILE   (001 ≤ xxx ≤ 010)
MONxxx   FILE   (001 ≤ xxx ≤ 010)
COMAND TABLE
FIELDS TABLE
ENDING TEST2
```

3.5.1.5 RDWAIT

Purpose

To read a file with no processing overlap

Description

A. Operation: RDWAIT(Name1, Name2, Relloc, Area(x) to y, Eof, Eofcnt)

RDWAIT works the same as RDFILE, except that it incorporates a call to FCHECK, a CTSS file checking routine which will cause RDWAIT to wait until the read has been completed before returning to the user.

B. Procedures Calling RDWAIT:

ANDER, CALLIT, CHKSAV, CONDIR, CONNAM, FSO, GETLIS, IFSINT, IFSRCH, INITYP, LISFIL, LISTSL, MOVEIT, NUMBER, SAVE, SUMOUT, SYSGEN, TRETRI, USE

C. Procedures Called By RDWAIT:

None

D. COMMON References:

None

E. Arguments:

See Section E of description of RDFILE (3.5.1.4)

F. Values:

None

G. Error Codes:

See Section G of description of RDFILE (3.5.1.4)

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

(MOVIE TABLE)		
sysnam	SGMT0n	$(1 \leq n \leq 4)$
sysnam	.TBLE.	
SMFILE	DIRTAB	
SMFILE	DISTXT	
LMFILE	DIRTAB	
LMFILE	DISTXT	
IFDA	date	
IFDS	date	
IFTABA	date	
IFTABS	date	
SIxxx	date	$(001 \leq xxx \leq 260)$
AIxxx	date	$(001 \leq xxx \leq 030)$
DUMxxx	FILE	$(001 \leq xxx \leq 010)$
NAMxxx	FILE	$(001 \leq xxx \leq 010)$
TEMxxx	FILE	$(001 \leq xxx \leq 010)$
SAVED	DIRECT	
name	SAVE	
NEW	SUMARY	
FICHE	DIRECT	

3.5.1.6 WRFILE

Purpose

To modify a file

Description

A. Operation: WRFILE(Name1, Name2, Relloc, Area(x) to y,Eof,Eofcnt)

WRFILE initiates the I/O necessary to move y words starting at Area(x) to the file Name1 Name2 starting at word Relloc.

B. Procedures Calling WRFILE:

BFWRIT

C. Procedures Called By WRFILE:

None

D. COMMON References:

None

E. Arguments:

See Section E of description of RDFILE (3.5.1.4)

F. Values:

None

G. Error Codes:

3 - file is not active
4 - file is not in write status
5 - no buffer assigned
6 - allotted space exhausted for this device
7 - previous I/O out of bounds
8 - I/O error
9 - non-zero relloc with write-only file
10 - maximum length exceeded

H. Messages:

None

I. Length:

2 (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

MON xxx FILE $(001 \leq xxx \leq 010)$

3.5.1.7 WRWAIT

Purpose

To modify a file

Description

A. Operation: WRWAIT(Name1, Name2, Relloc, Area(x) to y, Eof,Eofcnt)

WRWAIT functions the same as WRFILE, except that it waits until the I/O operation has been completed before returning to the user.

B. Procedures Calling WRWAIT:

ANDER, CHKSAV, CONDIR, CONNAM, FSO, IFSRCH, INIRES, MOVEIT, QUIT, SAVE, SUMOUT, SYSGEN

C. Procedures Called By WRWAIT:

None

D. COMMON References:

None

E. Arguments:

See Section E of RDFILE (3.5.1.4)

F. Values:

None

G. Error Codes:

See Section G of WRFILE (3.5.4.6)

H. Messages:

None

I. Length:

3 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

DUMnnn FILE ($001 \leq nnn \leq 010$)
PASnnn FILE ($001 \leq nnn \leq 010$)
NAMnnn FILE ($001 \leq nnn \leq 010$)
TEMnnn FILE ($001 \leq nnn \leq 010$)

TIMING SUMARY	(001 $\leq$ nnn $\leq$ 010)
SAVED DIRECT	(001 $\leq$ nnn $\leq$ 010)
xxxxxx SAVE	(xxxxxx is user-assigned name)
(Sysnam) SGMTnn	(01 $\leq$ nn $\leq$ 04)
(Sysnam) .TBLE.	

3.5.1.8 TRFILE

Purpose

To truncate a file

Description

A. Operation: TRFILE(Name1, Name2, Relloc)

The file Name1 Name2, which is open for writing, is truncated immediately before the relative location Relloc.

B. Procedures Calling TRFILE:

CLEANP, CONDIR, CONNAM, DROP, INIRES, NAME, QUIT

C. Procedures Called By TRFILE:

None

D. COMMON References:

None

E. Arguments:

a. Name1 Name2: name of opened file to be truncated

b. Relloc: location of first word to be truncated

F. Values:

None

G. Error Codes:

3 - file is not active
4 - file is not write status
5 - buffer has not been assigned to file
6 - previous I/O out of memory bound
7 - relloc larger than file length
8 - I/O error

H. Messages:

None

I. Length:

2 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

DUMnnn FILE $(001 \leq nnn \leq 010)$
TEMnnn FILE $(001 \leq nnn \leq 010)$
SAVED DIRECT

3.5.1.9 FWAIT

Purpose

To wait for I/O to be completed

Description

A. Operation: FWAIT (Name1, Name2)

If any I/O operation is in progress for file Name1 Name 2 when FWAIT is called, FWAIT will wait until the operation is completed before returning control to the user.

B. Procedures Calling FWAIT:

ANDER, BFREAD, BFWRIT

C. Procedures Called By FWAIT:

None

D. COMMON References:

None

E. Arguments:

Name1 Name2: first and second names of a file (BCD)

F. Values:

None

G. Error Codes:

3 - file is not active
4 - I/O is out of memory bound
5 - I/O error

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

SIxxx	(date)	$(001 < xxx < 260)$
AIxxx	(date)	$(001 \leq xxx \leq 30)$
DUMxxx	FILE	$(001 \leq xxx \leq 010)$
NAMxxx	FILE	$(001 \leq xxx \leq 010)$
MONxxx	FILE	$(001 \leq xxx \leq 010)$
COMAND	TABLE	
FIELDS	TABLE	
ENDING	TEST2	

3.5.2 Buffered Disk I/O

3.5.2.1 BFOPEN

Purpose

To open a file and assign buffers

Description

A. Operation: BFOPEN(Status, Name1, Name2, Buf1, Buf2, Buf3, Err)

BFOPEN opens the file Name1, Name2 for reading or writing and assigns up to 3 buffers to the file. BFOPEN will transfer to Err if an error occurs.

B. Procedures Calling BFOPEN:
INIDSK, PREP, REND, TABLE

C. Procedures Called By BFOPEN:
BUFFER, FWAIT, OPEN

D. COMMON References:
None

E. Arguments:

a. Status: may be R(Read) or W(Write)
b. Name1 Name2: name of file
c. Buf1, Buf2, Buf3: beginning locations of 432-word buffers. Reading requires at least one buffer and writing requires two.
d. Err: location of error handling routine.

F. Values:
None

G. Error Codes:
None

H. Messages
None

I. Length:
144_8 or 100_{10} words

J. Source:
CTSS

K. Files Referenced:

MONxxx FILE $(001 \leq xxx \leq 010)$
COMAND TABLE
FIELDS TABLE
ENDING TEST2

3.5.2.2 BFREAD

Purpose

To read a record

Description

A. Operation: BFREAD(Name1, Name2, Area(x) to y, Eof, Eofcnt, Err)

BFREAD moves y words from the current buffer to the location beginning of Area(x).

B. Procedures Calling BFREAD:

RDFILE, FCHECK

C. Procedures Called By BFREAD:

RDFILE, FCHECK

D. COMMON References:

None

E. Arguments:

a. Name1 Name2: file name

b. Area(x) to y: Area(x) is the first location of the area that will receive the data. "Y" words will be transmitted.

c. Eof: location to which control is transferred when end of file is encountered before end of file.

d. Eofcnt: number of words transmitted before the end of file is encountered.

e. Err: location of error routine.

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

3 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

COMAND TABLE
FIELDS TABLE
ENDING TEST2

3.5.2.3 BFWRIT

Purpose

To modify a file

Description

A. Operation: BFWRIT (Name1, Name2, Area(x) to y, Eof, Eofcnt, Err)

BFWRIT moves y words from the core location Area(x) to the current buffer.

B. Procedures Calling BFWRIT:
SUMOUT TYPEIT

C. Procedures Called By BFWRIT:
FWAIT, WRFILE

D. COMMON References:
None

E. Arguments:
See Section E of BFREAD (3.5.2.2)

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
246_8 or 166_{10} words

J. Source:
CTSS

K. Files Referenced:
MONxxx FILE ($000 \leq xxx \leq 010$)

3.5.2.4 BFCLOS

Purpose

To re-activate a file

Description

A. Operation: BFCLOS(Name1, Name2)

BFCLOS will clear the buffers used for Name1 Name2 and close the file.

B. Procedures Calling BFCLOS:
ERRGO, INIDSK, PREP, REND, TABLE

C. Procedures Called By BFCLOS:
CLOSE

D. COMMON References:
None

E. Arguments:
Name1,Name2: first and second names of file in BCD

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
246_8 or 166_{10} words

J. Source:
CTSS

K. Files Referenced:

```
MON xxx FILE      (001 ≤ xxx ≤ 010)
COMAND TABLE
ENDING TEST2
FIELDS TABLE
```

3.5.3 File Status

3.5.3.1 CHFILE

Purpose

To change the name and/or mode of a file

Description

A. Operation: CHFILE(Name1, Name2, Mode, Name3, Name4)

CHFILE will change the mode of Name1 Name2 to Mode and will change its name to Name3 Name4. To change the mode only, Name3 and Name4 must be represented by -0. The file being modified may not be in active status at the time of the change.

B. Procedures Calling CHFILE:

SYSGEN

C. Procedures Called By CHFILE:

None

D. COMMON References:

None

E. Arguments:

Name1, Name2: name of file to be changed
Mode: new mode of file (binary)
Name3, Name4: new name of file (or -0)

F. Values:

None

G. Error Codes:

3 - attempt to change M.F.D. or U.F.D. file
4 - file not found
5 - "linked" file not found
6 - "linking depth exceeded
7 - attempt to change "private" file
8 - attempt to change "protected" file
9 - record quota overflow
10 - Name3 Name4 already exists
11 - machine or system error
12 - file in active status

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

(MOVIE TABLE)

3.5.3.2 DELFIL

Purpose

To delete a file

Description

A. Operation: DELFIL(Name1, Name2)

DELFIL will delete Name1 Name2 from the file directory

B. Procedures Calling DELFIL:

CONNAM, DROP, SUMOUT, SYSGEN, USE

C. Procedures Called By DELFIL:

None

D. COMMON References:

None

E. Arguments:

Name1 Name2: Name of file (BCD)

F. Values:

None

G. Error Codes:

3. - file not found in U.F.D.
4 - "linked" file not found
5 - linking depth exceeded
6 - file is protected, private, read-only, or write-only
7 - machine or system error
8. - file in active status

H. Messages:

None

I. Length:

3 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

NAM xxx FILE $(001 \leq xxx \leq 010)$
(name) SAVE
(sysnam) SGMT0n $(1 \leq n \leq 4)$
(sysnam) .TBLE.
NEW SUMARY

3.5.3.3 FSTATE

Purpose

To retrieve statistical data on a file

Description

A. Operation FSTATE(Name1, Name2, A(0) to 8)

Upon return, the array A will contain the following data about the file Name; Name2:

A(0):	length of file in words
A(1):	mode
A(2):	status (open/closed)
A(3):	device (disk/drum/tape)
A(4):	address of next word to be read
A(5):	address of next word to be written
A(6):	date and time of last modification
A(7):	date of last read, author of file

B. Procedures Calling FSTATE:

FILCNT, GETLIS, INIDSK, INITYP, TYPEIT, SEARCH, SYSGEN, USE

C. Procedures Called By FSTATE:

None

D. COMMON References:

None

E. Arguments:

Name1, Name2: name of file
A(0) to 8: 8-word array, which FSTATE fills with data

F. Values:

None

G. Error Codes:

3 - file not found
4 - "linked" file not found
5 - linking depth exceeded

H. Messages:

None

I. Length:

2 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

COMAND	TABLE	
ENDING	TEST2	
FIELDS	TABLE	
GUIDEA	DIRTAB	
IFDA	date	
IFDS	date	
LMFILE	DIRTAB	
name	SAVE	
SAVED	DIRECT	
SMFILE	DIRTAB	
sysnam	.TBLE.	
AInnn	date	$(001 \leq nnn \leq 035)$
SInnn	date	$(001 \leq nnn \leq 270)$
DUMnnn	FILE	$(001 \leq nnn \leq 010)$
MONnnn	FILE	$(001 \leq nnn \leq 010)$
NAMnnn	FILE	$(001 \leq nnn \leq 010)$
PAS nnn	FILE	$(001 \leq nnn \leq 010)$

3.5.4 Disk I/O Errors

3.5.4.1 FERRTN

Purpose

To set error exit

Description

A. Operation: FERRTN(Errloc)

FERRTN establishes a single error return for all I/O system errors at location Errloc.

B. Procedures Calling FERRTN:

SETRTN, SYSGEN

C. Procedures Called By FERRTN:

None

D. COMMON References:

None

E. Arguments:

ERRLOC: core location of error routine

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

6 words

J. Source:

CTSS

K. Files Referenced:

All files

3.5.4.2 IODIAG

Purpose

To retrieve data on error conditions

Description

A. Operation: IODIAG(Area(0) to 7)

IODIAG may be called to obtain specific information about the I/O systems error. Upon return, the array Area will contain the following information:

A(0):	location of call causing the error
A(1):	BCD name of routine causing error
A(2):	error code
A(3):	I/O error code (1-7)
A(4):	Name1 of file
A(5):	Name2 of file
A(6):	empty

B. Procedures Calling IODIAG:

ERRGO

C. Procedures Called By IODIAG:

None

D. COMMON References:

None

E. Arguments:

Area(0) to 7: 7-word array, which will receive information

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

2 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

All files

3.5.5 Console I/O

3.5.5.1 RDFLXA

Purpose

To receive a line from the console

Description

A. Operation: Chars = RDFLXA (Area(x) to y)

RDFLXA reads a line from the console and moves y words into the core beginning at location Area (x). On return, "chars" will contain the number of 6-bit characters read, including the break character. In 12-bit mode, the number of characters read will be chars/2. The word containing the break character and subsequent words are padded with blanks. If the break character is not received before the input buffer is full, bit 21 of "chars" will be set to 1, indicating that another call to RDFLXA is necessary to continue reading the line.

B. Procedures Calling RDFLXA:

GETLIN, INIVAR

C. Procedures Called By RDFLXA:

None

D. COMMON References:

None

E. Arguments:

Area (x) to y:	defines an area beginning at Area(x) which is y words long.

F. Values:

Chars:	length of input line in 6-bit characters

G. Error Codes:

None

H. Messages:

None

I. Length:

2 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

None

3.5.5.2 RDFLX

Purpose

To receive a line from the console

Description

A. Operation: RDFLX (Area (x) to y)

RDFLX will read a line from the console using RDFLXA. It will then strip the break character from the line, pad any remaining characters up to y words with blanks, and move the y words into memory beginning at location Area (x). If y, which cannot be greater than 14, is less than the number of words read, the excess will be lost.

B. Procedures Calling RDFLX:

SYSGEN

C. Procedures Called by RDFLX:

RDFLXA

D. COMMON References:

None

E. Arguments:

Area(x) to y: defines a memory area starting at Area (x) and y words long.

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

75_8 or 61_{10} words

J. Source:

CTSS

K. Files Referenced:

None

3.5.5.3 WRFLXA

Purpose

To write data on console

Description

A. Operation: WRFLXA (Area (x) to y)

WRFLXA will print y words beginning at location Area (x), where y is less than 29 in 12-bit mode and y is less than 15 in 6-bit mode. It does not add a carriage return at the end of the line and does not delete trailing blanks.

B. Procedures Calling WRFLXA:

CALLIT, CLP, OUT., PRT12 SENTRY, SYSGEN, TRETRI
WFLXA

C. Procedures Called WRFLXA:

None

D. COMMON References:

None

E. Arguments:

Area (x) to y: Defines an area beginning at Area (x) which is y words long

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

3 words (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

None

3.5.5.4 WRFLX

Purpose

To write data on console.

Description

A. Operation: WRFLX (Area (x) to y)

WRFLX will print through the last non-blank character within y words beginning at location Area (x). Trailing blanks will be deleted and a carriage return inserted.

B. Procedures Calling WRFLX:
CALLIT, SYSGEN

C. Procedures Called by WRFLX:
None

D. COMMON References:
None

E. Arguments:
Area (x) to y: defines a memory area starting at Area (x) which is y words long.

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
1 word (resides in A-core)

J. Source:
CTSS

K. Files Referenced:
None

3.5.5.5 SETFUL

Purpose

To set character mode to 12-bit

Description

A. Operation: SETFUL( )

SETFUL sets the console character mode switch to send and receive in 12-bit mode.

B. Procedures Calling SETFUL:

GETLIN, PRT12, TRETRI

C. Procedures Calling SETFUL:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

CTSS

K. Files Referenced:

None

3.5.5.6 SETBCD

Purpose

To set character mode to 6-bit

Description

A. Operation: SETBCD( )

SETBCD sets the console character mode switch to send and receive a 6-bit mode.

B. Procedures Calling SETBCD:
GETLIN, PRT12, TRETRI

C. Procedures Called By SETBCD:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
1 word (resides in A-core)

J. Source:
CTSS

K. Files Referenced:
None

3.5.6 Data Conversion

3.5.6.1 BCDEC

Purpose

To perform BCD to binary conversion

Description

A. Operation Bin = BCDEC(BCD)

BCDEC converts the BCD number "BCD" to the binary equivalent Bin.

B. Procedures Calling BCDEC:

PREP, SYSGEN

C. Procedures Called By BCDEC:

None

D. COMMON References:

None

E. Arguments:

BCD: number in BCD representation

F. Values:

Bin = binary number

G. Error Codes:

None

H. Messages:

None

I. Length:

22_8 or 18_{10} words

J. Source:

SS

K. Files Referenced:

None

3.5.6.2 DEFBC

Purpose

To perform binary to BCD conversion

Description

A. Operation: A = DEFBC(K)

DEFBC converts the full 35 bits of K into a right justified, zero padded, BCD number. The sign bit is ignored.

B. Procedures Calling DEFBC:

GETFLD, IFSRCH, INIRES, SEARCH

C. Procedures Called By DEFBC:

None

D. COMMON References:

None

E. Arguments:

K: binary number

F. Values:

A = BCD number, right justified, zero padded.

G. Error Codes:

None

H. Messages:

None

I. Length:

3 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.6.3 DERBC

Purpose

To perform binary to BCD conversion

Description

A. Operation: A = DERBC(K)

DERBC converts the right 18 bits of K into a right justified, zero padded BCD number.

B. Procedures Calling DERBC:

TRETRI

C. Procedures Called By DERBC:

None

D. COMMON References:

None

E. Arguments:

K: binary integer

F. Values:

A = right justified, zero padded, BCD number

G. Error Codes:

None

H. Messages:

None

I. Length:

3 words

J. Source:

C 'SS

K. Files Referenced:

None

3.5.6.4 OCABC

Purpose

To convert from binary to octal

Description

A. Operation: A = OCABC(K)

OCABC converts the address field of K to 5 octal BCD digits with a leading blank.

B. Procedures Calling OCABC:

SYSGEN

C. Procedures Called By OCABC:

None

D. COMMON References:

None

E. Arguments:

K: binary digit

F. Values:

A = value of address field of K in octal, expressed in 5 BCD numbers and one leading blank.

G. Error Codes:

None

H. Messages:

None

I. Length:

4 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.6.5 RJUST

Purpose

To right-adjust BCD word

Description

A. Operation: A = RJUST(K)

RJUST will take the BCD word K and replace trailing blanks with leading blanks. If the word is all blanks "bbbbb" is returned.

B. Procedures Calling RJUST:

CHKNUM, QUIT, SYSGEN

C. Procedures Called By RJUST:

None

D. COMMON References:

None

E. Arguments:

K: BCD word

F. Values:

A = right adjusted BCD word

G. Error Codes:

None

H. Messages:

None

I. Length:

26_8 or 22_{10} words

J. Source:

RJUST

K. Files Referenced:

None

3.5.6.6 BZEL

Purpose

To convert zeroes to blanks

Description

A. Operation: A = BZEL(K)

BZEL will replace the leading zeroes in the BCD word K with blanks. If K is zero, "bbbbb*" will be returned.

B. Procedures Calling BZEL:

CHKNUM, QUIT

C. Procedures Called By BZEL:

None

D. COMMON References:

None

E. Arguments:

K: BCD word

F. Values:

A = BCD word with leading zeroes converted to blanks.

G. Error Codes:

None

H. Messages:

None

I. Length:

20_8 or 16_{10} words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7 Miscellaneous Utilities

Purpose

To return control to CTSS

Description

A. Operation: DORMNT( )

DORMNT returns control to the CTSS supervisor and puts the user in dormant status. Machine conditions, status and memory are saved. If the START or RSTART command is given, control returns to the machine instruction beyond the call to DORMNT.

B. Procedures Calling DORMNT:

ANDER, CLP, ERRGO, PREP, QUIT, SHORT, SUPER, TABLE, TYPASH, TYPEIT

C. Procedures Called By DORMNT:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

4 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.2 SLEEP

Purpose

To halt execution momentarily

Description

A. Operation: SLEEP( )

The program is placed in dormant status and is restored to working status after n seconds have elapsed, where n is the contents of the accumulator. Since the call to SLEEP is not convenient via AED, an AED-oriented procedure named NAP (Section 3.4.5.3) is employed to call it.

B. Procedures Calling SLEEP:
NAP

C. Procedures Called by SLEEP:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
2 words

J. Source:
CTSS

K. Files Referenced:
None

3.5.7.3 WAIT

Purpose

To enter WAIT status

Description

A. Operation: WAIT(S, N)

The INTREX system is put into WAIT status. The system will be restarted after n seconds have elapsed or when an input line is completed.

B. Procedures Calling WAIT:

GETLIN

C. Procedures Called By WAIT:

None

D. COMMON References:

None

E. Arguments:

S: 0: Timer - wait status: the program will be restarted after n seconds. No commands are accepted. Input lines are saved; the program is not restated when input lines arrive.

1: Input-wait status: the program will be restarted after n seconds have elapsed or when an input line is completed. If n is zero, the program will be restarted only when an input line is completed.

2: Dormant status: the program will be restarted after n seconds. An input line while dormant is interpreted as a command. This mode is equivalent to SLEEP.

N: number of seconds for which execution is to be suspended.

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

3 words

J. Source:

CTSS

K. Files Referenced

None

3.5.7.4 GETBRK

Purpose

To get location of interrupt.

Description

A. Operation: Loc = GETBRK( )

GETBRK is called to find the value of the instruction location counter at the point of interruption via the ATTN button.

B. Procedures Calling GETBRK:

INTONE, INTTWO, TYPEIT, LISTEN

C. Procedures Called By GETBRK:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

Loc = location of break

G. Error Codes:

None

H. Messages:

None

I. Length:

2 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.5 SETBRK

Purpose

To set the entry point for an interrupt handling routine.

Description

A. Operation: SETBRK(Loc)

When a program is started, it is at interrupt level 0. A program may drop the interrupt level and set the entry point for an interrupt handling routine for each level. During execution, the level may be raised either by a program call to the supervisor or by the user sending the interrupt signal. The interrupt signal causes the interrupt level to be raised by one and control to be transferred to the entry point previously specified by the program.

An interrupt at level 0 will be ignored. Every interrupt will cause the superviosr to print INT.n, where n is the level to which control is to be transferred.

B. Procedures Calling SETBRK:
INInt, INTONE, INTTWO

C. Procedures Called By SETBRK:
None

D. COMMON References:
None

E. Arguments:
Loc: location of interrupt routine

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:

2 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.6 SAVBRK

Purpose

To raise the interrupt level

Description

A. Operation: Loc = SAVBRK( )

SAVBRK raises the interrupt level by one and returns in the accumulator the entry point of the interrupt handling routine of the level just entered.

B. Procedures Calling SAVBRK:
TYPEIT

C. Procedures Called By SAVBRK:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
Loc = location of interrupt-handling routine for level ju [illegible] entered.

G. Error Codes:
None

H. Messages:
None

I. Length:
2 words

J. Source:
CTSS

K. Files Referenced:
None

3.5.7.7 GETMEM

Purpose

To obtain memory bound

Description

A. Operation: Mem = GETMEM( )

The current memory bound is stored in Mem.

B. Procedures Calling GETMEM:

FREE, FRET, SIZE

C. Procedures Called By GETMEM:

None

D. COMMON References:

None

E. Arguments

None

F. Values:

Mem = current memory size in binary

G. Error Codes:

None

H. Messages:

None

I. Length:

None

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.8 SETMEM

Purpose

To set memory bound

Description

A. Operation: N = M, SETMEM( )

SETMEM sets the memory allotment to the value of N. The prior expression "N = N" is necessary because SETMEM expects to find N in the accumulator.

B. Procedures Calling SETMEM:

FREE, SYSGEN

C. Procedures Called By SETMEM:

None

D. COMMON References:

None

E. Arguments:

N: Memory bound (binary)

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

2 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.9 WHOAMI

Purpose

To identify user

Description

A. Operation: WHOAMI(Area(0) to 7)

WHOAMI obtains status information from the supervisor. The array Area will contain the following:

Area (0):	Problem number
Area (1):	Programmer number
Area (2):	CTSS system name
Area (3):	Console ID code
Area (4):	Name of the login command
Area (5):	User's home file directory
Area (6):	User's name

B. Procedures Calling WHOAMI:

FSO, INXCON, MONTOR, WHOM

C. Procedures Called By WHOAMI:

None

D. COMMON References:

None

E. Arguments:

Area(0) to 7: 7-word array, which WHOAMI fills with data.

F. Values:

None

G. Error Codes:

None

H. Messages:
None

I. Length:
3 words

J. Source:
CTSS

K. Files Referenced:

None

3.5.7.10 SETWRD

Purpose

To set A-core word

Description

A. Operation: SETWRD (Word)

SETWRD stores the contents of "Word" in the A-core location that is reserved for the logged in user.

B. Procedures Calling SETWRD:

DYNAMO, INIDSK, LONG, OPFILE, QUIT, SHORT, TYPEIT

C. Procedures Called By SETWRD:

None

D. COMMON References:

None

E. Arguments:

Word: 1 word of data to be stored in A-core

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

3 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.11 GETWRD

Purpose

To get A-core word

Description

A. Operation: Word = GETWRD (Userno)

GETWRD will retrieve from A-core the contents of the location reserved for the logged-in user.

B. Procedures Calling GETWRD:

INITYP

C. Procedures Called By GETWRD:

None

D. COMMON References:

None

E. Arguments:

Userno: user number (in binary)

F. Values:

Word = contents of word in A-core reserved for user with user number Userno.

G. Error Codes:

None given in CTSS description

H. Messages:

None

I. Length:

1 word

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.12 SETBLP

Purpose

To set blip characters

Description

A. Operation: SETBLP(Chars,N)

SETBLP will cause the system to transmit to the console the three 12-bit characters stored in Chars every N seconds.

B. Procedures Calling SETBLP:

AND., SEARCH

C. Procedures Called By SETBLP:

None

D. COMMON References:

None

E. Arguments:

Chars:	word containing 0-3 12-bit characters
N:	number of seconds per blip (binary)

F. Values:

None

G. Arguments:

None

H. Messages

None

I. Length:

3 words

J. Source:

None

K. Files Referenced:

None

3.5.7.13 CHNCOM

Purpose

To chain commands

Description

A. Operation: CHNCOM(J)

CHNCOM determines if another command exists in the command buffer. If it exists, it is executed. If no new command is there, DORMNT is called if J is 1, DEAD if J is 0.

B. Procedures Calling CHNCOM:
IFSINT, QUIT, REND, SYSGEN

C. Procedures Called By CHNCO
None

D. COMMON References:
None

E. Arguments:
J: 0 or 1

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
6 words

J. Source:
CTSS

K. Files Referenced:
None

3.5.7.14 GETCOM

Purpose

To get command argument

Description

A. Operation: Arg = GETCOM(N)

GETCOM sets Arg to be the value of the Nth argument of the user's latest command. The command itself is number 0. The arguments may be numbered 1-19, including a fence at the end of all octal 7's. N must be a binary number in the location immediately after the call to GETCOM, not a TXH of an address containing N. Therefore, it is not suitable for the standard AED calling statement. (See COMARG in next Section.)

B. Procedures Calling GETCOM:
COMARG, SYSGEN

C. Procedures Called By GETCOM:
None

D. COMMON References:
None

E. Arguments:
N: sequential positional of argument desired. (Command itself is argument zero)

F. Values:
Arg = Nth argument of command line

G. Error Codes:
None

H. Messages:
None

I. Length:
3 words

J. Source:
CTSS

K. Files Referenced:
None

3.5.7.15 COMARG

Purpose

To get command argument

Description

A. Operation: Arg = COMARG(N)

COMARG is the AED procedure which provides access to selected arguments in the command line as explained in GETCOM in previous section.

B. Procedures Calling COMARG:

DYNAMO, INIVAR

C. Procedures Called By COMARG:

GETCOM

D. COMMON References:

None

E. Arguments:

N: sequential position of desired argument on command line (binary

F. Values:

Arg = Nth argument of command line

G. Error Codes:

-377777777777: no such argument N

H. Messages:

None

I. Length:

65_8 or 53_{10} words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.16 LDOPT

Purpose

To load option bits

Description

A. Operation: LDOPT(Bits)

The value of Bits will replace the current contents of bits 18-35 of the option word in A-core. The options are as follows:

1:	Search user UFD first for command
2:	Search user or system files (not both) for command
4:	Reset active files for command
10:	User subsystem trap enabled
20:	Inhibit quit signals for user
40:	Current user program is subsystem
100:	Automatic save before loading subsystem
200:	User is dialable (for attaching and slaving consoles)

B. Procedures Calling LDOPT:

DYNAMO, QUIT

C. Procedures Called By LDOPT:

None

D. COMMON References:

None

E. Arguments:

Bits: setting for option word

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

5 words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.17 SETSYS

Purpose

To set up subsystem

Description

A. Operation: SETSYS(Command, Mask)

SETSYS will establish either a CTSS command or a user's saved file as a subsystem (specified by command) which will take control when any of the conditions specified in Mask are met.

B. Procedures Calling SETSYS:

DYNAMO, QUIT

C. Procedures Called By SETSYS:

None

D. COMMON References:

None

E. Arguments:

Command:	name of subsystem to be used
Mask:	Condition bits:
	1. Trap new command
	2. Trap DEAD or DORMNT call
	3. Trap CHNCOM if no chain
	10. Trap on error condition

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

2 words (resides in A-core)

J. Source:

UTILIB BSS

K. Files Referenced:

None

3.5.7.18 GETSYS

Purpose

To get Subsystem Condition Code

Description

A. Operation: GETSYS(Word, Code)

Upon return from GETSYS, the two arguments will contain information pertaining to subsystem conditions. The first argument will contain the name (in 6-bit BCD) of the user's current subsystem. The second argument will contain the condition bits (set during Intrex initialization) which specify under what circumstances the subsystem is to be trapped (in rightmost 18 bits), and the corresponding bit which indicates which condition actually did cause a trap (in leftmost 18 bits).

B. Procedures Calling GETSYS:

INXSUB

C. Procedures Called By GETSYS:

None

D. COMMON References:

None

E. Arguments:

WORD:	.bcd. character string
CODE:	integer

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

1 word (resides in A-core)

J. Source:

NOLIB BSS

K. Files Referenced:

None

3.5.7.19 GNAM

Purpose

To identify calling program

Description

A. Operation: Code = GNAM( )

GNAM is used by a subroutine to determine the type of program which has called it. GNAM has been rewritten by the Intrex staff so as to return a value of zero, which indicates that the type is other than FAP, FORTRAN, or MAD.

B. Procedures Calling GNAM:
COMARG

C. Procedures Called By GNAM:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
Code = 0: procedure of type "unknown"

G. Error Codes:
None

H. Messages:
None

I. Length:
3 words

J. Source:
SYSNEW FAP

K. Files Referenced:
None

3.5.7.20 RSCLCK

Purpose

To reset CPU clock

Description

A. Operation: RSCLCK( )
RSCLCK sets the B-core timer (word 5) to zero.

B. Procedures Calling RSCLCK:
DYNAMO

C. Procedures Called By RSCLCK:
None

D. COMMON References:
None

E. Arguments:
None

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
5 words

J. Source:
CTSS

K. Files Referenced:
None

3.5.7.21 JOBTM

Purpose

To retrieve job time

Description

A. Operation: JOBTM(Tm)

JOBTM returns in the word Tm the elapsed CPU time since the first call to RSCLCK.

B. Procedures Calling JOBTM:

MONTIM

C. Procedures Called By JOBTM:

Nc e

D. COMMON References:

None

E. Arguments:

Tm: elapsed time in 60th's of a second

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

25_8 or 21_{10} words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.22 GETIME

Purpose

To get time of day

Description

A. Operation: TIME = GETIME( )

GETIME returns as its value the elapsed number of 60th's of seconds since midnight.

B. Procedures Calling GETIME:

GETLIN, GETTM, MONTIM, WHEN.

C. Procedures Called By GETIME:

None

D. COMMON References:

None

E. Arguments:

None

F. Values:

Time = time of day in 60th's of a second (binary)

G. Error Codes:

None

H. Messages:

None

I. Length:

4 words

J. Source:

CTSS

K. Files Referenced:

None.

3.5.7.23 GETTM

Purpose

To get time of day and date

Description

A. Operation: GETTM (Date, Time)

GETTM returns the date as a BCD value in its first argument and the time of day as a BCD value in its second argument. It obtains these values from GETIME.

B. Procedures Calling GETTM:

INIMON

C. Procedures Called By GETTM:

GETIME

D. COMMON References:

None

E. Arguments:

Date:	BCD value of the form MM/DDt
Time:	BCD value of the form HHMM.M

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

65_8 or 53_{10} words

J. Source:

CTSS

K. Files Referenced:

None

3.5.7.24 SCLS

Purpose

To set up a command list in a command buffer

Description

A. Operation: SCLS(TAB(n) BUF)

The array working backward from TAB(n) toward TAB(0) is transferred to the CTSS command buffer number BUF until a fence of all binary 1's is found. If BUF is a zero, the current command buffer will be filled. Since command buffers are twenty locations in length, Intrex's subsystem, INXSUB, uses TAB(19) as a SCLS argument to load the current buffer with arguments to a "Resume Intrex" command which were previously set up in the array TAB.

A call to NCOM (Section 3.5.7.25) will then cause the command and its arguments to be executed.

B. Procedures Calling SCLS:

INXSUB

C. Procedures Called By SCLS:

None

D. COMMON References:

None

E. Arguments:

TAB(n):	array location
BUF:	CTSS command buffer no.

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

20_8 or 16_{10} words

J. Source:

CTSS

K. File References:

None

3.5.7.25 NCOM

Purpose

To execute specified CTSS command from program

Description

A. Operation: NCOM(COM, PROG)

COM is a BCD integer containing the name of a CTSS command. PROG is a BCD integer containing the first argument to be used with COM. PROG replaces the existing first argument of the current command buffer before COM is executed.

In the Intrex subsystem INXSUB, the command in COM will be RESUME and the argument in PROG will be INTREX. The other necessary arguments will have been set up in the command buffer by the procedure SCLS (Section 3.5.7.24).

B. Procedures Calling NCOM:
INXSUB

C. Procedures Called by NCOM:
None

D. COMMON References:
None

E. Arguments:

COM:	BCD coded command
PROG:	BCD coded program name

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
4 words (resides in A-core)

J. Source:
CTSS

K. File References:
None

3.6 AED Utilities

3.6.1 Miscellaneous Utilities

3.6.1.1 WFLX

Purpose

To print a line with carriage return

Description

A. Operation: WFLX(.BCI./string/)

WFLX is a convenient means for printing a pre-set character string. WFLX will use WRFLX to print the string defined by the AED string expression defined by .BCI.

B. Procedures Calling WFLX:
CNTLOC, FREE, FRET, PUTOUT, TYPASH

C. Procedures Called By WFLX:
WRFLXA

D. COMMON References:
None

E. Arguments:
.BCI. pointer

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
2 words + the 36_8 or 30_{10} of WFLXA

J. Source:
AEDLB1 BSS

K. Files Referenced:
None

3.6.1.2 WFLXA

Purpose

To print a line

Description

A. Operation: WFLXA(.BCI./string/)

WFLXA is the same procedure as WFLX, except that it does not provide a final carriage return.

B. Procedures Calling WFLXA:

CNTLOC, FREE, FRET

C. Procedures Called By WFLXA:

WRFLXA

D. COMMON References:

None

E. Arguments:

None

F. Values:

None

G. Error Codes:

None

H. Messages:

None

I. Length:

36_8 or 30_{10} words

J. Source:

AEDLB1 BSS

K. Files Referenced:

None

3.6.1.3 ISARGV

Purpose

To extract an argument from a subroutine call

Description

A. Operation: Arg = ISARGV(Return, n, Mask)

ISARGV checks for the existance of an nth argument. If there is an nth argument, it is returned as the value of ISARGV. If there is not an nth argument, ISARGV returns Mask as its value.

B. Procedures Calling ISARGV:

NEXITM, TYPEIT, .C.ASC

C. Procedures Called By ISARGV:

None

D. COMMON References:

None

E. Arguments:

Return:	AED label of exit code for subroutine
n:	position number of requested argument
Mask:	any value

F. Values:

Arg = contents of argument n or value of Mask

G. Error Codes:

None

H. Messages:

None

I. Length:

60_8 or 48_{10} words

J. Source:

AEDLB1 BSS

K. Files Referenced:

None

3.6.1.4 ISARG

Purpose

To check for existance of an argument in a subroutine call

Description

A. Operation: Bool = ISARG (Return, n)

If n is positive, ISARG checks the nth position after the call to see if it is an argument. If n is negative, it checks the first through nth positions to make sure all are arguments. If all checked positions are arguments, the procedure exits with the value TRUE; otherwise, FALSE.

B. Procedures Calling ISARG:
NEXITM

C. Procedures Called By ISARG:
None

D. COMMON References:
None

E. Arguments:

Return:	AED label for exit code for subroutine calling ISARG (implicitly declared as type label)
n:	position number of argument

F. Values:
Bool = True if argument(s) exist(s); false otherwise.

G. Error Codes:
None

H. Messages:
None

I. Length:
60_8 or 48_{10} words (shares code with ISARGV)

J. Source:
AEDLB1 BSS

K. Files Referenced:
None

3.6.1.5 GETP

Purpose

To get ARDS parameters

Description

A. Operation: GETP(ID, N)

This procedure reads the current values of the WRFLX screen-size parameters for the ARDS console. A maximum of four parameters can be read into the array ID, the actual number being specified by N. They are stored in this order; line count, maximum lines per page, y-coordinate of the top of the page, number of characters per line. The procedure INXCON (Section 3.1.3.2) uses GETP to inspect the maximum lines per page which is set to 30 for the Intrex console and is greater than 30 for ordinary ARDS consoles.

B. Procedures Calling GETP:
INXCON

C. Procedures Called By GETP:
None

D. COMMON References:
None

E. Arguments:

ID:	4-word array
N:	number of words (1-4) to be read

F. Values:
None

G. Error Codes:
None

H. Messages:
None

I. Length:
17_8 or 21_{10} words

J. Source:
NOLIB or UTILIB

K. Files Referenced:
None

3.6.1.6 OCTTOI

Purpose

To perform binary to spread-octal conversion

Description

A. Operation: Bcdptr = OCTTOI (Val)

OCTTOI converts the binary number Val to a BCD-coded octal representation. Instead of returning the actual converted value, as does OCABC (see Section 3.5.6.4), it returns a pointer to the value, with a word count in the decrement.

B. Procedures Calling OCTTOI:

CNTLOC, FREE, FREZ

C. Procedures Called By OCTTOI:

None

D. COMMON References:

None

E. Arguments:

Val: binary number

F. Values:

Bcdptr = pointer to octal representation

G. Error Codes:

None

H. Messages:

None

I. Length:

26_8 or 22_{10} words

J. Source:

AEDLB1 BSS

K. Files Referenced:

None

IV. DATA BASE GENERATION

This chapter describes the procedural steps and programs used in the creation and updating of the Intrex data base. The format is basically similar to that of Chapter III. Part A provides a category of information not needed in Chapter III — "usage" of the program being described. Since we are here describing entire programs and not just subroutines, the proper activation of each program is of obvious importance. Part B describes the program's operation. In Part C, "Files Referenced", the names of files used as either input or output are listed. The conventions employed in Chapter III to designate fixed or variable names of files are used here as well. Part D contains a list of messages which may be produced by the program. These messages are numbered and references to them in Part B are made by the corresponding number appearing in parentheses. Message parts which are variable are again expressed symbolically and underlined.

Part E lists the source files which contain the major parts of the program. There are usually several utility procedures also required which are obtained from the library program files at load time.

Part F shows the CTSS commands needed to create and set aside an executable program file.

The process of updating the data base is a lengthy one and includes many steps. A complete list of these steps is given in Appendix H. In brief, the process is as follows:

Input files are generated on-line* by typists keying the catalog record data supplied by the Intrex catalog group. Computer printouts of the input are proofread and corrections are made via an on-line editing program. The program DRYRUN (Section 4.1) is run on a batch of edited files to detect further errors in format and another editing pass is made to remove the errors found by DRYRUN.

The program WETRUN (Section 4.1) is run on the corrected input files to produce new catalog file segments and Inverted File "shreds". The new catalog segments are condensed to digram-coded representation by MASH (Section 4.2). The subject/title shreds are phrase decomposed and stemmed

*During times when the computer is down, files are punched off-line on paper tape and later brought to the computer.

by the program STEMER (Section 4.3). These stemmed shreds and author shreds are sorted alphabetically on the stem or name by SORT (Section 4.4).

The sorted shreds are added to the current set of Inverted File segments by IFGENS (subject/title file) and IFGENA (author file) (Section 4.5). The new set of Inverted File segments is then checked for format and count errors by IFTEST (Section 4.6).

An ASCII printout of the Inverted Files may be obtained by generating a formatted ASCII-coded disk file via the program IFLIST (Section 4.7) and then requesting an off-line printout of that file.

4.1 DRYRUN, WETRUN

Purpose

To prepare input data for inclusion in data base

Description

The formatting program transforms disk-stored catalog records into files for the Intrex retrieval system by: (1) updating the Catalog Directory file (CATDIR FILE), (2) updating the Fiche Directory (FICHE DIRECT), (3) appending new records to the Catalog Record files (CRxxx M25100), and (4) creating two files of search terms: AUTHOR date, which contains the authors' namesand SUBTIT date which contains the subject and title terms.

```
A.  Usage:    r  DRYRUN
              r  WETRUN
```

The formatting program resides in Comfil 4 as two different versions, called DRYRUN SAVED and WETRUN SAVED. DRYRUN SAVED is loaded with a dummy version of WRWAIT and BFWRIT. DRYRUN will generate zero-length SUBTIT date and AUTHOR date files, but will not write any data on disk. It is used to detect errors in the input files. WETRUN contains active disk procedures WRWAIT and BFWRIT, and is used for the actual updating of the catalog.

Both programs require a file (PRE FORMAT) which contains the names of the input files. Each such file contains about 10 catalog records. The first name of these files has the format W..xxxx, where xxxx is the file number. The nature of these names is not checked by the program, but if a file whose name appears in PRE FORMAT is not found, an error message is printed.

The program gives on-line indications of its progress and of errors that it finds. The design of the program is aimed at (1) continuing processing as long as possible and (2) avoiding the admission of bad data into the catalog record file.

The operational steps involved in running DRYRUN and WETRUN are listed in Appnedix H (Steps 1-5).

B. Operation:

The catalog records which WETRUN receives as input are organized so as to be easily read and edited. The task of the WETRUN program is to re-format this data so that it may be efficiently handled by the computer. All of the fields (except field 5, the fiche location) are copied into a re-formatted catalog record which has three sections. The first section if four words long and contains data which has been converted to fixed-length, binary-encoded fields. The second section contains pointers to ASCII-encoded variable-length fields which comprise the third section of the record. The re-formatted catalog record is appended to the current segment of the catalog. If this segment exceeds a certain length threshold, then a new segment is created. A pointer to the record, specifying its segment number, word position and length is stored in word D + 10 of CATDIR FILE, where D is the document number. The data in field 5 (fiche location) is compressed into a one-word binary format and stored in word D of FICHE DIRECT. WETRUN generates special formatted records called shreds from fields 21, 24 and 73 (author, title and subject). These shreds will be used by other programs to generate new entries in the Inverted File. The author shreds, which contain one author per shred, are written in the file AUTHOR date. The subject shreds, which contain one subject term in each, and the title shreds, containing an entire title, are written into SUBTIT date.

The WETRUN program processes a catalog record in two basic phases. First, it reads in a record and constructs the re-formatted record in core. If no fatal errors* are encountered in this phase, it continues to the second phase: appending the record to the catalog segment, updating the Catalog Directory, and generating shreds. Otherwise, it goes on to process the next record

*See Part D of this section.

without writing any data on disk.

The main procedure FORFOR calls OPERUN to open the output files and the file PRE FORMAT, which contains the list of input files. OPERUN also performs general initialization functions. The main processing loop begins with the call to OPEINP by FORFOR. OPEINP extracts the next available name from the file PRE FORMAT and opens that file. FORFOR then calls REAREC, which is the basic control module for the first phase of processing. REAREC calls READON, which steps through the catalog record looking for fields. When a new field is found, the program transfers control to the appropriate subsection for that field. If the field is field 5 (fiche location), FGEN will be called to update the file FICHE DIRECT. If the field is to be converted to binary, GULP will be called to perform the conversion and STASH will be used to store it in the buffer used for the first section of the catalog record. If the field is to be left as an ASCII string, STUFF is called to copy it into the buffer for Section 3 and to generate a pointer to it. READON and STUFF both use the procedure REACHA (Leave, Labl3) to extract characters from the input file. REACHA returns via Leave when it encounters the end of a field and via Labl3 if there are no more characters in the input stream. Otherwise, REACHA returns normally.

If REAREC returns control to FORFOR without having encountered a fatal error, FORFOR calls CLOREC to carry out the second phase of processing. CLOREC calls AUTSHR, TITSHR and SUBSHR to generate author, title and subject shreds and to write them out and disk in the files AUTHOR Date and SUBTIT Date. CLOREC calls CATRAD to append the formatted record to the current catalog segment and to enter a pointer to it in CATDIR FILE. If the length of the segment exceeds a certain threshold, a new segment is created.

If an end of file has not been encountered in the current input file, REAREC is called to process the next record. Otherwise OPEINP is called to extract the next name from PRE FORMAT and to open the file. If OPEINP reports that the list of files is exhausted, CLORUN is called to close the run.

C. Files Referenced:

CATDIR FILE
CRxxx M25100
SUBTIT date
AUTHOR date
FICHE DIRECT

D. Messages:

AUTSHR

1. "Too many authors in record."
2. "Author shred overflow"***
3. "Bizarre error making author shreds-aborting—eof encountered"†
4. "Write error for author shreds"**

<u>CANERR</u>

1. "Possible error in shred type" (back space)

<u>CAT 1</u>

1. "Nth record lacks a record number N = <u>n</u>" *
2. "record already in Catdir File*

<u>CAT 2</u>

1. "Error in opening Catrec File" **

<u>CLOREC</u>

1. "The 1st N subject terms have been processed where N = <u>n</u>."
2. "Subject terms, first N author terms have been processed N = <u>n</u>."
3. "The subject and author terms have been processed."
4. "Error in writing or reading Catrec File or Catdir File."**

<u>REAREC</u>

1. "Field 5 missing"!
2. "Record exceeds max. length of 4095."*

<u>READON</u>

1. "Erroneous field number."*
2. "Two record numbers in same record.-.- may be missing."*

*** See Footnotes at the end of this section.

3. "Slot filled in fiche file"
4. "Bad fiche field"
5. "—.– missing – record skipped.*

FORFOR

1. "Record has not been processed."*
2. "An input file has been run. Time = t."
3. "File closing error in (format)."**
4. "No file created for this record."*

GULP

1. "bizarre error may have occurred. Please examine the last record in record file. Files may have been unprocessed after it. –input eof."*†

OPEINP

1. "Files in "pre format" have been run!'
2. "Error in opening input file."**

OPERUN

1. "File in "Pre Format" not found."**
2. "Name of Catrec File not found in Catdir File."**
3. "Catdir file end of file error (OPERUN). "**
4. "Error in opening input files."**

EGAD

1. "Error in closing out Catdir File."**

ROTATE

1. "Error in reading input file. "**

CLORUN

1. "An input file has been run"
2. "File closing error."**

OPESHR

1. "Field missing."

CHEWOR

1. "Bizarre program error-run aborted, but files saved."†
2. "Illegal backspace"

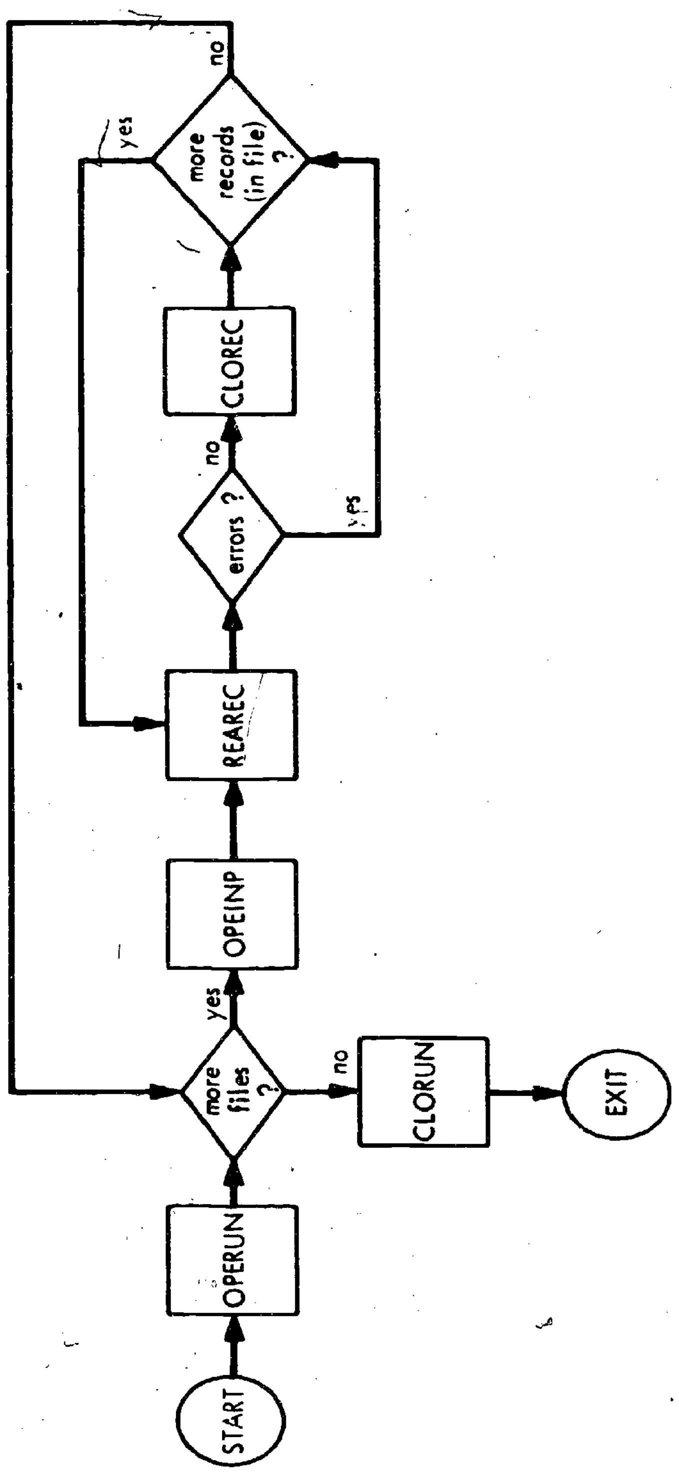


Fig. 4.1 DRYRUN/WETRUN (Main Flow of Control)

STUFF

1. "Too many fields."*

SUBSHR

1. "Too many subj. terms—Processing halted at term 63."
2. "Subject shred buffer overflow – rec aborted."***
3. "bizarre error in subj. shred making — rec aborted."†
4. "File writing error (subj. shred) – run aborted."*
5. "Difficulties in making shred number = n."***

TITSHR

1. "Bizarre error in TITSHR- record aborted."†
2. "Title shred buffer overflow — record aborted."***
3. "Prob. in writing title shred operations ended."**

ERRFIE

1. "Error in making shred."**

E. Source:

WETRUN and DRYRUN are constructed from about 20 separate AED source files. These files are listed in Fig. 4.2.

F. Loading Instructions:

Load files have been prepared for both WETRUN and DRYRUN (see Fig. 4.2).

```
LAED - ncload   - WETRUN (or DRYRUN)
SAVE  WETRUN  (or DRYRUN)
```

* No data will be written out for this record and processing will begin on the next catalog record.

** Run is aborted.

*** Processing of this catalog record will stop, but some subject, title, or author shred have been written out into the shred files. The record has not been appended to the catalog.

† This kind of error condition can only be generated by a program bug.

DRYRUN LOAD

- FORFOR
- CATRAD
- GULFOR
- VARFOR
- FGEN2
- DAN
- ASCINT
- NEXITM
- REAFOR
- OPERUN
- RUNFOR
- CLOREC
- STOFIV
- SHRFOR
- AUTFOR
- TITFOR
- SUBFOR
- SRCH
- MASAGE
- I/O
- (SRCH)
- UTILIB
- (SRCH)
- STRLIB
- (SRCH)
- (SQZ)
- NEWLB1

WETRUN LOAD

- FORFOR
- CATRAD
- GULFOR
- VARFOR
- DAN
- ASCINT
- NEXITM
- REAFOR
- OPERUN
- RUNFOR
- CLOREC
- STOFIV
- SHRFOR
- AUTFOR
- TITFOR
- SUBFOR
- SRCH
- MASAGE
- (SRCH)
- UTILIB
- (SRCH)
- STRLIB
- (SRCH)
- (SQZ)
- NEWLB1

Fig. 4.2 Loading Sequence

4.2 MASH

Purpose

To compress catalog files

Description

MASH is used to convert catalog records generated by WETRUN into a more compact format. The ASCII-encoded fields 2, 20, 37 and 46 are converted to fixed-length binary-encoded fields. All other ASCII fields are converted to digram-encoded ASCII. That is, two contiguous nine-bit ASCII characters are represented by one nine-bit digram encoded character whenever possible. Since a nine-bit byte allows 512 different codes and the ASCII character set uses only 128, a considerable number of combinations of characters can be encoded.

A. Usage:

```
R MASH M25100 INTREX first last
```

MASH uses the directory SORTED M25100 to convert a segment CRxxx M25100 to the digram-encoded segment CRxxx INTREX. The process of creating a directory for MASH involves the following steps:

1. The directory is generated out of the newly-updated file, CATDIR FILE.

   CATDIR FILE must be shortened by using the CTSS procedure SPLIT. The output of SPLIT is CAT FILE:

   ```
   SPLIT CATDIR FILE (WDCT) *864 CAT
   ```

2. The pointers in CAT FILE are ordered according to document number. They must be re-ordered according to the position of the records in the catalog files. Also, empty slots (for documents not yet included in the data base) must be removed. The following command produces a file SORT OUT which is ordered on segment and word numbers, with empty words removed:

   ```
   R SORTER CAT FILE 777000 0777777 0 000000 000000
   ```

3. SORT OUT is renamed SORTED M25100:

   ```
   RENAME SORTOUT SORTED M25100
   ```

If CATDIR INTREX, the catalog directory for the compressed catalog, is shorter than the newly updated CATDIR FILE, then it must be lengthened by:

a. Finding the length of CATDIR FILE and CATDIR INTREX by calling FSTATE:

```
CALL FSTATE CATDIR FILE
CALL FSTATE CATDIR INTREX:
```

b. Using EXTEND to lengthen CATDIR INTREX:

R EXTEND CATDIR INTREX Len.

A copy of CATDIR INTREX called CATDIR M25100 must be created.

MOVE CATDIR INTREX CATDIR M25100.

This file is not actually used for anything, but it is needed to accommodate a fluke in MASH. The last segment of CRxxx INTREX must be deleted.

DELETE CRxxx INTREX

This segment will be completely re-written by MASH

MASH is now resumed.

R MASH M25100 INTREX first last

"M25100" is the second name of the uncompressed catalog records and "INTREX" is the second name of the first segment to be processed and "last" is the number of the last segment.

B. Operation:

1. The directory SORTED M25100 is searched word-by-word for the first occurrence of the segment indicated by "first".
2. When a pointer to the specified segment is found, the segment is opened and the firstword of the segment is read.
3. The catalog record is read into memory.
4. A check is made for six octal 7's in word 5 of the record.
5. MASH steps through the lists of fields in the header. Fields 2, 20, 37 and 46 are converted to binary and placed in word 3 of the new record. All others are converted to digram encoded characters. The lengths of fields 67, 70, 71, and 73 are added to arrays.
6. The record is written out in CRxxx INTREX and control returns to step 3.
7. When MASH has finished with a segment, the next segment is opened and the process continues.
8. When segment "last" has been processed, MASH prints the contents of its statistical table.

C. Files Referenced:

```
SORTED M25100
CATDIR INTREX
CRxxx M25100        (000 < xxx < 295)
CRxxx INTREX        (000 < xxx < 295)
CATDIR M25100
```

D. Messages:

1. "No old CATDIR second name given"
2. "No new CATDIR second name given"
3. "Input file is CATDIR (name)"
   Output file will be CATDIR (name)"
4. "File CATDIR (name) does not exist"
5. "Length of CATDIR M25100 does not equal length of CATDIR INTREX"
6. "Output files will not have name (name)"
7. "File CRxxx not found"
8. "Duplicate entry for record number x"
9. "Fence not found in record x"
   File is CRxxx M25100
   y current catdir entry
   z last record written
   ****** File is not good *****"
10. "Field number x put in lower body, not upper.
    File is CRxxx INTREX Record number n
11. "Finished with file CRxxx INTREX
    Length is n"
12. "Finished Last file is CRxxx INTREX. Total document count x"
13. "FILE Error"

E. Source:

```
MASH ALGOL
STAT ALGOL
DTABLE ALGOL
```

F. Loading Procedure:

```
LAED -ncload- (GET) MASH
SAVE MASH
```

The load file MASH LOAD contains the following:

```
MASH
DTABLE
STAT
(SRCH)
IOLIB
```

4.3 STEMER SAVED

Purpose

To decompose and stem subject/title terms.

Description

The shreds produced by WETRUN (see Section 4.1) from the subject terms and title fields of the catalog records being processed contain the entire phrases used in those fields. STEMER reads these shreds, dissects the phrases into individual words, stems each word (encoding the endings removed), and writes new single-word shreds. These new shreds are used to update (enlarge) the Inverted File later in the generation process (see Section 4.5).

A. Usage: R STEMER name2

The argument name2 must be the second name of the shred file whose first name is SUBTIT. This second name is the date when WETRUN created the shred file and has the form MMDDYY in 6-bit BCD code. These names are assigned by WETRUN as it creates the shred file. STEMER creates a new file name STEMED name2.

B. Operation:

STEMER SAVED performs the following steps in its processing of the subject/title shred file.

1. Read the second name of the file from the command argument name 2.
2. Initialize the ending table via INIEND.
3. Open the input file for reading.
4. Open the output file for writing.
5. Read the first word of a shred from the shred file into core.
6. Take the length of the shred from the first word read in Step 5.
7. Read the rest of the shred as determined by the length obtained in Step 6.
8. Dissect and stem the phrase contained in the stem just read by calling the subroutine STEMER.
9. Write the new stemmed shreds created by the subroutine STEMER into the output file
10. Return the used shred array to free storage.
11. If more input shreds remain to be read, return to Step 5.
12. If all input shreds are processed, close the files and print ending count statistics (11 and 12) by calling SHOW.

Some of the more important procedures contained in STEMER SAVED are described below.

INIEND( ): This procedure, called in Step 2 above, reads a file of common English word endings named ENDING TEST2. These endings were derived from the work on stemming of J. Lovins.[8] The ending file having been read into core, a table of pointers is constructed by calling the procedure ENDTAB. These pointers point to the various groups (by length) of the endings in the file. The pointer table acts as a connection between an ending code (described below) and the corresponding ending in the file. The table is stored in the top of a large array (471 words), the balance of which is used for statistical data gathered during stemming of the shreds. The address of this pointer table is returned to the main program (STEMER) as a value of INIEND.

STEMER: To avoid confusion between STEMER, the main program resumed at command level, and the procedure STEMER, we shall refer to the procedure, hereafter, as STEMER and the entire module as STEMER SAVED. STEMER accepts an argument OSHRED containing a pointer to the original (phrase) shred and, through repeated calls to the sub-procedure NEXITM, extracts each word from the phrase. Pointers to these single words are passed to another procedure, STEM, which does the actual matching of the endings in the ending file against the end of the word in question. NEXITM and STEM are the same routines used by the Intrex System and are described in Sections 3.2.1.4 and 3.2.2.5 respectively.

STEM returns a pointer to the stemmed word to STEMER, which adds it to an array of stemmed shreds being created from the phrase shred in process. STEM also returns (in its second argument) a code representing the ending removed from the original word. This is a 12-bit code where bits 1-4 contain the number of characters in the ending and bits 5-12 contain the position of that ending within its length group. As each ending is removed, an associated counter is incremented to keep track of the number of times each ending is used.

Both the stems and the ending codes of the entire phrase are stored in their separate arrays by STEMER until all the words in the phrase are stemmed. Then they are extracted from these arrays one by one and a new shred is created from each by forming a field one containing the stem, a field two containing a copy of the reference in the original shred, and a field three containing the ending code. The copied reference is modified to contain the word number (within the phrase) of the stem whose shred is being created.

The logic exists within STEMER to create a new shred out of the entire stemmed phrase as well as the individual words and is controllable by the setting of a maximum phrase length parameter. However, this parameter has for some time been set to 1 for the creation of Intrex stemmed shreds so that only single word-stems appear in the Inverted Files as presently constituted.

The array which holds the collection of new shreds as they are created after stemming is obtained from free storage. After all the new shreds are created, a pointer to this array is returned to the main routine of STEMER SAVED for outputting. This array is returned to free storage by STEMER SAVED.

<u>SHOW ( ):</u> When all the shreds from the input file have been read and converted to individual, stemmed shreds, STEMER SAVED proceeds to display statistics on the number of times each common ending was removed during the processing. These counts are kept and incremented within an array set up by INIEND as an extension of the ending pointer table referred to earlier. Each location in the array corresponds to one ending in the ending list and is incremented whenever that ending is removed by STEM.

SHOW prints a header line containing the column headings, ENDING and TIMES USED, and then proceeds to print a list of all the endings in the list whose corresponding counts are greater than zero. The count will appear beside each ending. When all endings and counts have been listed, the total number of all stemmed words will be printed. SHOW then returns to STEMER SAVED.

C. Files Referenced:

```
SUBTIT  name2
STEMED  name2
ENDING  TEST2
```

D. Messages:

1. "No ending file found"
2. "bf routine error"
3. "Error in reading endings"
4. "Error in opening file"
5. "Error in reading file"
6. "Premature end-of-file"
7. "Word length 0 on word _n_ of name _lisnam_"
8. "Shred fence missing. Previous name's stems were — _stems_"
9. "Insufficient free storage called for new shreds. Stems are — _stems_"
10. "SUBTIT — name2 has been run"
11. "ending times used"
12. "total words stemmed is _s_"

E. Source Files:

STEMER ALGOL	(Main, STEM)
STEM1	(STEMER, SHOW)
STEM2	(INIEND)

F. Loading Procedure:

```
LAED   -ncload-* STEMER STEM1  STEM2 (SRCH) UTILIB(AEDP)
SAVE STEMER
```

*Optional argument which eliminates loader from core image.

4.4 SORT SAVED

Purpose

To sort stemmed shred on index words

Description

The program SORT was developed by Technical Information Program personnel to be used on files in a special format of their design. This format, which is called "TIP Searchable," has been adopted by Intrex to be used for sorting the shreds which are produced by WETRUN (Section 4.1) and which eventually serve as input to the Inverted File Generator IFGEN (Section 4.5). Thus SORT, together with STEMER described in 4.3; prepare the shreds for inclusion into the data base.

A. Usage R SORT STEMED date SORTS date 1

The arguments STEMED date will be taken by SORT to be the name of the input file to be sorted. These are the names assigned to the stemmed and decomposed shred file by STEMER. The arguments SORTS date will be assigned to the output file created by SORT. The last argument specifies that the SORT is to be keyed on field 1 (the search word) of the shreds.

Author shreds are also sorted, although they do not go through the stemming process. The WETRUN program produces an author shred file named AUTHOR date which is sorted by the command:

R SORT AUTHOR date SORTA date 1

B. Operation:

For a description of the sort method used by this program, see Reference 14.

C. Files Referenced:

STEMED date
SORTS date
or
AUTHOR date
SORTA date

D. Messages:

n items in n items out
(n is the number of items sorted)

E. Source Files:

Property of Technical Information Program, M. I. T.

F. Loading Procedure:

Loaded only by TIP

4.5 IFGEN SAVED

Purpose

To generate or update Inverted Files from stemmed, sorted shreds

Description

IFGEN SAVED exists in two versions named IFGENS and IFGENA which are used in the generation of subject/title and author Inverted Files respectively. The two programs are basically similar but contain enough differences to make combining them into a single dual-purpose program unwieldy. In describing them, however, it is reasonable to talk about both as though they were one and the same; and to merely point out the more important ways in which they differ.

Each program reads a sorted shred file, SORTS or SORTA, and merges the shreds into an existing set of Inverted File segments, creating new segments in the process. If no old Inverted File segments exist, fresh segments are created from the input shreds. These segments will have a first name format of SInnn or AInnn, respectively, with the number nnn ranging sequentially from 1 to as high as necessary. The second name of the segments will be the same as that of the input file — usually the date of creation in the form MMDDYY (month, day, year).

A. Usage: R IFGENS(A) old date

The argument old is the second name of the current set of Inverted Files to be updated. If none yet exist, this argument must be (0). It may not be omitted. The argument date is the second name of the sorted shred file which is to be read to create the new set of Inverted Files. This name will be used by IFGEN as the second name of all new Inverted File segments.

B. Operation:

Since IFGEN is too large to be compiled all in one source file, it has been split into two segments. The first segment has either of two names as mentioned above, IFGENS ALGOL for processing subject/title files and IFGENA ALGOL for processing author files. The second segment, IFGENB ALGOL, is used in conjunction with both versions of the first segment.

IFGENS(A) contains the main program, a GETLIS routine to read the next list from the input Inverted File, a GETSHR routine to read and collect references and affixes for the next group of shreds which have the same stem field, and a REFMER routine which merges shred references (after they have been sorted) with Inverted File list references.

IFGENB contains a PUTLIS routine to write the output lists into the new Inverted File segments, an ADDTAB routine to enter the appropriate information into the directory files, an AFFMER routine to merge shred affixes with old Inverted File affixes, a DCECNT routine to count the number of documents for each affix header, a FIXPOS routine to update the affix position numbers in the reference list, a SETNAM routine to set up file names according to whether author or subject Inverted Files are being processed, a NAM5 routine which converts the ASCII stem into 5-bit codes for insertion into the IFDS(A) directory file, and two segment-name incrementing routines IUPNAM and OUPNAM for updating to the next input or output Inverted File segment respectively.

The main program uses the foregoing routines as it executes the following steps.

1. Read the input and output file names from the command line arguments.
2. If the first argument is a 0, set the indicator IFDONE to avoid reading the old Inverted Files.
3. Initialize counters and indicators.
4. Set up storage buffers and their addresses.
5. Set file names via SETNAM.
6. Open table of contents file and input shred file.
7. If old Inverted File segments are all processed but there are more input shreds, go to Step 13 (if SAMESH false) or Step 31 (if SAMESH true).
8. If input shred file is also exhausted, then go to Step 41.
9. Open next input Inverted File segment and read first list header.
10. Use header to read next list and following header via GETLIS and compute pointers to various parts of the list.
11. If the use-same-shred indicator SAMESH is set, go to Step 18.
12. If the input shreds are all processed, go to Step 36.

13. Read the next input shred via GETSHR.
14. Sort the shred references on document number and term number via MYSORT.
15. Count the number of different document numbers in the sorted references.
16. Insert document count and affix count into shred header.
17. If old Inverted File all processed, go to Step 31.
18. Match the shred stem against the Inverted File list stem.
19. If shred stem is earlier in the sort order than the list stem, go to Step 31.
20. If list stem is earlier than shred stem, go to Step 35.
21. If stems are equal, merge affixes via AFFMER.
22. Add shred reference, document, computer word and affix counts to counts in list header.
23. Write list header and stem via PUTLIS.
24. Write affix list via PUTLIS.
25. Update affix numbers in references via FIXPOS.
26. Reset "use-same shred flag" SAMESH.
27. Merge shred and list references via REFMER.
28. Write merged references via PUTLIS.
29. If all Inverted File lists of this segment are processed, close this segment and go to Step 7.
30. If more lists in this segment, go to Step 10.
31. Write the shred list via PUTLIS.
32. Reset SAMESH to false.
33. If both Inverted File and shred lists are exhausted, go to Step 41.
34. If more shred lists, go to Step 11.
35. Set "use-same-shred flag," SAMESH, to true.
36. Write Inverted File list header and name via PUTLIS.
37. Write affix list via PUTLIS.
38. Set shred references count to zero force dummy reference merge.
39. Go to Step 27 to write references.
40. If Inverted File lists not done, then go to Step 35.
41. Blank pad and close last output segment.

42. Close shred file.
43. Write two directory files containing data stored by PUTLIS. (via ADDTAB).
44. Close table of contents file and any other open files.

Some of the more important procedures of the IFGEN module are discussed below.

SETNAM (MODE, SHD1, IFT, IFD, TAB): This procedure which resides in Part 2 (IFGENB), is called during initialization of the main program in Part 1 to set file names and affix parameters according to whether the MODE passed from the calling program indicates that subject/title files or author files are being processed. SHD1 is set to SORTS or SORTA, IFT is set to IFTABS or IFTABA, IFD is set to IFDS or IFDA, and TAB is set to STABLE or ATABLE, respectively. If the mode is author, CASEMK is set to mask out upper case bits in initials. For subjects, it is set to zero.

GETLIS ( ): This procedure is called by the main program to obtain the next Inverted File from the segment currently being updated. The header of the next list to be read has already been brought into core along with the previous list. This supplies the counts or lengths needed to determine how much storage is necessary to hold the remainder of the list. If the list is too large to be contained in one 432-word block, an overflow flag is set to force the continued reading of the list after the first block is processed.

The previously read and saved header is transferred to the top of the new list area and the counts are updated. The number of section pointers (if any) within the list are computed and subtracted from the word counts in the header, since they may not exist in the same quantity when the updated list is written.

The list is then read into core following its new header, along with the header of the next list is the segment. The next header is set aside for use by the next call to GETLIS and the word counts are extracted from this header to be used in looking ahead. When the end-of-file is read, GETLIS calls TESDON to see if any more input segments remain to be processed. TESDON attempts to open the next segment in sequence by calling IUPNAM. If that routine indicates there are no more input segments, TESDON returns a zero to GETLIS.

The core address of the list just read is returned as a value to the calling program.

GETSHR( ): This procedure is called by the main program to obtain the next shred list from the input shred file. The list is really a gathering of all shreds with the same stem (field 1), which have been clustered together by the SORT program described in Section 4.5. The first call to GETSHR is handled differently from all subsequent calls. This call is a kind of pump primer which causes the shreds to be read one shred ahead of the last one processed. This is due to reading the next header along with the current shred. The first call reads in the first shred header to get the length of the first shred. An array of this size is obtained from free storage and the shred along with the next shred header, is read into core. Shreds whose stem field is longer than 80 characters are rejected and the next shred is immediately read in.*

The shred stem is copied into a save area for comparison to the stem from the previous shred. In the case of the first shred, the stem is also copied into the previous stem area and no comparison is made. In this case, and in the cases where the new stem is the same as the previous one, the reference word for this shred is added to an array set up to hold the collection of references from shreds with similar stems and a reference counter is incremented.

The affix field of the new shred is compared to the affixes already collected for this list and if not like any other, it is added to the affix array. If it is similar to a previously stored affix, the reference count in that affix header is incremented. In both cases the position or sequence number of the affix in relation to the affix list is inserted into the appropriate component of the reference word. If the shred list is later merged with an Inverted File list, these numbers will be modified.

If the end of the shred file has not been reached, GETSHR loops back to read the next shred. This process continues until either a shred is read whose stem does not match the previous stem or the end of the shred file is reached. When either of these events occurs, GETSHR finishes the

*Later, in the main program, stems longer than 28 characters are ignored.

construction of the new list by inserting the affix list before the reference list, the previous stem before the affix list, and a list header (3 words) before the stem. The header counts and fence are computed and inserted and the address of the reference list is set aside for use by REFMER.

REFMER(R1, R2): When the main program finds a shred list whose stem matches a list from the input Inverted File, it must update the old list rather than create a new one. Updating includes merging of the new reference list gathered by GETSHR with the old Inverted File reference list. Such merging is the task of REFMER.

REFMER is called from the main program with two arguments, each pointing to one of the reference lists to be merged. The length of each list is extracted from the decrement portion of its pointer. Indices for stepping through the lists are initialized to zero. An index is set to point to an area obtained from free storage during main initialization which will be used to store the merged list. The index is incremented whenever a new output reference is stored.

References from the two lists are compared and the one with the higher document number is stored as output. Since all the new (shred) documents should be different from any already in the Inverted File, a message (1) is typed if identical document numbers are found. Both the old Inverted File list and the output list are buffered and the flushing and refilling of the buffers are controlled by the user of indicators.

One additional function of REFMER is the deletion of the section pointers fromthe old Inverted File list as it is read, examined, and merged. Pointers must be deleted even when no shred list is being merged and only the original Inverted File list is being output. In this case, REFMER is employed in a pseudo-call for the sole purpose of deleting these pointers. Such a call is triggered by using a shred list pointer argument of zero.

When the output list is completed, a pointer which contains the length (in decrement) and address of the new list, is returned as a value to the main program.

AFFMER (A1, A2, NA, PT): Another part of updating an Inverted File reference list whose stem matches that of a shred list is the combining of the affixes of the two lists. The old Inverted File list of affixes (A1) is transferred to an output array (obtained from free storage) and the shred affixes (A2) which are new and different (if any) are appended to the end of that group. During the transfer of the old list affixes, any section pointers which are encountered are removed. This must be done by employing pseudo-calls to AFFMER with zero for the second argument when no shred affixes are being merged.

In an actual combining of affix lists, the affix codes of each list are extracted and compared to each one in the other list. When different, the shred affix is added (along with its header) the output group and the count of affixes (NA) is incremented. When the new affix is the same as some Inverted File list's affix, the header counts of the old affix are updated to include the counts of the shred affix. A table of affix positions (PT) is filled as the affixes are combined into a single group. This table will show the relative position within the combined affix group of each shred affix, whether added to the end of the group or included with one already in the group. The table is later used by the procedure FIXPOS, to update the affix position number in the shred references.

<u>FIXPOS(SR, PT):</u> This procedure is called from the main program after a shred list has been combined with an Inverted File list and AFFMER has made a composite affix group and a table of affix position numbers for the reference list created from the shreds. This table provides a converter from affix sequence numbers relative to the original shred list to sequence numbers relative to the now combined affix list.

The number of references in the shred list is extracted from the decrement of the shred list pointer (SR) and is used to terminate the following processing loop:

1. The temporary affix number (inserted by GETSHR) is extracted from the current reference and used as an index to the position table.

2. The contents of that location of the table is taken and stored in the reference word in place of the temporary affix number.

3. The index to the reference list is moved up to the next reference.

FIXPOS is located in the second segment of IFGEN (IFGENB) and decides whether the affix position number is a 4-bit (subject endings) or a 7-bit (author initials) component on the basis of the variable CASEMK. This word is zero for processing subject files and contains a mask to screen out upper case bits in initials of author file affixes.

DCECNT (SR, SA): This procedure is called from the main program after a new shred list has been obtained via GETSHR and before AFFMER is called. The first argument SR is a pointer to the list of shred references and the second argument contains the number of affixes associated with the list. The number of references is extracted from the decrement portion of the pointer and the address of the first affix is computed. DCECNT then scans the reference list once for each affix in the group looking for affix position numbers corresponding to the affix being checked. When such a reference is found, its document number is compared to the document number of the one found previously. When the numbers differ, a count is incremented. At the completion of each reference list scan, this document count is inserted into the affix header and the list is scanned again for the next affix.

PUTLIS (LP, RL, HW):

PUTLIS is the output control procedure of IFGEN. It is called from several different places in the main program to output the various parts (header and stem, affix group, references) of the new Inverted File lists as they are pieced together by other sections of the program. The first argument (LP) contains a pointer to the data to be added to the output buffer — part of a new Inverted File list. The decrement of this pointer contains the length of the area to be output. The second argument (RL) contains the length of the entire list remaining to be processed (minus any blank padding). The third argument is non-zero only when the header (including the stem or name of the list) is the portion being written. This tells PUTLIS that it must check the size of the list against the space remaining in the output Inverted File section and determine if any blank padding is going to be needed to fill out unused words at the end of the section. Even when a list containing enough references to extend through several sections (432-word blocks) is being processed the length contained in RL may be used to compute how far into the last used section the list will extend. A minimum of ten computer words must be left over at the end of the list to accommodate the header of the next list. When this is not the case, the space must be padded with blanks and the number of blanks must be entered into the header of the list being processed before it can be output. The padding itself is not added to the output, of course, until the list is entirely written. Padding is also added when the last list of a segment is being processed. The final list will be indicated by a value of 10 for the third argument HW. This will force the pad-out of the remainder of the section.

PUTLIS counts both segments (files) and sections (records). When a new segment is opened, a heading is written into the table-of-contents file (e.g. SEGMENT n). When a new section is started, an entry is made to the table-of-contents file indicating that, Section No. n starts with lisnam and the offset is s. The list whose stem is lisnam will then appear in segment n at the depth of s. When s is greater than 1, it is because the previous list spilled over into this section. A section header containing this offset is written at the top of each section. As each new section is started, the first list stem is converted to 5-bit ASCII by NAM5 and added to the primary directory IFDS(A) by a call to the procedure ADDTAB. This routine also determines when the initial letter of the stem differs from that of the previous stem. When this is the case, the new initial is used on an index to the secondary directory IFTABS(A). This location is then filled with the index position of IFDS(A) where the new alphabetic group begins.

As each portion of a list is output, the section space remaining (SECREM) and the list length remaining (REMLIS) are reduced accordingly. When REMLIS reaches zero, any necessary blank padding is written to complete the remaining words of the section and, if the section count has reached ten or more, the segment is closed. On the next call to PUTLIS, a new segment name is obtained by calling the procedure OUPNAM, which increments the output file number and inserts it into the first name (SInnn or AInnn) of the output Inverted File.

C. Files Referenced:

SORTS(A)	date
S(A)Innn	old
S(A)Innn	date
date	S(A) TABLE
IFDS(A)	date
IFTABS(A)	date

D. Messages:

1. "Ifgen error has occurred. Error code is—e."
2. "Same D.N. found in I.F. and shred list with name lisnam D.N. is ---n."
3. "Error in writing output Inverted File segments"
4. "Error in opening output file no. f"
5. "Job done."

E. Source Files:

```
IFGENS ALGOL
     or
IFGENA ALGOL
IFGENB ALGOL
```

F. Loading Procedure:

```
LAED - ncload -   IFGENS(A) IFGENB (SRCH) UTILIB (AEDP)
SAVE IFGENS(A)
```

4.6 IFTEST SAVED

Purpose

To check the header counts of Inverted File lists

Description

IFTEST is an adaptation of IFLIST (described in the next section) and was originally designed to serve as a "dry run" for the creation of an Inverted File listing via IFLIST. Since IFGEN (described in the previous segment) has occassionally, due to its complexity, constructed a list with an improper header count, and since these counts are used to chain from one list to the next, it is important to verify the accuracy of these counts. IFTEST is now routinely used to verify a new set of Inverted Files, whether IFLIST is going to be run on these files or not.

The program performs the same chaining computations that are required of IFGEN and IFLIST (using code borrowed from IFLIST) but produces no output unless an erroneous count is detected. Then the Inverted File segment number, section number, depth into the segment and count which triggered the error are printed and processing is terminated. IFTEST may be resumed with arguments which select the subject/title or author files and which indicate if all the segments are to be tested or only selected ones.

A. Usage:

```
    1.   R IFTEST   AInnn   date
                    SInnn
    2.   R IFTEST   AI   date    s
                    SI
    3.   RIFTEST  batch  file
```

The example given in 1 above is the mode used regularly as part of the file generation procedure. The argument SInnn (or AInnn) specifies the first name of the segment where checking is to begin. The argument date specifies the second name of the file segments. IFTEST will proceed from this segment to the next sequentially until all existing segments are processed or until an error is encountered.

Example 2 allows the selection of an individual segment number (as given in s) of the author or subject files to be checked. In this mode, only one file segment is processed.

Example 3 allows selected segment names to be read from a line-marked batch file named in the command arguments. The file may be created by EDL[15] or QED[15] (CTSS edit programs) and consists of pairs of segment names, one pair per line.

B. Operation:

IFTEST reads the arguments from the command line and determines which of the options shown in Part A have been employed. If the author files are being tested, an indicator AUT is set. If a batch file name appears in the arguments, an indicator BATCH is set. If a single segment is being checked, the segment number is read from the command line and used to create the input file name. In this case, an indicator SEG is set.

Having made these preparations, IFTEST proceeds with the following steps:

1. If BATCH is set, extract the next segment name from the batch file.
2. If BATCH is not set, create the next segment name from the current segment number via UPNAM.
3. Initialize the section counter.
4. Open the segment for reading.
5. Read the next section into core and, if an end-of-file is encountered, set end-of-file flag and go to Step 20.
6. Initialize section pointers and increment section counter.
7. Check for section fence and, if absent, go to error exit.
8. Use section header offset to position section address pointer to first list.
9. Extract word count and blank count from list header.
10. Compute number of words remaining in this section.
11. Compute the number of section pointers in this list.
12. Get length of stem and length of affix group.
13. Save pointer to stem for possible error printout.
14. Save address of affix group.

15. Move list position pointer BP to start of reference list.
16. Extract number of references in this list from list header.
17. If references start in next section, read in next section via OFLIST — but reset READIN flag.
18. If reference list extends beyond end of section, use repeated calls to OFLIST to read through rest of reference list.
19. Check offset of each section header read and, if incorrect, go to error exit.
20. If READIN flag is off, meaning OFLIST not employed in Step 18, move list position pointer up to end of reference list.
21. Set up address pointer to next list header; see if still in this section; check for list header and, if absent, go to error exit.
22. Reset READIN flag.
23. Move list address up to next list.
24. If still in this section, go to Step 9.
25. If end-of-file flag not set, go to Step 5.
26. If end-of-file flag is set, close segment.
27. Reset end-of-file flag.
28. If SEG flag set, stop processing.
29. If BATCH flag set, get next file name from batch file.
30. If no more names in file, stop processing, else go to Step 3.
31. If BATCH flag not set, get next segment name via UPNAM.
32. If this segment exists, go to Step 3, else stop processing.

<u>ERROR EXITS:</u> Whenever a missing header or ɪence indicates that the chaining process has gotten off the track, or a reading error occurs, the type of error and location of the list being processed is printed and processing is terminated.

<u>OPNAM( ):</u> In this subroutine the integer holding the binary segment number is incremented by one and then converted to BCD via the CTSS utility DEFBC. The BCD number is then inserted into the right half (18 bits) of the first name of the Inverted File segments, thus creating a name of the form, SInnn or AInnn where nnn is now one larger than before.

OFLIST( ): This subroutine uses the CTSS procedure BFREAD (see Section 3.5.2.2) to read the next section of references in a list in which the references extend beyond the section where the list begins. It increments the main section counter and also a counter for the number of overflow sections being read in this list. An adjustment is made to the list position pointer BP which takes into account the original depth into the section where the list began and the size of the section. This adjustment, together with others at Step 18 above, ensures that BP will be positioned at the start of next list when all the overflow references have been read. Other such adjustments and preparations made by OFLIST include the setting of the flag READIN, the positioning of the section address pointer to the top of the section, and the resetting of the depth parameter to zero.

C. Files Referenced:

S(A)Innn date
batch file

D. Messages:

1. "premature end of file reached"
2. "read error"
3. "fence error"
4. "offset error"
5. "CWL count list header error"
6. "RFL count list header error"
7. "Last list contained --- lisnam" (where lisnam is the stem or name in the list being processed when an error occurred).
8. "Depth is d in file No. f " (where d is the list depth within the file whose name contains the segment number f).
9. "Batch file empty or nonexistant"

E. Source File:

IFTEST ALGOL

F. Loading Procedure:

```
LAED  -ncload-  IFTEST (SRCH) UTILIB (AEDP)
SAVE  IFTEST
```

4.7 IFLIST SAVED

Purpose

To create an ASCII-coded file listing the contents of the Inverted Files

Description

The Inverted Files contain both binary and ASCII data in a fairly complex format. IFLIST provides a means of presenting the information contained in a readable, tabulated form. The program produces an ASCII-coded disk file which may then be printed off-line.

The file generated is named SUBFIL IFLIST or AUTFIL IFLIST, depending upon whether subject or author files are being processed. The author names or subject/title stems are listed in the left column, showing the separation of stems and endings. Other columns contain the number of references associated with that stem, the number of different documents, and the number of affix strings.

Listed in the "Term" column are the subject/title stems taken from the Inverted File lists, or the Author names in the case of Author files. The subject/title stems are followed by a hyphen when an ending has actually been removed. If only one ending has been removed from a term word, then the ending appears after the hyphen ~~on the same~~ line. If more than one ending has been removed, the variations are listed in separate, indented entries under the stemmed term. A typical page of an Inverted File listing is shown in Fig. 4.3.

For author file listings, there is no stemming of endings and author's initials appear in place of the endings. Like multiple ending strings, multiple sets of initials appear in additional lines instead of appended to the name.

The second column contains the number of references associated with this list. This number should correspond to the number of references actually printed when full output is requested.

Column three contains the number of different documents designated by the references in that list.

Column four indicates the number of affix strings (endings or initials) associated with the term or name.

```
                              SI150 042771

    Term                      No. Refs  No. Docs  No. Fnds
martensite-ic                    1         1         1
8741 10 5 1 (2)  OP

marter-ials                      1         1         1
8321 5 3 1 (2)  OP

Martin-                          26        23        2
 Martin                          21        2
 Martin's                        5         4
16964 2 4 1 (1)  OP  16276 0 6 2 (3)  OP  15514 2 10 1 (1)  OP  14472 8 2 1 (3) J P  13530 6 2 1 (4)  OP
12334 7 6 1 (3)  OP  12195 1 15 1 (1)  OP  [illegible] 5 2 1 (4)  OP  7600 9 2 1 (4)  OP  622 3 14 2 (2)  OP
6122 10 3 1 (2)  OP  5726 7  1 (3)  OP  5725 4 2 1 (4)  OP  5663 18 2 2 (2)  OP  5663 19 6 1 (4)  OP
5261 13 1 1 (3)  OP  4614 6 . 1 (3)  OP  4593 20 2 1 (4)  OP  2803 10 2 2 (3)  OP  2603 30 2 2 (3)  OP
2516 10 2 1 (2)  OP  2516 11 2 1 (4)  OP  2371 5 2 1 (3)  OP  *2354 29 2 1 (4)  OP  1630 4 2 1 (3)  OP
1240 16 2 1 (4)  OP

martinsit-e                      1         1         1
3475 1 2 1 (1)  OP

Martius                          1         1         1
3500 11 2 1 (4)  OP

Marubeni                         1         1         1
10668 31 6 1 (3)  OP

Marx                             3         2         1
13906 5 1 1 (2)  OP  13906 5 9 1 (2)  OP  8570 4 6 1 (4)  OP

Mas-ing                          1         1         1
10340 4 1 1 (2)  OP

MASD                             1         1         1
10614 8 3 1 (3)  OP

maser-                           1154      215       3
 maser                           934       193
 masers                          217       91
 masering                        3         1
17709 1 8 1 (0)  OP  17418 8 2 1 (2)  OP  17418 9 2 1 (3)  OP  12949 7 6 1 (2)  OP  12949 14 6 1 (2)  OP
11627 3 5 1 (3)  OP  11625 0 5 1 (5)  OP  11625 3 5 1 (0)  OP  11347 3 3 . (2)  OP  10618 8 4 1 (2) J P
10618 10 2 1 (3) J P  10618 12 2 1 (3) J P  10618 13 5 1 (3) J P  10618 17 4 1 (3) J P  9940 1 11 1 (1)  OP

   •      •      •                                                                                   •

3645 18 3 1 (2)  OP  3645 19 7 1 (2)  OP  3072 17 9 1 (3)  OP  3071 11 3 1 (4)  OP  2494 3 5 . (2)  OP
2224 0 6 1 (5)  OP  2224 1 6 1 (1)  OP  2224 2 3 1 (4)  OP  1875 0 6 1 (5)  OP  1876 3 2 1 (2)  OP
1876 4 2 1 (4)  OP  1681 0 5 1 (5)  OP  1681 1 5 1 (1)  OP  1669 2 1 2 (0)  OP

mask-                            23        14        6
 masked                          2         2
 mask                            17        8
 masks                           1         1
 masking                         3         3
25713 5 1 4 (2)  OP  25490 5 3 4 (3)  OP  15571 1 14 2 (1)  OP  15[illegible] 4 4 2 (3)  OP  13665 4 6 . (3)  OP
13145 8 5 2 (4)  OP  12956 6 5 2 (3)  OP  12942 1 9 4 (3)  OP  11572 10 4 2 (3)  OP  8575 0 14 . (5)  OP
8575 1 15 2 (1)  OP  7234 1 11 3 (1)  OP  5721 2 3 2 (1)  OP  5721 3 10 2 (3)  OP  5721 4 3 2 (3)  OP
5721 4 1 2 (3)  OP  5721 5 8 2 (2)  OP  5721 6 1 2 (2)  OP  5721 7 8 2 (2)  OP  5721 8 9 2 (4)  OP
5721 9 2 2 (3)  OP  3212 3 6 1 (2)  OP  2517 5 3 1 (4)  OP

Mason-                           26        18        2
 Mason                           25        17
 Mason's                         1         1
17920 1 4 1 (1)  OP  17920 3 6 1 (2)  OP  1792 4 2 1 (2)  OP  17

   •      •      •                                                                                   •
```

Fig. 4.3   Sample Page of I.F. Listing

When multiple affix strings are listed, the reference count and the document for each affix is shown in the appropriate columns.

When the listing of references is requested by the mode argument, the contents of each reference word is shown in the following format: DN TN WN EN (WT) WJO. DN is the document number, TN the term number within the document, WN the word number within the term, and EN the affix position associated with the term. The weight (0-5) appears within the parentheses and the property code is indicated by the presence of one or more of the letters W, J, O (whole work, journal article, original work).

Up to five references are printed to a line with as many lines as needed used to print the entire collection of references in the list.

A. Usage:

```
        1:    R IFLIST  SInnn  date  -mode-
        2.    R IFLIST  AI  date mode s
        3.    R IFLIST  batch file  -mode-
```

The above options of IFLIST usage correspond to the options described in Section 4.7 for IFTEST except that the mode argument allows reference word contents to be included in or excluded from the listing. A mode of 0 or the ommision of a mode (examples 1 and 3 only) produces the full reference listing. A mode of 1 causes references to be suppressed.

B. Operation:

After initializing pointers and flags, IFLIST reads the arguments from the command line and determines which of the options of Part A have been employed. If the first argument is of the form shown in example 1, then the starting segment number is extracted from the file name in the first argument. If the first argument indicates that author files are to be listed, then an indicator AUT is set. If a batch file name appears in arguments one and two instead of Inverted File names, an indicator BATCH is set.

An attempt is then made to read a third argument (mode) from the command buffer. If this argument exists, it is used to set or reset an indicator NOREFS, according to whether the mode is a 0 or a 1. If it does not exist, NOREFS remains reset (false).

If a mode argument is found, an attempt is made to extract a fourth argument, which if present is used as the segment number to be processed. An indicator SEG is set to prevent processing beyond this segment.

With these preparations having been made, IFLIST loops through the following steps as it produces its output file.

1. If BATCH is set, extract the next segment name from the batch file.
2. If BATCH is not set, create the next segment name from the current segment number via UPNAM.
3. Initialize the output buffer pointer to the top of the buffer
4. Output a carriage return and three tabs followed by the name of the segment being listed.
5. Output two carriage returns, the column headings Term, No. of Refs., No. Docs., No. Ends., and two more carriage returns.
6. Open the output file for writing (SUBFIL IFLIST for subject/title files, AUTFIL IFLIST for author files).
7. Open the current input segment for reading.
8. Read the next section into the input buffer and, if the end-of-file is encountered, set the end-of-file flag and go to Step 61.
9. Set section address pointer to the top of the section and reset section depth to zero.
10. Check for section fence and, if absent, go to error exit.
11. Use section header offset to position section address pointer to first list.
12. Check for list fence and, if absent, go to error exit.
13. Extract number of blank words and length of name or stem from list header and construct a pointer to the name or stem.
14. Return any free storage used on last call to the affix processing procedure GROUP.
15. Extract the number of affixes from the list header.
16. Place list position pointer BP at address of first reference.
17. Compute the address of the first affix.
18. Construct a list of all affix codes in the list via GROUP.
19. If only one affix in list, go to Step 58.
20. Output the name or stem.

21 Output a hyphen if AUT is not set, a comma if it is set.

22. Output necessary tabs to get to column two via TABIN.

23. Columnize the reference, document, and affix counts via COLNUM.

24. Output a carriage return to end this line.

25. If single-affix flag EDSONE is set, reset it and go to Step 43.

26. Set up loop to attach affix codes to name or stem.

27. Output 1 space for indentation.

28. Output the name or stem.

29. If AUT flag is on, go to Step 33.

30. Convert the next ending code saved by GROUP to ASCII characters via GETEND (see Section 3.2.6.3).

31. If the EDSONE flag is set and GETEND returned a converted ending, output a hyphen and add 1 to the affix (ending) length.

32. Bypass following author file steps by going to Step 35.

33. Create a pointer to the initials string.

34. Output a comma and add 1 to the affix (initials) length.

35. Output this affix character string.

36. Add affix length to name or stem length.

37. If EDSONE flag is set, go to Step 22.

38. Output necessary tabs to get to column two via TABIN.

39. Columnize the reference and document counts for this affix via COLNUM.

40. Output a carriage return.

41. Move affix address pointer up to next affix in list and reset affix length to zero.

42. If more affixes in list, loop back to append next affix by going to Step 27.

43. If all affixes are processed, extract reference count from list header.

44. If references start in next section, read in next section via OFLIST but reset READIN flag.

45. If NOREFS flag is not set, go to Step 48.

46. If not outputting references (NOREFS set) and reference list extends beyond end of this section, use repeated calls to OFLIST to read through rest of reference list.

47. When all references read in, go to Step 59.

48. Set references-output-per-line parameter N to 5.

49. If reference count is less than N, go to Step 57.

50. If next reference is in next section, read in the next section via OFLIST.

51. Dissect next reference and output its relevant data via CUTREF.

52. If output buffer address is within fifty computer words of the end of buffer, write out and reset buffer via RITOUT.

53. Increment list position pointer to next reference.

54. If N references have not yet been done, go to Step 50.

55. If N references finished, reduce the reference count by N and, if count is now zero, go to Step 59.

56. If more references to do, output a carriage return and go to Step 49.

57. Set N equal to remaining reference count and go to Step 49.

58. (from Step 19 only) Set the single-affix flag, EDSONE, set the affix processing loop counter to zero, and go to Step 28.

59. (from Step 55) Output two carriage returns between lists.

60. If NOREFS flag is set and READIN flag is not set, move list position pointer BP past references.

61. If output is at a word boundary and buffer is over half full, write out buffer contents and reset buffer via RITOUT.

62. Move section address pointer to start of next list and update depth into section.

63. Reset READIN flag.

64. If new list starts in this section, go to Step 12.

65. If end-of-file flag is not set, then go to Step 8.

66. Close segment.

67. Reset end-of-file flag.

68. If single-segment flag S[illegible] is set, then go to Step 72.

69. If BATCH flag is set, extract next segment name from batch file.

70. If no more names in batch file, go to Step 72.

71. If BATCH flag is not set, get next segment name via UPN[illegible]M.

72. Close the output file.*

73. If next segment exists, go to Step 3.

74. If no more segments to process, terminate and return to CTSS.

*The output file is closed and reopened for each segment processed so that an error which fails to take a prescribed error exit will not cause the entire output file to be lost.

ERROR EXITS

Although the prior use of IFTEST (Section 4.6) should catch any fence, count or offset errors in the Inverted File, error exits are provided in IFLIST which will cause processing to be terminated if the program detects such an error. The type of error and number of the segment in which it occurred are printed out on-line and the output file is closed before the program halts.

Some of the more important subroutines are described below.

UPNAM( ): This subroutine increments the segment counter and converts the new number to BCD via the CTSS utility procedure DEFBC. The BCD number is then inserted into the right half of the first name of the Inverted File to make a segment name of the form SInnn or AInnn where nnn is now one larger than before.

OFLIST( ): This subroutine uses the CTSS procedure BFREAD (see Section 3.5.2.2) to read references in a list in which the references extend beyond the section where the list begins. In the case where the procedure GROUP has read into the next section to complete an affix list which crosses a section boundary the amount of the extension into the new section is stored by GROUP in a parameter PRE. OFLIST uses PRE in controlling the address into which to read the new (partial) section and the portion of the section to be read. OFLIST also adjusts the list position pointer BP, the section address pointer and the section depth to reflect the new section contents and sets the flag READIN to indicate that these adjustments have been made.

RITOUT( ): This subroutine is used to empty the output buffer by calling the CTSS procedure BFWRIT (see Section 3.5.2.3) which writes the contents of the buffer into the disk file SUBFIL (or AUTFIL) IFLIST . The number of words written is computed from the current storage address. Any unfilled bytes in the current address are padded with null codes. The current storage address is then set to the start of the output buffer for refilling.

TABIN(L): This subroutine is used to compute the number of tabs (10 spaces to a tab) needed to reach the second column in the output format. This column is set at the 30th character position or 3rd tab stop of the line. The argument L contains the current number of characters already on the line, and therefore, determines how many tabs will be needed. If L is thirty or more, then the third tab stop has already been reached. In this case, a carriage return is output followed by three tabs. If L is less than thirty, the number of tabs needed is computed and output.

COLNUM(NUMS): This subroutine is used to output the counts of references, documents, and, where applicable, affixes related to the name or stem in column one. Each count is followed by a tab code which positions the next count in a new column. Since the number of arguments containing counts varies, the AED argument reading procedure ISARGV (Section 3.6.1.3) is used to fetch the arguments. Each number received is converted to ASCII via INTASC (Section 3.4.2.3) before being stored in the output buffer.

CODES = GROUP(E, N): This valued subroutine is used to gather the affixes of a given list into a consolidated group. In the case where the affixes span from the end of one section into the next, they are moved to a temporary storage area obtained from free storage, and the affix address is changed to this area.

When subject/title ending codes are being processed, another kind of consolidation also takes place. The ending codes are copied from the original list into an array containing only the codes and no accompanying affix headers. This step is left-over from the time when phrases (whole index terms) were included in the Inverted File and all the ending codes related to the phrase had to be unpacked from each "affix string". Since only single words are included now, this step could be eliminated.

The address of the consolided list of codes is returned as a value to the calling program.

CUTREF(R): This subroutine extracts the components of a reference word whose address is given the argument and outputs the data which those components represent. First, the document number is extracted, converted to ASCII via INTASC, and copied into the output buffer. Similar treatment is given to the term number (position of the term in subjects field or author name in author field), the word numb (word position within the term), and the affix number (position of the related affix within the affix group). Next, the weight (range number) is taken and is output within parentheses.

The three bits of the property code, standing for "whole work", "journal article", and "original work", are tested. When a bit is on, the letter W, J, or O are output, respectively.

A single space is output between each of the main elements listed above (but not between the letters WJO except to take the place of an unused letter). Two spaces are included at the end of CUTREF to separate this reference's data from the next.

C. Files Referenced:

```
SInnn    date
AInnn    date
SUBFIL  IFLIST
AUTFIL  IFLIST
```

D. Messages:

1. "premature end of file reached (plus message No. 6)"
2. "write error"
3. "read error"
4. "list header error"
5. "Last list name contained ---LISNAM (plus message No. 6)"
6. "in file No. F" (where F is the number which is part of the name of the segment in error).
7. "Batch file empty or non-existant"

E. Source File:

```
IFLIST  ALGOL
```

F. Loading Procedure:

```
LAED  -ncload-    IFLIST  STEM2 (SRCH) UTILIB (AEDP)
SAVE  IFLIST
```

V. SUPPORTING SOFTWARE

In this chapter, programs are described which provide supporting roles to the operation of Intrex but which are not part of the generation or updating of the data base. Like the programs described in Chapter IV, these are independantly compiled, loaded, and executed programs, each designed to perform a specific task.

The format of the descriptive sections is the same as in Chapter IV (See Chapter IV introduction for explanation of sub-sections).

5.1 DIRGEN

Purpose

To create a label-controlled directory to disk- and core-stored message text.

Description

DIRGEN (DIRectory GENerator) creates a directory file which serves as an index to both a disk file and a core-stored table of ASCII coded messages. The message files may be created and altered via the CTSS console using the editing programs EDA or QED.[15] Each message is delimited at each end by a special code and prefaced by a label consisting of up to six ASCII characters, the first of which is alphabetic. The labels are delimited at each end by another special code. Other characters sometimes appear elsewhere in the message file which help format the file print-out or serve as comments. Only the characters between pairs of special codes are processed by DIRGEN however.

DIRGEN reads the message file, finds and checks the length of labels and messages, and places each label in the directory, with a pointer to the location of the accompanying message.

The disk-stored messages (designated by labels set off with $'s) are written onto a disk-text file. To conserve space, only the message text is written — the labels are omitted. The core-stored messages (designated by labels set off with /'s) are added to the end of the directory. Directory entries contain a BCD coded representation of the message label and a message file position pointer. If bit 2 of the pointer is a 1, the message pointed to is a core-stored message.

The last name of the input message file is TEXT. The last name of the output message file is DISTXT. The last name of the directory file created by DIRGEN is DIRTAB. The first name will be the same for both output files and is specified when DIRGEN SAVED is Resumed by entering it as the first argument of the command (See Part A below).

The directory file must be read into core by calling INITYP (name1.) before the first call to TYPEIT which has a disk-message or core-message label as an argument.

DIRGEN will use the $ code as the disk label delimiter and the / code as the core label delimiter. A second argument may be given in the Resume DIRGEN command specifying a substitute character to be used for the $. No provision has been made to substitute for the slash.

Similarly, DIRGEN will use the * code as the message delimiter, unless a substitute code is specified in the third argument (optional) of the Resume DIRGEN command.

In general, short, frequently used message labels are kept in core, while longer, or infrequently used lables are stored on the disk for most efficiency.

A. Usage: R DIRGEN mfile -ld- -md-

In the above command, mfile is the first name of an ASCII file (whose last name is TEXT) containing the messages and labels to be processed. The optional arguments, ld and md are substitute label delimiter and message delimiter characters to be used if the standard $ and / are undesirable for some reason.

B. Operation:

The arguments are read from the command buffer and used to set the name of the TEXT file to be processed and, if the second and third arguments exist, to establish the new delimiters. If a directory file with this first name already exists, it and the corresponding DISTXT file are deleted so as to start with a clear slate. Then the TEXT file is opened for reading and the DISTXT file is opened for writing and address pointers to a core-message storage area and a disk-message storage area are set to their starting locations. Finally, all flags and counters are initialized to zero and the first block of data from the TEXT file is read into core by the subroutine SUMMON.

Having made the above preparations, DIRGEN loops through the following steps as it processes the TEXT file and creates the DISTXT and DIRTAB files:

1. Examine input characters looking for a disk-label or core-label delimiter.
2. When a label delimiter is found (set core-label flag if it is a core-label), look for similar delimiter within the next seven characters.
3. If no second label delimiter is found within seven characters, go to error exit.
4. Convert label string to BCD and see if it is already in the directory table formed so far.
5. If a similar label is already in the table, store this one in a special array, set the "ignore flag", and skip next step.
6. Add this label to the directory table and increment the table index by two.
7. Examine more input characters looking for a text delimiter.
8. If at any time during character scanning the end of the input buffer is reached, call SUMMON to read in more input.
9. When a text delimiter is found, copy text which follows into either the disk-message buffer or the core-message buffer, depending upon setting of the core-label flag.
10. Insert address of the buffer area just filled into the directory table just after the label inserted in step 6.
11. If core table address inserted above, set a special core bit (bit 2) in the pointer.
12. If disk-message buffer is filled during step 9, write buffer into DISTXT file and reset buffer parameters.
13. If end of input buffer is reached before a second delimiter is found, call SUMMON to read more.
14. Insert character count of input between text delimiters into decrement of text pointer just added to table.
15. Reset flags and return to step 1.

When SUMMON finds no more input text to be read, DIRGEN completes the output files and prints a list of the labels created. The directory file DIRTAB is written in two stages, the pointer/label table and the core-message text, but only after the pointer addresses related to the core-stored messages are adjusted to include the length of the directory table to which it is appended. After printing an on-line report of the number of labels in the new table, DIRGEN prints any multiply defined labels set aside earlier and a list is made of all created labels. Then DIRGEN closes all files and halts.

SUMMON( ): This subroutine reads the next block of input data into the input buffer. The exact amount of data and starting storage address can be adjusted by setting certain parameters outside of SUMMON. The CTSS I/O utility BFREAD, described in section 3.5.2.2, is used for reading. If this routine encounters an end-of-file mark, return to SUMMON is made to a place where a flag is set. This flag is tested upon the next entrance to SUMMON and, when found to be on, transfer is made out of SUMMON to the wrap-up area of DIRGEN.

After reading a new block of input, SUMMON reinitializes the input pointer, length, and character count before returning to the calling program

C. Files Referenced:

Name		Function
name1	TEXT	message text (input)
name1	DISTXT	text without labels (output)
name1	DIRTAB	directory table and core-messages (output)

D. Messages:

1. "error in reading message file"
2. "error in writing disk text file"
3. "too many characters in label --- long label message skipped"
4. "label delimiter not found within 400 characters — improper file"

5. "text delimiter not found within 20 characters — improper file"

6. "directory of -N message pointers created"

7. "the following labels are multiply-defined, only 1 set occurrence used

   <u>lab 1</u>

   <u>lab 2</u>

   etc."

   (complete list of labels created).

8. "directory labels are

   <u>lab1</u>

   <u>lab2</u>

   etc."

   (complete list of labels created)

E. Source File:

DIRGEN ALGOL

F. Loading Procedure:

```
LAED   - ncload-   DIRGEN  (SRCH)   UTILIB  (AEDP)
SAVE   DIRGEN
```

5.2 PRINTM, PRNASC

Purpose

To print ASCII files on line.

Description

A. Usage: R PRINTM / PNNASC name1 name2 -wdct-

The programs PRINTM and PRNASC will print the ASCII file name1 name2 starting at computer word wdct. If the final argument, wdct is omitted, the file will be printed from the top. If there are non-printing characters in the file, they will be represented by the expression <nnn>, where nnn is the octal representation of the character. In addition, PRINTM prints formatting codes, such as space, horizontal tab, and carriage return, as octal codes rather than transmitting them to the console as effective control characters. PRINTM transmits the carriage return control character after every 84 characters of output. PRNASC, in contrast, transmits the control characters that occur in the text and does not send any of its own.

B. Operation:

PRNASC and PRINTM begin by extracting arguments from the command line: name1, name2 and, if included, the first word position WDCT. FSTATE is called to determine the length of file name1 name2. If the length is less than wdct, an error message is printed and the program quits. Otherwise, a block of 432 words is read from the file. If an end-of-file condition occurs, the EOF indicator is set, Next, TYPASC is called to print the contents of the buffer. If the EOF flag is not set, the next block is read and printed.

The operation of the sub-procedures TYPASC, TRASCI and PRT12 is explained below:

TYPASC (Ascptr): TYPASC performs the following steps:

1. TYPASC allocates an output area which is seven times the size of the block of ASCII code pointed to by Ascptr.

2. TYPASC calls TRASCI (Ascptr, Out). TRASCI translates the ASCII code in the area pointed to by Ascptr into 12-bit BCD, which is stored in the area pointed to by Out.

3. TYPASC calls PRT12(Out), which prints the 12-bit characters.

4. TYPASC releases the storage used as an output area and returns control to the main procedure.

<u>TRASCI (Ascptr, Out):</u> TRASCI performs the following steps:

1. Using the string manipulation procedures GET and INC, TRASCI steps through the input area, extracting characters one by one.

2. If an ASCII character is non-printing, it is translated into the expression < nnn >, where nnn is its octal equivalent.

3. If an ASCII character is printable, it is first translated to 12-bit by ASCTSS and then inserted in the output buffer by PUTOUT.

<u>PRT12(Output):</u> PRT12 uses WRFLX to print the 12-bit characters pointed to by Output.

C. Files Referenced:

name1, name2: any ASCII file

D. Messages:

1. "Given starting word beyond end of file."

E. Source File:

```
PRINTM  ALGOL
PRNASC  ALGOL
```

F. Loading Procedure:

```
LAED  PRINTM  (SRCH)  NOLIB
              or
LAED  PRNASC  (SRCH)  NOLIB
SAVE PRINTM  (or PRNASC)
```

5.3 UPDIR

Purpose

To update a master file directory

Description

After a batch of input files have been added to the data base, they are written out on tape and deleted from disk. The project TIP program PUTOUT transfers them to tape and simultaneously creates a directory for the tape. The tape is copied on to a master by means of a 1401 program. Finally, UPDIR is used to combine the directory for the current batch of files with the directory for the master tape.

A. Usage R UPDIR master -direct- -inc- -direct- -1-

"master direct" is the name of the file to be updated. "inc direct" contains the new (incremental) directory information. If master direct has never been updated, this must be indicated by the argument "1," otherwise that argument is omitted.

B. Operation:

The directories are six-bit line-marked BCD-coded files. A file is defined in the directories by a line with the following format:

name1 name2 wdct * line

"name1 name2" is the name of the file. "wdct" is the length of the file in words. "line" is the line number.

UPDIR combines the two directories by reading the master directory into a work area and then reading in the incremental directory immediately behind it. At this point, UPDIR is left with two tasks. First, it must step through the second part of the combined directory and increment all of the line numbers by n, where n is the number of lines in the first part. Secondly, since the last record of the master file is padded out with zeroes, it must make an entry in the directory to account for it. UPDIR steps through the new entries and totals the word counts. The sum is divided by 432 (the number of words per record) and the remainder is used as the length of a dummy entry, called BLANK PADDING. Finally, UPDIR deletes the old master directory and writes out the new one.

C. Files Referenced:

master directory (name entered in command line)
Increment directory (name entered in command line)

D. Messages:

1. 'Can't find last" (last line of master directory)
2. 'Error in reading master file.'
3. "Error in reading incremental file."

E. Source File:

```
UPDIR ALGOL
```

F. Loading Procedure:

```
LAED  UPDIR (SRCH) UTILIB
SAVE UPDIR
```

VI. INTREX BEYOND 1971

The preceeding chapters described the Intrex Retrieval System as it has been implemented to date. This chapter discusses some additional functions that we hope to implement, the factors involved in exporting Intrex, and the characteristics of future generations of Intrex-like retrieval systems.

6.1 Modification of the Intrex System

Intrex, being an experimental retrieval system, has been a continually growing system. This report is being written at a time when growth of the current system has leveled off so that future new features will not significantly change the basic organization and operation of the system. However, some features yet to be implemented would improve operating efficiency while others would affect the user interface and permit more flexible operation and greater convenience. Some of these features are discussed below.

6.1.1 Extended Primary Search Capability

Currently the argument of a subject search command is a string of search words all of which (in their stemmed version) must be found in one of the index phrases for the document. Searches are made more flexible by the ability to name lists of documents resulting from a search request and to combine the named lists via Boolean operations. The need to name lists however, may be an unneccessary encumbrance upon a user who is interested in only the resultant combined list. In such cases it would be desirable to incorporate the Boolean operations directly in the search requests. For example the request

subject (iron oxides <u>or</u> ferrous oxides) <u>and</u> structural degradation

would be equivalent to the sequence of commands currently necessary:

```
subject iron oxides
name iron/subject ferrous oxides
or iron/name ironfe/subject structural degradation
and ironfe
```

Note the use of parentheses in the first request to indicate order of Boolean operations.

The Boolean operations might also be used between commands rather than strictly within the arguments of a single command. For example

subject ferrous oxides and not author pearlman

would produce all documents indexed under "ferr-ous oxid-es" not written by Mr. Pearlman.

Another useful extension of the subject and the title search commands would be the ability to control stemming. For example a command of the form

subject reactor+

would retrieve documents indexed under "reactor" and "reactors" but not those indexed under "reactivity" ~~or~~ "reactivation". Similarly a command of the form

subject past*

would retrieve documents indexed under [illegible] not those indexed under "pastel" or "paste", thus eliminating some false drops. As pointed out in the earlier chapters the provision for adding this capability already exists in the structure of the inverted files.

A third feature which would also effect the precision of retrieval is user controlled word adjacency. That is, a search request of the form

subject high-temperature

would match only documents where "high" preceeds "temperature" with no intervening words in the index phrase.

An extension of the "autho[illegible]" search command would permit searching of documents with joint authorship. For example the command

author brown, j. s. little, c. w.

would retrieve all documents authored jointly by Brown and Little. This command would have the same effect as a similar command issued with the Boolean connective and between the names but would be shorter and would place the syntax of the author command in closer agreement with that of the other search commands.

A final but more subtle point of improvement is to make all commands immediately executable. That is, the CLP module, responsible for initiating commands, would execute a command immediately upon encountering either a slash (/) or a carriage return delimiter. Currently, commands are interpreted a line at a time in such a way that certain, apparently reasonable command sequences cannot be executed. For example, the sequence

```
subject aluminum/output title
```

is permitted, but the sequence

```
subject aluminum/name alu/output title
```

is not. With the suggested change there would be no restrictions on the commands that could be combined on a single line. This would result in a command language syntax that would be much clearer to users. In the process of implementing this modification, the current "anding" capability implied by a series of search commands separated by slashes would be replaced by the more explicit Boolean capability described above.

6.1.2 Extended Output Features

Several extended features related to output appear to be desirable. One of the most important of these features, and one which is in fact being implemented at this writing, is the request for output off-line. This feature will permit a user to initiate a search and examine results at a later time. It will also permit a user at a display terminal to conveniently obtain a hard-copy list of the documents he retrieved together with pertinent information about each document. In addition, this feature will prepare the way for a mode of operation where users, remote from the consoles, can request searches (via telephone or through the mail) and have results delivered to them (the so-called "delegated" search mode).

Another improvement would automatically decode certain information in the catalog before presenting it to the user. For example, within the location field the journal name is represented as a CODEN and the date is represented in a coded form. Both of these could be automatically interpreted. In addition, certain "transfer codes" appearing in the catalog indicate that information pertaining to a field is contained in another catalog record. This information could be located by the Intrex programs and presented directly to the user whenever a transfer code is encountered. Other codes of lesser utility such as the cataloger code, could also be interpreted.

Various other improvements in the output format are possible. Further extentions of existing commands could permit increased flexibility— allowing output of information for the n^th^ document on a list, increased paging capability for text access, and so on.

6.1.3 Other Search Aids

Several other features could be implemented that would allow users to control the general searching process more completely and improve precision of retrieval. The RESTRICT command which selects documents from an already retrieved list based on the appearance of a given string in the index terms could be modified to function in a manner more like the generalized search command of Section 6.1.1. A second minor improvement would be to implement a form of this command that would apply to the pro-pe codes of documents:

> J - journal article (or not), O - original work, and P - professional level.

In keeping with the immediately executable principle for commands (Section 6.1.1), the RANGE command should be reprogrammed to apply *after* a search command has been executed. Then the command could be given either as part of the search request (as now) or separately after the search has been executed and acknowledged.

A problem for the user of any retrieval system is to think of synonymous and other related terms in order to improve his search strategy. Currently, users of Intrex are aided in chosing synonyms by a printed thesaurus available at the console. A better arrangement would be to have a thesaurus related to the inverted file available on-line and displayable at the user's request.

Another feature related to user aids for more quick and convenient searching is a more sophisticated form of the on-line guide. Currently a user types "info n" to display the nth selection of the guide. An easier to use implementation would permit following the INFO command word by command names or other words intuitively related to the concept for which the user needs help. Thus the command INFO SIMPLE SEARCH would be equivalent to the current command INFO 2. The implementation of this feature could range from a set of tagged messages to a context search of the guide, to a mini retrieval program complete with inverted file to sections of the on-line guide.

6.1.4 Data Base Update

Since the current Intrex program has been aimed toward establishing a data base sufficiently large for meaningful experimentation, the programs described in Chapter IV were designed only for expansion of the data base. However, at this writing there is no general procedure for removing and/or replacing documents in the data base. A program for removal of documents must 1) remove records from the catalog and remove corresponding entries in the catalog directory and 2) delete appropriate document numbers (pointers) from the lists in the inverted file. The latter is the more difficult of the two operations.

An approach to reducing the data base without completely removing documents is as follows. Catalog records for some selected documents are reduced to little more than title and author, and the catalog directory is adjusted correspondingly. References to these documents resulting from terms other than those found in the title are removed from the inverted file. Thus searches on title terms can still locate the document. This approach reduces the storage associated with the document by about 80%. The inverted file operations however are more complex than those required for complete removal of a document. In order to strike a compromise between size of data base and economics of storage we have already embarked on an effort to reduce the data base in this way. The programs written for this purpose can be easily modified to completely remove documents when that becomes desirable.

6.2 Exportation of Intrex

The current Intrex programs were designed primarily to support the goal of experimentation with some advanced novel concepts in information retrieval. Therefore the question of transferral of these programs to other computers has been subordinated to the primary goal of experimentation. Indeed, the programs have been somewhat customized to the CTSS system in order to maximize operational efficiency. As a result, the transition to another computer system would be made difficult for a number of reasons including

1) The lack of an AED source language compiler on many computer systems.

2) The dependence of variables, pointers, coding schemes, etc. on the 36-bit word size of the 7094.

3) The intimate relation of some program features to system features, the structure of which is peculiar to CTSS (file system, I/O techniques, interrupts, etc.)

In addition to these fundamental difficulties, certain subroutines have been written in machine language for more efficient operation. The exportation of the present system to a machine other than the IBM 7094 would require complete recoding of these programs.

Although the current Intrex programs are not easily exportable, in themselves, the concepts upon which the system is based were designed to be exported and hopefully will be. The most effective vehicle for their exportation is probably another retrieval system, Intrex II, designed both to incorporate all the salient features of the present system and to be more easily adapted to other computers. Further discussion of the requirements and properties of the next generation of the Intrex Retrieval System is found in the next section.

6.3 Future Generations of Intrex

The present Intrex Retrieval System was designed as an experimental vehicle in which several important new concepts in library automation would be tested. A second-generation more operational Intrex would probably differ from the experimental system in a number of ways.

First, the user interface would be somewhat more sophisticated, encompassing at least the additional features described in Section 6.1 and perhaps other features of a more advanced nature. For example on-line instructional aids might be provided that adapt to the user's past experience and suggest customized search strategies.

Secondly, there would be significant changes in the augmented catalog. The catalog record would be much shorter— several fields would be either eliminated or compacted. We anticipate the result would be a reduction (in words per record) of 2.5 or 3 to 1. Just which fields would be eliminated or reduced must be determined from further experimentation with the present catalog*. Present indications are however that the number of subject terms could be reduced, the abstract could be eliminated where available through text access, and several shorter fields of less utility could be removed. That is not to suggest that these fields are of no value but that for near future systems the benefits do not justify the additional costs.

Third, there would be changes in the general progr[illegible] architecture and structure of Intrex. To a certain extent this structure would depend on whether the system operates on a large general-purpose time sharing system or a smaller dedicated machine. We believe that there are many reasons to favor the dedicated machine and that ultimately information retrieval centers will have to have machines dedicated to their use. In either case future generations of Intrex should take advantage of the many advanced features available on modern third-generation computer systems. For example in the current environment, Intrex runs as an ordinary user under the CTSS supervisor. This means that each user of Intrex has a separate copy of the Intrex programs which is swapped in and out of core by the supervisor in the process of time sharing. There is no provision on this early system to

* An item related to the catalog itself is the indexing process, which produces the catalog record. Questions related to the amount and depth of indexing must still be resolved by our experiments and will relate to the size and format of the catalog.

take advantage of the fact that all Intrex users are in fact employing the same basic code and therefore could share it. In many more modern computer systems it is possible for users to share code and data in core through techniques such as segmentation (on MULTICS[16]) and reentrant programming. Such operation would be expecially desirable for a machine dedicated to information retrieval since it could completely eliminate the need for swapping. Even without shared program code, large memories and the ability to perform multiprogramming in most machines reduces swapping to some extent since more than one potentially active user program can reside in core at a given time.

Several other features are available on the newer machines which should serve to increase the operational efficiency of the Intrex programs significantly. Not the least of these features is the ability to address bytes of information directly through hardware features and special related instructions. A very large portion of operating time of the current Intrex System is consumed by character string manipulation, i.e., packing and unpacking of bytes in a computer word, and code conversion. Through direct byte addressing the packing and unpacking could be completely eliminated. The code conversion although generally less time consuming than packing operations, could also be accelerated on some machines by the availability of special microprogrammed code conversion "super-instructions".

A fourth way in which future generations of Intrex would differ is in the data base as it relates to the file system. The relation of the Intrex data base to the CTSS disk file system has been discussed at length by Kusik[9]. Synoptically, the CTSS file system, which is not unlike a paged memory, is well suited to an environment where files of varying length are continually being created, deleted, and modified. In this kind of environment the file system makes best use of the available hardware storage resources. When the set of files is well structured and static as in the Intrex data base, the CTSS file system is quite slow and inefficient. The inefficiency is primarily due to the fact that records comprising a file are scattered at random throughout the disk and chained together so that a large number of disk accesses are needed to locate a particular piece of information in a file of moderate length. A better approach to the file system structure has bee proposed by Kusik in which contiguous records of a file are put on the same or adjacent disk tracks. Directories to the files (such as CATDIR for the catalog records) are distributed on the disk in such a manner that an entry in the directory points only to records in the same cylinder. This arrangement minimizes the total access time for a given piece of information by eliminating "chaining" and minimizing disk seek time. In addition, it should be noted that if the storage device is not the traditional disk then the file structure could differ significantly from both schemes mentioned here (see for example Goldschmidt[10]).

A final way in which the current Intrex differs from its possible future implementation is in its flexibility. Ultimately, as information retrieval requirements become more demanding and more diverse, the single IR center will not be able to fulfill these requirements. A Network of IR centers, each center perhaps specializing in a particular area of information for its data base will be needed. Each must then provide a variety of different types of service including SDI and retrospective searching carried out as a batch processing operation subordinate to the on-line service. In particular, search requests of the latter type should be initiated at the on-line terminal and be directed to any center or data base in the network. Indeed, the Intrex-like system of the future may be broadened far beyond the scope of simple bibliographic retrieval. It may serve as the catalyst for integrating publishing and clearing houses in a nation or world wide network where information of all kinds can be collected, retrieved, and distributed. The Intrex-like system of the future must be prepared for this network type operation and must have the evolutionary ability to adapt to further system requirements possibly unknown at the time of its design.

APPENDIX A

SUMMARY OF COMMON REFERENCES

1. Parameter Option Table (POT)

BLIP

Function: Hold blip characters
Interrogated by: AND., FSO, IN., OUT., SEARCH
Changed by: INXCON

BYTEC

Function: Count of bytes printed from catalog
Interrogated by: GETFLD
Changed by: DYNAMO, GETFLD

CATS2

Function: Name2 of catalog files
Interrogated by: DYNAMO, GETINT, QUIT
Changed by: INIVAR

CD

Function: Command line delimeters pointer
Interrogated by: AND., INIRES, SAVE
Changed by: INICON

CET

Function: End Trim-Table pointer
Interrogated by: AND., INIRES, SAVE
Changed by: INICON

CFT

Function: Front-Trim-Table pointer
Interrogated by: AND., INIRES, SAVE
Changed by: INICON

COMBF0

Function: Common buffer 0 pointer
Interrogated by: INIRES, QUIT, SAVE
Changed by: INIFIX

COMBF1

Function: Common buffer 1 pointer
Interrogated by: ATSCRN, GETBUF, GETINT, GETLIS, IFSRCH, SUMOUT, TABLE, TRETR1
Changed by: INIFIX

COMBF2

Function: Common buffer 2 pointer
Interrogated by: ANDER, IFSRCH, MOVEIT, TABLE
Changed by: INIFIX

COMBF3

Function: Common buffer 3 pointer
Interrogated by: ATSCRN, IFSRCH, TYPEIT
Changed by: INIFIX

COMBF4

Function: Common buffer 4 pointer
Interrogated by: ANDER, ATSCRN, FSO, IFSRCH, NUMBER
Changed by: INIFIX

COMBF5

Function: Common buffer 5 pointer
Interrogated by: ATSCRN, FSO, IFSRCH
Changed by: INIFIX

COMBF6

Function: Common buffer 6 pointer
Interrogated by: ANDER, CHKSAV, CONDIR, CONNAM, FSO, INIFIX, LISFIL, LISTSL, LONG, MOVEIT, NAME, NEWPT, SAVE, SHORT, USE
Changed by: INIFIX

COMLIN

Function: Command line pointer
Interrogated by: GETLIN, SAVE
Changed by: INIFIX

COMTB

Function: Command table pointer
Interrogated by: CLP, NAME
Changed by: INIFIX

CPUS

Function: CPU times array pointer
Interrogated by: MONINT
Changed by: MONINT

DFLN1

Function: Long message file name1
Interrogated by: DYNAMO, GETLIN, INFO, LONG, TYPEIT
Changed by: INITDB

DFN1

Function: Dump file name1
Interrogated by: ANDER, ATSCRN, BUFSCN, CLEANP, DELIST, FSO, NEWPT, QUIT, USE
Changed by: INIRES

DFSN1

Function: Short message file name1
Interrogated by: DYNAMO, GETLIN, INFO, SHORT, TYPEIT
Changed by: INITDB

ESCODE

Function: Escape code
Interrogated by: INIRES, OPFILE, QUIT
Changed by: DYNAMO

IFS2

Function: Inverted file name2
Interrogated by: GETLIS, IFSINT, IFSRCH, MEADIR, SEARCH
Changed by: INIVAR

INTIAD

Function: Location of level 1 interrupt routine.
Interrogated by: ININT, INTONE
Changed by: ININT

LMAP

Function: Pointer to field lengths array
Interrogated by: GETFLD
Changed by: GETFLD

MAXCHR

Function: Maximum length of output line
Interrogated by: TYPEIT
Changed by: DYNAMO, INITDB

MAXCIN

Function: Maximum number of input characters per line
Interrogated by: GETLIN
Changed by: INITDB

MAXLIN

Function: Maximum number of lines of input
Interrogated by: GETLIN
Changed by: INITDB

MFN1

Function: Monitor file name1
Interrogated by: ERRGO, MONTOR, INIRES
Changed by: INIRES

MODEG

Function: Mode used by procedure ANDER
Interrogated by: AND., NOT
Changed by: CLP, WITH

MODS

Function: Address of module call count array
Interrogated by: MONINT
Changed by: MONINT

MONBF1

Function: Location of monitor buffer 1
Interrogated by: INIDSK
Changed by: INIDSK

MONBF2

Function: Location of monitor buffer 2
Interrogated by: INIDSK
Changed by: INIDSK

MFUN1

Function: Monitor File used name1
Interrogated by: MONTIM, QUIT, SEARCH, SUMOUT
Changed by: MONTOR

MFUN2

Function: Monitor File used name2
Interrogated by: MONTIM, SUMOUT
Changed by: MONTOR

NFN1

Function: Name File name1
Interrogated by: ANDER, CONNAM, DROP, NAME, SAVE, USE
Changed by: DROP, USE

PFN1

Function: Password File name1
Interrogated by: DYNAMO, LONG, QUIT, SHORT
Changed by: INIRES

RAM

Function: Residual author mode
Interrogated by: GATP, LN
Changed by: INITDB

REAS

Function: Real time array pointer
Interrogated by: MONINT
Changed by: MONINT

RTM

Function: Ready message time pointer
Interrogated by: GETLIN
Changed by: INITDB

SFN1

Function: Save File name1
Interrogated by: LIST, SAVE
Changed by: SAVE, USE

STM

Function: System CPU time array pointer
Interrogated by: CALLIT, LISTEN, MONTOR, SUPER, TYPEIT
Changed by: INITDB

SYSNAM

Function: Intrex system name
Interrogated by: MONTOR
Changed by: DYNAMO, INIVAR

TEXTX

Function: Text access pointer
Interrogated by: FSO, GETLIN
Changed by: CLP, FSO, OUT.

TOTBLK

Function: Dump File pointer
Interrogated by: ANDER, ATSCRN, BUFSCN, FSO, GETLIS
Changed by: ANDER, DELIST, INIRES, NEWPT

TOTNAM

Function: Name File pointer
Interrogated by: ANDER, CONNAM
Changed by: ANDER, CONNAM, DROP, INIRES, USE

TOTSAV

Function: Save File pointer
Interrogated by: SAVE
Changed by: SAVE, USE

VERBOS

Function: TYPEIT mode
Interrogated by: CLP, FSO, GETLIN, INIRES, SEEMAT, TRETRI, TYPEIT
Changed by: DYNAMO, INITDB LONG, SHORT

2. System State Table (SST)

CATII

Function: Off-line output
Interrogated by: FSO, GETLIN
Changed by: DYNAMO

CLAMP

Function: Restricted user
Interrogated by: INIRES, LONG, MONITOR, QUIT, SHORT
Changed by: DYNAMO, QUIT

FSONX

Function: FSO not executed
Interrogated by: EVAL, SUPER
Changed by: EXIT, FSO, OUT., SUPER

GCE

Function: Go command exists
Interrogated by: SUPER
Changed by: EXIT, FSO, OUT., SUPER

GETFLG

Function: Interrupt in GETLIN
Interrogated by: INTONE, LISTEN
Changed by: GETLIN

HELD

Function: Intrex held up for Message File update
Interrogated by: GETLIN
Changed by: GETLIN

IBEG

Function: In begin stage
Interrogated by: CLP, GO, LISTEN, SUPER
Changed by: DYNAMO, GO

INFO1

Function: None
Interrogated by: Not Used
Changed by: INFO

INFO2

Function: None
Interrogated by: Not Used
Changed by: INFO

INFOX

Function: Info. begin performed
Interrogated by: GETLIN
Changed by: INFO, TYPEIT

INT1

Function: Interrupt level 1
Interrogated by: TYPEIT
Changed by: INTONE, TYPEIT

INT2

Function: Interrupt level 2
Interrogated by: TYPEIT
Changed by: INTTWO, TYPEIT

ISI

Function: In sign-in
Interrogated by: LISTEN
Changed by: SIGNIN, SUPER

LISAV

Function: At least one reference list named
Interrogated by: DROP
Changed by: DROP, NAME

RLIC

Function: Restored list in core
Interrogated by: AUTHOR, DELIST, RESTOR, SUBJ., TITLE
Changed by: NUMBER, RESTOR

RRLE

Function: Reference list exists
Interrogated by: SUPER
Changed by: DELIST, RESTOR, SUBJ., TITLE

SKIPS

Function: Skip sign-in
Interrogated by: SIGNIN
Changed by: DYNAMO

SPEC1

Function: Special condition 1
Interrogated by: GETFLD
Changed by: DYNAMO

SPEC2

Function: Special condition 2
Interrogated by: FSO
Changed by: DYNAMO

SNX

Function: Search not executed
Interrogated by: AUTHOR, OUT., RESTOR, SUBJ., SUPER, TITLE
Changed by: EXIT, GATP, SEARCH, SUBJ., SUPER, TITLE

TESTIT

Function: In test mode
Interrogated by: DYNAMO, SUPER
Changed by: DYNAMO

TIMES

Function: Time request active
Interrogated by: GETLIN
Changed by: TIME

3. Command List

ASF

Function: Author search form
Interrogated by: ASRCH, CLEANP, EVAL, FSO
Changed by: ACLN, GATP

DCNT

Function: Document count
Interrogated by: EVAL
Changed by: RESTOR, SEARCH

FSL

Function: Secondary search list
Interrogated by: EVAL, FSO
Changed by: FCLEAN, INITDB, IN.

ORL

Function: Output request list
Interrogated by: EVAL, FSO
Changed by: FCLEAN, INITDB, OUT.

RESUB

Function: Recordered subject search list
Interrogated by: STCLN
Changed by: CLEANP, SSRCH

RETIT

Function: Reordered title search list
Interrogated by: STCLN
Changed by: CLEANP, TSRCH

RRL

Function: Reference list
Interrogated by: AND., ASRCH, CLEANP, DELIST, DRPPTR, EVAL, FSO, SEARCH, STRCH
Changed by: ASRCH, CLEANP, DELIST. RESTOR, STRCH, USE

SSF
Function: Subject search form
Interrogated by: CLEANP, EVAL, FSO, RANGE, SSRCH
Changed by: STCLN, SUBJ.

TSF
Function: Title search form
Interrogated by: CLEANP, EVAL, FSO, TSRCH
Changed by: STCLN, TITLE

APPENDIX B
DATA BASE FORMATS

1. Catalog

The Intrex catalog uses a directory file, CATDIR INTREX, and a set of catalog files. The catalog files are named CRxxx INTREX, where xxx is a 3-digit number. For each document in the data base, CATDIR INTREX contains a one-word pointer which points to the document's catalog record within the segmented catalog files.

The first ten words of the file CATDIR INTREX contains general information about the catalog (see Table 1). Starting in word 11 of the directory, each word in position n of the directory references the document with document number n-10 (see Table 2). If a document is not in the data base, the corresponding word in the directory is zero.

Each catalog record has three separate regions. The first region is four words long and consists of fixed-length, binary encoded fields (see Table 3). The second region contains pointers to variable length fields in the third region (see Table 4). This region begins with a one-word header which specifies the length of the region. Following the header are the pointers which contain the field number in the decrement* and the byte position of the first character of the next field in the address.** The first byte of the first field is counted as position I. The character strings follow one after the other in region three. The strings are digram encoded; that is, pairs of ASCII characters are represented whenever possible by one unique nine-bit code.

* Bits 3-17 of the pointer
** Bits 21-35 of the pointer

Table 1

Format of first 10 words of CATDIR INTREX

Word	Contents
1	Highest document number in file
2	Size of catalog file in words
3	No. of catalog records in catalog
4	No. of changes made to catalog
5	Name1 of last segment
6	Name2 of catalog
7	Maximum length of catalog segment
8-10	Not used

Table 2

Format of CATDIR INTREX pointer

Word	Bits	Contents
1	0-8	Number of catalog segment
1	9-20	Length of record in words
1	21-35	Position of first word of record

Table 3

Format of Region 1 of catalog record

Word	Bits	Contents
1	0-20	Document number (Field 1)
1	21-35	Online date (Field 4)
2	0-11	Catalogers' codes (Field 3)
2	12-14	Level of approach (Field 66)
2	15-19	Language (Field 36)
2	20-24	Medium (Field 30)
2	25-30	Format (Field 31)
2	31-35	Purpose (Field 65)
3	0-9	Language of Abstract (Field 37)
3	10-15	Chosen by (Field 2)
3	19-20	Main entry pointer (Field 20)
3	21-35	Date acquired (Field 46)
4	0-35	Not used

Table 4

Format of Region 2 of catalog record

Word	Bits	Contents
1	0-17	All 1's
1	18-35	Number of words in Region 2
2	3-17	Field number
2	21-35	First byte position
3	0-35	Same as word 2

2. Inverted Files

The Author and Subject/Title Inverted Files consist of n segments. Each segment of the Inverted File contains ten sections of 432 computer words (one disk record) each. If a long list begins in Section 10, it may run over into one or more additional 432 word blocks. Each section begins with a one-word header containing a fence of octal 7's in the left half and a list offset in the right half (see Figure B1). This offset is the number of computer words into the section where the first new list begins. If no new list begins in that section (because of a long list beginning in a previous section and ending in a subsequent section), then the offset will be zero.

Each list is introduced by a three word header. The first word of this header contains an octal 7 in the left-most three bits (prefix), the number of all-zero computer words used as padding at the end of the list (BWL) in the next 15 bits (decrement), and the total number of computer words in the list (CWL) in the right 15 bits (address). The second word in the header contains the number of documents in the list (DCL) in the decrement, and the number of references in the list (RFL) in the address. The third word contains the number of characters (bytes) in the index word stem or name (BYN) in the address, a 1 (author files) or a 4 (subject title files) in bits 9-14 representing the maximum number of affix codes which may be associated with this stem/name (EWN), and the number of affixes in the list in bits 3-8 (EDS) if a subject/title file or bits 1-8 (INS) if an author file (see Figure B1).

The ASCII-coded index word or list name is packed four characters to a computer word in the words immediately following the header. Any unfilled words are left-justified and (binary) zero-padded.

Pairs of affix words follow the name. Each pair consists of a header and an ending code or set of author's initials. The ending code, found in the subject/title Inverted Files, is a left-justified 12-bit code which addresses one of the entries in the ending table residing in a file called Ending Test2. This file-table contains ASCII ending strings organized according to the length of the string from longest to shortest. The left-most four bits of the 12-bit code gives the length of the ending and, therefore, group within the table. The other eight bits provide the position of that ending within the group. Author's initials found in the affix of the author Inverted Files are strings of

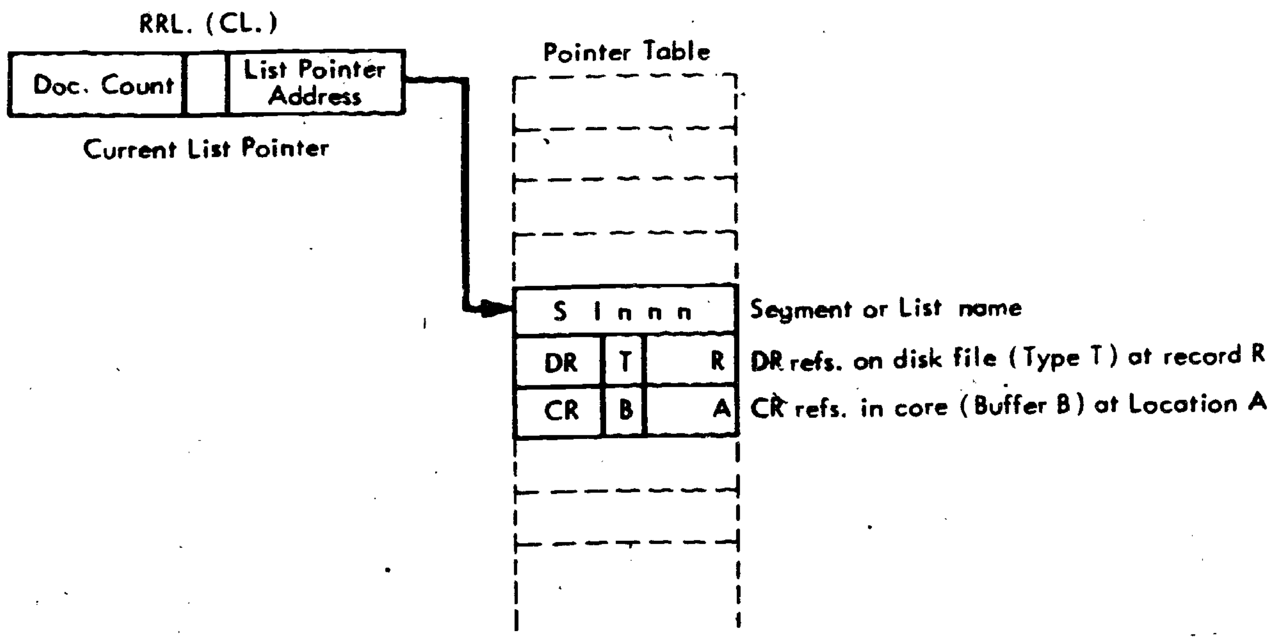


Fig. B.1 Inverted File Format

from one to four ASCII characters packed left-justified into the affix word. The affix header contains the number of such initial codes in the tag portion (bits 18-20). In subject/title files, the tag of these headers is empty. In both types of files, the address portion of the affix header words contains the number of references associated with this affix (REF), which is a subset of the entire reference list, and the decrement portion contains the number of documents in that subset of references (DCE).

Following the affix list is the list of reference words associated with the stem or name. The length of this list may range from 1 to several thousand computer words taking up many sections of the Inverted File. Each reference word contains: a document number (DN); a property code (PC); an affix number (EN or IN); and a term or author number (TN or AN). In addition to these, the subject/title references contain a weight (WT) and word number (WN). The bit positions and meaning of these components are shown in Figure B1. The affix number is used to connect the reference word to a particular affix/header word pair. For instance, a three in this component means that the third affix in the group of affixes should be used whenever this reference prompts the printing of the stem with its ending or the author name with its initials.

Inverted File lists may be split between sections only at points within the affix group or the reference list and never in the header or stem/name. To avoid the latter condition, an area of ten computer words or more must exist between the end of any list and the end of a section in order for a new list to begin in that section. When fewer than ten words are left in the section, they are (ASCII) blank-padded and the next list begins at the start of the next section. It is these blank words that are recorded in the BWL component of the three-word list header. They are also reflected in CWL (computer words in the list).

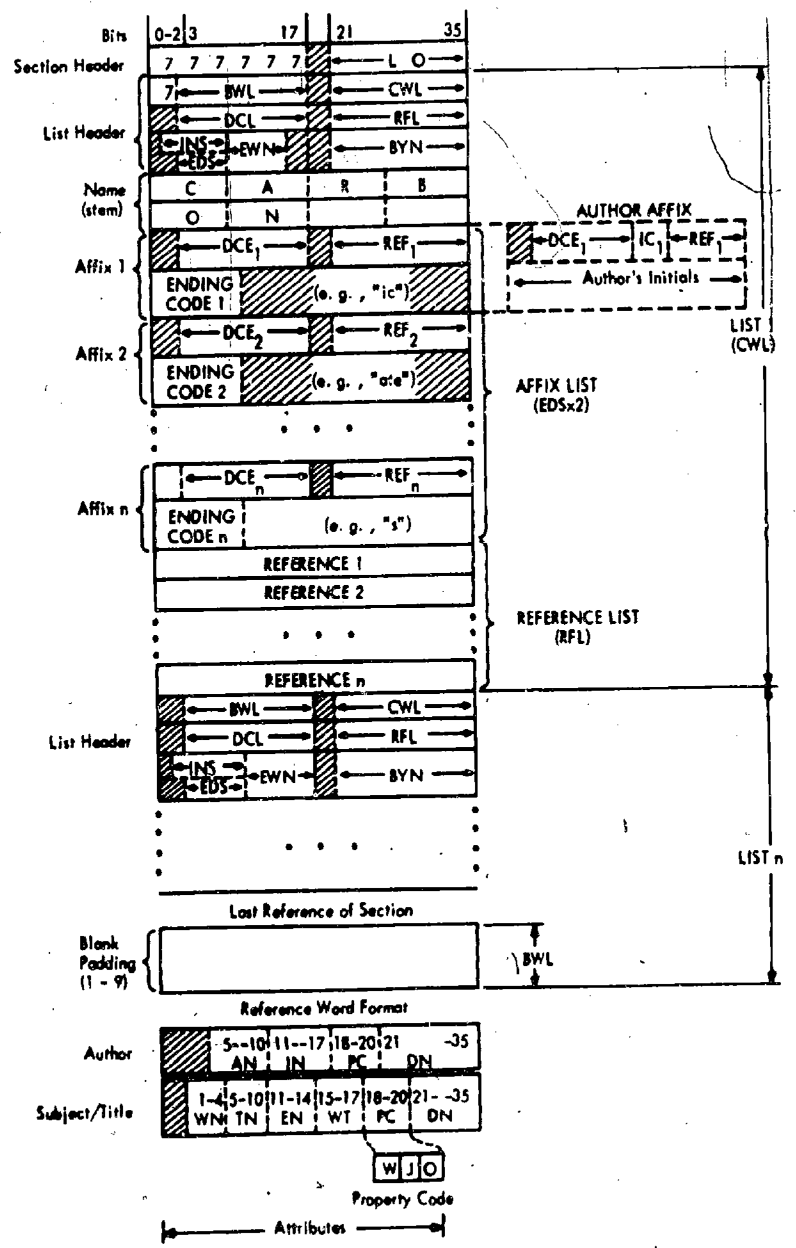
3. List Pointers

The "current list" pointer resides in the "resultant reference list" component of the Command List, RRL.(CL.), which is a single computer word whose address portion points to an entry in the table of augmented list pointers. An augmented list pointer is a group of three contiguous words containing data about a reference list. Pointers in this table may be the result of searches, Boolean operations, NAME commands, or DOCUMENT commands. At present, the table is 120 words in length and can, therefore, hold forty list pointers.

The first word of a three-part pointer is used for one of two purposes. If the list is one which has been NAMEd by the user and is, therefore, residing in the Name File, the first pointer word holds the name (in 6-bit BCD) of the list. If the list is at least partially disk-resident in some other file (such as the "Dump File or an Inverted File segment), then the first word will hold the name of that file. In any case, the first word may be thought of as the "name" portion. If the list is entirely core-resident, this word is empty.

The second word of the augmented pointer is used to point to that part of the list which resides on the disk. The address portion contains the file depth expressed in number of 432-word records at which the list begins. The decrement portion contains the length, or number of references on the disk for this list. The tag of this word contains a type code. A '1' in the tag means the list is to be found in an Inverted File segment. A '2' means the list is to be found in the Dump File, and a '4' means it is in the Name File. This code provides a means of determining how to treat the name in word one. If the entire list is core-resident, word two is empty.

Word three provides the location (address part) and length (decrement) of the core-stored portion of the list. It also contains, in the tag component, a "common buffer number" designating which of the Intrex buffers should be used for reading the disk-stored portion, if any, of the list. If the entire list is disk-resident, word three is empty. A NAMEd list pointer uses word three to hold the document count. Figure B2 shows a pointer to an Inverted File List.



LO:	List offset (position in list where first list begins)
BWL:	Blank words in list (amount of padding at end of list)
CWL:	Total computer words in list (including blanks if any)
DCL:	Number of distinct documents in list
RFL:	Number of references in list
INS:	Number of initial sets in author list
EDS:	Number of word endings in subject/title list
EWN:	Number of English words in subject/title term (always 1 at present) or number of initials in author name (always 4)
BYN:	Number of bytes (characters) in name stem (here 6)
DCE_n:	Number of distinct documents referring to the name using affix n
REF_n:	Number of references associated with the name using affix n
IC_n:	Initial count (No. of initials in this author affix)

DN:	Document number
WN:	Word number within phrase
TN:	Term number within document (subject/title references)
AN:	Author number within document (author references)
EN:	Ending number associated with reference
IN:	Initial set number associated with reference
WT:	Weight (level) of subject/title term
PC:	Property code containing the following indicators
W:	Is document whole work?
J:	Is document journal article?
O:	Does document reflect original work?

Fig. B.2 Augmented List Pointer Format

4. Fiche Direct

Fiche Direct gives the locations of the full texts of documents in the data base. The directory is ordered by document number: word n of the directory gives the location of document n. Each word contains four fields: fiche number, first frame position, last frame position and document number (see Table 5). If the full text of a document is not on microfiche, the fiche number field is zero and the first frame position has a special code indicated in Table 6.

Table 5
Fiche Direct Format

Field	Bits
Fiche Number	0-10
First Frame Position	11-16
Last Frame Position	17-22
Document Number	23-35

Table 6
Special Codes

Code (in first frame position)	Exceptions to Microfiche Text Storage
1	Microfiche text access to be available at a later date. (Filming will be done after cataloging the document.) (Access number, when known, will be added to FICHE DIRECT by update.)
2	Microfiche text access available only to individual bibliographic parts of this document. (For example, the document record is that for an entire journal.)
3	Microfiche text access will not be available. Hard copy access only. (Document text runs more than 60 pages; document typography too poor to film; document is "temporary" as say, preprint to be replaced by a permanent publication.) The last frame position contains the right-most six bits of an ASCII letter code for the location of the guaranteed hard copy access. If this location is in a library whose name has an established code (see field II in the manual) that code is used. If this location is a shelf adjacent to the Intrex display console, the code letter "i" is entered.

APPENDIX C

THE INTREX ENVIRONMENT

1. The Time-Sharing System:

The Compatible Time-Sharing System (CTSS)[15] was developed at MIT by Project MAC under the sponsorship of the National Science Foundation the Office of Naval Research, and the Ford Foundation. It was one of the pioneering efforts in the time-sharing field, becoming operational in 1963, and through continuous growth and improvement it has reached a high level of utility and reliability. CTSS is implemented on an IBM 7094 computer using two 32K word* units of core memory. One memory unit (referred to as A-core) holds the operating system. This operating system handles I/O, scheduling, the swapping of user programs, and general job monitoring. The other memory unit (referred to as B-core) is reserved for the user program which is currently being run. No multiprogramming or paging capability exists on CTSS so only one user program can reside in B-core at a given time. User programs are swapped in and out when an I/O transfer is called for or when the time slice allocated by the supervisor scheduling algorithm has elapsed. When a user's program is swapped out of core, its core-image is dumped onto the high speed drum from which it can be reloaded when its turn arrives again. The computer time used for swapping is tabulated by the supervisor and a share of it is assessed each user, together with the processing time used.

The user interacts with CTSS through commands typed on the terminal keyboard. Most commands have one or more arguments specifying how or upon which entity the command is to be implemented. For example, when the command is to begin execution of a user's previously loaded but disk resident program, the command is RESUME or R and the first argument after the word RESUME is the name of the program. Other (optional) arguments may follow the program name and these are read and used by the program as it begins operation. Thus, in the command, RESUME INTREX SHORT HOLD, the command RESUME will cause the program file named INTREX SAVED

*A word on the 7094 is 36 bits plus parity. Words have the following format:

Pre-fix	Decrement	Tag	Address
0-2	3-17	18-20	21-35

to be brought into B-core and executed. The program INTREX will then read the arguments SHORT and HOLD and use them to set the appropriate program mode switches. INTREX or any other program once started can be terminated immediately by pushing the "attention" or "quit" button twice in succession.** Other CTSS commands permit a user to create, edit, compile, or assemble programs and perform some other functions described below.

One of the most powerful and useful characteristics of the CTSS architecture is its method of storing, addressing, sharing, and generally managing files. A file is a collection of disk-stored or drum-stored symbols which may be a source program, an object program, printable text, or set of data. The actual location of files on physical storage devices is handled by the CTSS supervisor and need not concern the system user, who always addresses the files by a symbolic name. Each file has two names, the second of which usually classifies it as a particular type. For example; a file whose last name is SAVED is assumed by CTSS to be an executable program. Furthermore, any executable program must be given the last name SAVED before it can be executed at any time other than immediately after loading. Files whose last names are BSS are assumed to be relocatable binary (object) programs. Only these can be loaded together to become an executable program. Files whose last names are ALGOL are assumed to be source programs in the AED language and only these can be compiled by the AED compiler into an object program (See next section).

File names are kept by CTSS in a "User's File Directory" (UFD), which may be accessible to the individual programmer or shared by several users. A UFD which is sharable is called a Common File (or COMFIL). A CTSS account may consist of one or more "problem numbers", each maintaining its own set of Common Files and/or individual UFD's. A user upon "logging in" to CTSS enters his "home UFD". He then may switch into other UFD's where permission to do so has been established or merely share programs in other UFD's (again when permission has been established). The Intrex software system currently exists in COMFIL 4 of problem number T289.

** However see Section 3.1.10.3 in regard to INTREX.

System components (source files etc.) are stored in other Common Files. Through the sharing of programs many users who have individual accounts on CTSS can use the Intrex retrieval programs without having to maintain copies of the programs in their own directories. Such programs can be made available to users in a read-only mode to protect the programs from change.

2. The AED Programming Language[13]

AED (Automated Engineering Design) is a high level source language based on ALGOL-60 and developed under the direction of Douglas T. Ross at the Electronic Sytems Laboratory of MIT. AED includes, besides the conventional features of ALGOL, provision for dynamic storage allocation and list processing. Also included but not built directly into the language are procedures for string and character manipulation and several other features. AED was initially implemented on the IBM 709 but was brought to CTSS in 1963 and "grew up" there. Currently the language is also available on the IBM 360/67 and other third generation machines.

Intrex software employs several important features of the AED language and these greatly facilitated the integration of system components. Perhaps the most useful feature of all was the flexibility of subroutine calls and linkages such as the ability to pass a variable length string of arguments to the routine being called and to chain through an unlimited number of routine levels and back without restrictions. Other features which played a vital role in Intrex construction are the free storage package, the COMMON variable area, the ability to conveniently pack words and refer to the packed components, and the ability to insert machine language instructions where AED language statements are inadequate or inconvenient to perform the desired function.

Source programs written in AED are entered into the CTSS file system via input/edit programs such as EDL or QED[15] where each character is represented by a 6-bit BCD code. The implementation of AED on CTSS is via the command TAED which calls the compiler into action upon the file whose first name is specified as an argument to the command and whose last name is ALGOL. Other arguments may be appended to the command to designate special modes or to generate additional output files (e.g., a symbol table).

APPENDIX D
MESSAGE TEXT

1. "Long" Message Text

```
/exit/ *error*
/mtlab1/ *signin*
/mtlab2/ *sign2*
/mtlab3/ *clp*
/mtlab4/ *fso*
/mtlab5/ *eval*
/mtlab6/ *quit*
/mtlab7/ *search*
/mtlab8/ *int1*
/mtlab9/ *int2*
/op1/ *A search on your request*
/op2/ *found*
/op3a/ *documents.*
/op3b/ *document.*
/op4a/ *The catalog fields*
/op4b/ *TITLE, AUTHOR, LOCATION*
/op5a/ *for those documents*
/op5b/ *for that document*
/op6/ *which also match your field restrictions RESTRICT *
/op7/ *will be output*
/op8a/ *now.*
/op8b/ *when you type


#016#016o#017


(for output).*
/op9/ *You may terminate this output at any time by hitting the ATTN key ONCE.*
/op9a/ *    Otherwise, you may make other output requests (for information see Part 8 of the Guide)
or change your field restriction (see Part 9.5) or make another request of
Intrex (see Part 1)*
/op10/ *The results of the COUNT command showing how many documents matched each word in your request
is given below.*
$op10.5$ *You may make a new search by dropping some of your search words or chosing
other search words, or you may make some other/
request of Intrex (see Part 1).*
/op11/ *STANDARD*
/op12/ *MATCH*
/op13/ */RESTRICT*
/op14/ *SUBJECT*
/op14a/ *RANGE*
/op15/ *TITLE*
/op16/ *AUTHOR*
/op17/ **
/op90/ *TEXT*
/op18/ *You may now make a new search using other terms or make other requests of Intrex
(see Part 1 of Guide or type #016info 1#017).*
$sin1$ *Greetings! This is Intrex.  Please log in by typing the word LOG followed by a space and your name and
address as in the following example:
```

```
¢016loq smith, r j;mit 13-5251;ext 7234¢017


Note that your log in statement should end with a carriage return.*
$sin2a$ *Welcome to Intrex M.*
$sin2$ *  It you already know how to use Intrex, you may go ahead and type
in commands. (Remember, each command ends in a carriage return.)
Otherwise, for information on how to make simple searches of the
catalog, type


¢016info 2¢017


or,to see the Table of Contents (Part 1) of Intrex Guide
which will direct you to other parts of the Guide explaining how to
make more detailed searches, type


¢016¢016info 1¢017¢017
*
$exmes$ * We would appreciate your comments on the Intrex system.  For information
on how to make comments see Part 13 of Guide or type


¢016¢016info 13¢017¢017


You may also make additional service requests of the Intrex consultant.
  If you do not wish to make any other comments or requests, type


¢016quit¢017


*
$outmes$ *Thank you for using Intrex.*
$passer$ *Error in writing password file. No automatic resumption of Intrex.*
$passok$ *Password received--*
$begmes$ *Please type the word BEGIN followed by a carriage return.*
$beqer1$ *Intrex could not understand your log statement.*
$beger2$ *Please log in by typing the
word LOG followed by a space and your name and
address as in the following example:

log smith,r j;mit 13-5251;ext 7234

Note that your log statement should end with a carriage return.*
$beger3$ *Intrex could not find your name in your log statement.*
/redmes/ *READY*
$sin3$ *You may now look at any part of the Guide or use any other command.*
$sin4$ *To see Table of Contents for Intrex 1 Guide and how to use
the Guide on line, type
```

```
#016info 1#017

Otherwise, you may now make simple searches or use any other command.*
$infoer$ *is not a valid argument to the 'INFO' command.*
$infotr$ *This section of the on-line Guide has been truncated because of its length.
See printed version if you care to read the entire section.*
/fso1/ *Search and*
/fso2a/ *Output completed. You may now see other
catalog information from*
/fso2b/ *this document*
/fso2d1/ *these*
/fso2d2/ *documents*
/fso2c/ *by making an OUTPUT request (for information on how to
do this, see Part 8 of the Guide or type #016info 8#017). *
/fso2e/ *You may also select a subset of these documents by
making a RESTRICT request (see Part 9.5).*
$fso4$ *No documents found.  You may make a new search (see
Part 2 of Guide, or type #016info 2#017), or make other requests
(see Part 1).*
/fso5/ *     Otherwise, you may make a new search (see Part 2)
or make other requests (see Part 1).*
$fso6$ *Your last active list has been retained.*
$anor0$ *WITHing*
$dn1$ *The number of documents you now have on your current list is*
$dn2$ * To see output*
$dn3$ *you may use the OUTPUT command (see Part 8 of the Guide for instructions).*
$anor1$ *The list resulting from*
$anor2$ *ANDing*
$anor3$ *ORing*
$anor4$ **
$anor4b$ *contains*
$anor5$ *NOTing*
$anerr1$ *You have no current list on which to perform Boolean operations.*
$con1$ *    There will be a slight delay while Intrex condenses your Named-List File. Please stand by.*
$delay$ *Intrex is going dormant for a few moments to allow a system file to
be updated. You will be told to proceed when Intrex is ready for your next request. Please stand by.*
$proced$ *You may now proceed to issue requests to Intrex as soon as the next READY
or R appears.*
$nam0$ *No*
$nam1$ *Your list table is full. You must drop one or more lists before re-issuing your command.*
names before re-issuing your NAME command.*
$nam2$ *has not been NAMEd.*
$nam3$ *You have not provided the name of a NAMEd list.*
$nam4$ *NAMEd lists currently being held.*
$nam5$ *Your list name is ambiguous with*
$nam6$ *the Intrex command,*
$nam7$ *a previously NAMEd list.*
$nam8$ *Please use another name.*
$nam9$ *ALL is a restricted word for naming lists.*
$nam10$ *Your current list has already been named and cannot be named again.*
$nam11$ *You have not given a name for your list.*
$nam12$ *You have no current list. Your NAME command cannot be processed.*
$nam13$ *List names must consist of only one word.*
$nameok$ *is now the name of your current list.*
```

```
$sav2$ *The name you have assigned to your SAVE file is already in use.
Please repeat your SAVE FILE request using another name.*
$sav3$ *You have not provided a name for your SAVE file.*
$sav4$ *Your current SAVE file is full. You must assign a new name
 via the SAVE FILE command before saving any more lists.*
$sav5$ *SAVE files currently being held on file.*
$sav6$ *Lists in file*
$sav7$ *is not found to be stored on disk and is being deleted from the
SAVE file directory.*
$sav8$ *The word FILE may not be used as the name of a SAVEd file.*
$usem1$ *has become your current NAMEd List file and SAVE file.
The list names in this file may now be restored to active status by typing
their names, which are;*
$usem2$ *has become your current SAVE file and will accept SAVEd lists.
Your current NAMEd lists (if any) are retained and may now be SAVEd if
and when desired.*
$liser1$ *is not a SAVE file name.*
$user1$ *You have not provided the name of a SAVE file.*
/doc5/ * DOES NOT MATCH THIS RESTRICTION*
$doc6$ *This request is valid only for references originated by SUBJECT searches.*
$out.3a$ *See Part 8 of Guide for details on correct usage of
OUTPUT command. Please rephrase your request.*
$out.4$ *The OUTPUT command accepts only 10
fields. Your first 10 fields will be output.  You may output the
remainder on a subsequent request. *
$out.5$ *To see text you must say "
#016output fiche#017 and get the fiche location of the text.*
$super$ *You have no current active list for which to provide output.
You must perform a new search or restore a saved (NAMEd) list.*
$in.1$ *Only 10 RESTRICT restrictions allowed on a search.
Please begin a new search.*
$in.2$ *You have not fully specified your RESTRICT command.
See Part 9.5 of Guide for details on correct use of RESTRICT
command.  Please rephrase your request.*
$in.3$ *is not a legal field designation. Check for typing errors. See Part 15 of
Guide for full list of types of catalog information.*
$in.6$ *You cannot do a RESTRICT search on field *
$in.3a$ *See Part 9.5 of Guide for details on correct usage of
RESTRICT command.  Please rephrase your request.*
$sm01$ *WORD(STEM-ENDING)    NO. OF DOCUMENTS THAT MATCH
                          THIS STEM   ALL STEMS SO FAR*
/qs01/  *          (TITLE) *
$clp3$ *is not a legal command name.  Check for typing errors.  See Part 6.2 of Guide
for full list of commands.

Please rephrase your request.*
$clp1$ *System error attempting to interpret command line.*
$clp2$ *Please break up your request into requests not exceeding 200 characters in length.*
$clp4$ *Your command could not be understood.*
$loerr0$ *zero (0)*
$loerr1$ *one (1) *
$loerr2$ *You have typed the letter*
$loerr3$ *in place of a*
$loerr4$ *in the command argument*
$loerr5$ *  Your command will be processed as if a*
$loerr6$ *had been typed.  Please observe this distinction.  Failure to do so may cause
```

```
matching difficulties.*
$loerro$ *¢016o¢017*
$loerr1$ *¢0161¢017*
$raerr1$ *is not a legal RANGE.  Check for typing
errors.  See Part 9.2 of the Guide for details of
correct usage of the RANGE command.*

$raerr2$ *The RANGE command may only be used along
with a SUBJECT search as explained in Part 9.2 of the
Guide.

Please rephrase your request.*

$raerr3$ *You may use the RANGE command only once
on each SUBJECT search as explained in Part 9.2 of
the Guide.
Please rephrase your request.*
$dnerr1$ *is an improper document number and has been ignored in processing your command.*
$dnerr2$ *You have not included any (legitimate) document numbers in your command.*
$ioerr1$ *Error code*
$ioerr2$ *in call to*
$ioerr3$ *at location*
$ioerr4$ *involving file,*
$inter1$ *You have used more words in your search request than the system can handle.  Intrex will now search on the
first seven significant words you have given.*
$inter2$ *The AUTHOR command can accept a maximum of three initials.  All others are ignored.*
$inter3$ *Your search term contained no searchable words. Please review your search request.*
$inter4$ *Your search term contained an odd number of asterisks. Asterisks must be used
in pairs to set off a special code or symbol.*
/text4/ *      Fiche No.  First Frame   Last Frame*
/text1/ *This document is not yet available from the text-access
subsystem.  You may see a hard copy by asking a member of the
Intrex staff for it.*
/text2/ *Text is available only for the individual parts of this document (that
is, for articles or chapters) which were separately documented.*
/text3/ *Request a hard copy of text from a member of the Intrex staff.*
/text5a/ *Hard copy is found at library with code name *
/text5b/ *.  See Part 15.11 of the Guide for explanation of code.*
$text6$ *This document is not available at the console.  You may
obtain a copy of it from the Microfilm Service Area at the M.I.T. Engineering
Library (TN 4-6900, x3129).


          Please take the following fiche number with you:*
/empmes/ *There is no data in this field for this document*
$get1$ *The catalog record for this document can not be retrieved
at this time. Error code = *
$get2$ *INTREX is unable to print this field. Try using short mode
(type '¢016short¢017').*
/text5/ *The fiche for this document is not available*
$warn$ *NOTICE!
For more complete information see hard-copy Guide.*
$outlaw$ *The number you have used to log in is available only for use for the Intrex
retrieval system at certain times and locations. If you want to use CTSS for other than
Intrex retrieval, you must use another entry number. If you want to use the Intrex retrieval
system, please come to the Engineering Library between 11 AM and 4 PM
```

```
on weekdays or, for special hours or locations, contact Richard Marcus at MIT 35-406, ext. 2340.*
$serr$ *An error in the computer files caused zero documents to be retrieved
in searching on the word*
$serr2$ *Avoid using this word for searching if error re-occurs.*
/f30.01/ *Conventional*
/f31.34/ *Professional journal article*
$f31.33$ *Professional Serial article*
/f31.35/ *Professional letters journal article*
/f31.36/ *Conference proceedings article*
/f31.23/ *Conference proceedings*
$f31.37$ *Trade journal*
$f31.38$ *Mass media magazine*
$f31.44$ *Abstract*
/f31.53/ *Report*
$f31.05$ *Bibliography*
/f36.05/ *English*
$f36.06$ *French*
$f36.07$ *German*
$f36.19$ *Russian*
/f36.00/ **
/f02.0a/ *Radiofrequency, microwave and optical spectroscopy of
liquids and solids (Professor Benedek)*
/f02.0b/ *High temperature metallurgy (Professor Grant)*
/f02.0c/ *Microwave and quantum magnetics (Professor Epstein)*
/f02.0d/ *Casting and solidification (Professor Flemings)*
/f02.0e/ *Structural materials (Professor McGarry)*
/f02.0f/ *Transportation*
$f02.0g$ *Hybrid computing structures (Professor Dertouzos)*
/f02.01/ *Librarian*
/f02.02/ *Faculty*
/f02.03/ *Researcher*
/f02.04/ *Graduate student*
/f66.01/ *Professional*
/f66.04/ *Undergraduate*
$f66.07$ *Layman*
/f65.20/ *Theoretical*
/f65.16/ *Proposal*
/f65.19/ *Essay*
/f65.05/ *Experimental*
/f65.04/ *Report on a development or application*
/f65.02/ *Theoretical and experimental*
/f65.14/ *Non-critical review*
/f65.18/ *Critical review*
/f20.01/ *Personal author*
$f20.02$ *Corporate author*
$f20.03$ *Title*
```

2. "Short" Message Text

```
/exit/ *error*
/mtlab1/ *signin*
/mtlab2/ *sign2*
/mtlab3/ *clp*
/mtlab4/ *fso*
/mtlab5/ *eval*
/mtlab6/ *quit*
/mtlab7/ *search*
/mtlab8/ *int1*
/mtlab9/ *int2*
/op1/**
/op2/*found:*
/op3a/*docs*
/op3b/*doc*
/op4a/*o:*
/op4b/ *NORMAL*
/op5a/**
/op5b/**
/op6/ */R:*
/op7/ **
/op8a/ **
/op8b/ **
/op11/*STANDARD*
/op12/*MATCH*
/op13/*/R:*
/op14/ *S:*
/op14a/ *RA:*
/op15/ *T:*
/op16/ *A:*
/op17/ **
/op90/ *90*
/sin1/ *Please log in.*
/sin2a/ *Welcome M.*
/sin2/ T*
/sin3/ *
*
/sin4/ *
*
/redmes/ *R*
/exmes/ *Please comment or quit.*
/outmes/ *Thank you for using Intrex.*
$passer$ *Error in writing password file. No automatic resumption of Intrex.*
$passok$ *Password received--*
$begmes$ *Please type the word BEGIN followed by a carriage return.*
$beger1$ *Intrex could not understand your log statement.*
$beger2$ *Please log in by typing
the word LOG followed by your name and address.*
$beger3$ *Intrex could not find your name in your log statement.*
$infoer$ *is not a valid argument to the 'INFO' command*
$intotr$ *This section of the on-line Guide has been truncated because of its length.
```

```
See printed version if you care to read the entire section.*
/fso1/ **
/tso2a/ **
/fso2b/ **
/fso2c/ **
/fso2d1/ **
/tso2d2/ **
/tso2e/ **
/fso4/ *No documents found.*
/fso5/ **
$fso6$ *Your last active list has been retained.*
$anor0$ *WITHing*
$anor1$ *The list resulting from*
$anor2$ *ANDing*
$anor3$ *ORing*
$anor4$ **
$anor4b$ *contains*
$anor5$ *NOTing*
$anerr1$ *You have no current list on which to perform Boolean operations.*
$con1$ *    There will be a slight delay while Intrex condenses your Named-List File. Please stand by.*
$delay$ *Intrex is going dormant for a few moments to allow a system file to be updated.
You will be told to proceed when Intrex is ready for your next request. Please stand by.*
$proced$ *You may now proceed to issue requests to Intrex as soon as the next
READY or ? appears.*
$nam0$ *No*
$nam1$ *Your list table is full. You must drop one or more lists before re-issuing your command.*
$nam2$ *has not been NAMEd.*
$nam3$ *You have not provided the name of a NAMEd list.*
$nam4$ *NAMEd lists currently being held.*
$nam5$ *Your list name is ambiguous with*
$nam6$ *the Intrex command,*
$nam7$ *a previously NAMEd list.*
$nam8$ *Please use another name.*
$nam9$ *ALL is a restricted word for naming lists.*
$nam10$ *Your current list has already been named and cannot be named again.*
$nam11$ *You have not given a name for your list.*
$nam12$ *You have no current list. Your NAME command cannot be processed.*
$nam13$ *List names must consist of only one word.*
$nameok$ *is now the name of your current list.*
$sav2$ *The name you have assigned to your SAVE file is already in use.
Please repeat your SAVE FILE request using another name.*
$sav3$ *You have not provided a name for your SAVE file.*
$sav4$ *Your current SAVE file is full. You must assign a new file name via
the SAVE FILE command before saving any more lists.*
$sav5$ *SAVE files currently being held on file.*
$sav6$ *Lists in file*
$sav7$ *is not found to be stored on disk and is being deleted from the
SAVE file directory.*
$sav8$ *The word FILE may not be used as the name of a SAVEd file.*
$usem1$ *has become your current Named-List file and SAVE file.
The list names in this file may now be restored to active status by typing their
names, which are;*
$usem2$ *has become your current SAVE file and will accept SAVEd lists.
Your current NAMEd lists (if any) are retained and may now be SAVEd if
and when desired.*
$user1$ *You have not provided the name of a SAVE file.*
```

```
$liser1$ *is not a SAVE file name.*
$liser2$ *You have not provided the name of a SAVE file.*
$in.1$ *Only 10 RESTRICT restrictions allowed on a search. Please begin
a new search.*
$in.2$ *You have not fully specified your RESTRICT command. See Part 9.5 of Guide for
details on correct use of RESTRICT command.  Please rephrase your request.*
$in.3$ *is not a legal designation. Check for typing errors. See Part 15 of Guide for full
list of types of catalog information.*
$in.6$ *You cannot do a RESTRICT search on field *
$in.3a$ *See Part 9.5 of Guide for details on correct use of RESTRICT command. Please
rephrase your request.*
$out.3a$ *See Part 8 of Guide for details on correct use of OUTPUT command.
Please rephrase your request.*
$out.4$ *The OUTPUT command accepts only 10 fields.*
$out.5$ *To see text you must say ¢016output fiche¢017 and
get the fiche location of the text.
You may output the remainder on a subsequent request.*
$super$ *    You have no current active list for which to provide output.
You must perform a new search or restore a saved (NAMED) list.*
/sm01/ **
/qs01/ *          (TITLE) *
$clp3$ *is not a legal command name. Check for typing errors.  See Part 6.2 of
Guide for full list of commands.

Please rephrase your request.*
$clp1$ *System error attempting to interpret command line.*
$clp2$ *Please break up your request into requests not exceeding 200 characters in length.*
$clp4$ *Your command could not be understood.*
$loerr0$ *zero (0)*
$loerr1$ *one (1)*
$loerr2$ *You have typed the letter*
$loerr3$ *in place of a*
$loerr4$ *in the command argument*
$loerr5$ *  Your command will be processed as if a*
$loerr6$ *had been typed.  Please observe this distinction.  Failure to do so may cause
matching difficulties.*
$loerro$ *¢016o¢017*
$loerrl$ *¢016l¢017*

$raerr1$ *is not a legal RANGE.  Check for typing
errors.  See Part 9.2 of the Guide for details of
correct usage of the RANGE command.*

$raerr2$ *The RANGE command may only be used along
with a SUBJECT search as explained in Part 9.2 of the
Guide.

Please rephrase your request.*

$raerr3$ *You may use the RANGE command only once
on each SUBJECT search as explained in Part 9.2 of
the Guide.

Please rephrase your request.*
$dnerr1$ *is an improper document number and has been ignored in processing your command.*
$dnerr2$ *You have not included any (legitimate) document numbers in your command.*
```

```
$ioerr1$ *Error code*
$ioerr2$ *in call to*
$ioerr3$ *at location*
$ioerr4$ *involving file,*
$inter1$ *You have used more words in your search request than the system can handle.  Intrex will now search on the
first seven significant words you have given.*
$inter2$ *The AUTHOR command can accept a maximum of three initials.  All others are ignored.*
$inter3$ *Your search term contained no searchable words. Please review your search request.*
$inter4$ *Asterisks in search terms must appear in pairs to offset special character symbols.
Your search command cannot be processed.*
/text4/ *      Fiche No.  First Frame   Last Frame*
/text1/ *This document is not yet available from the text-access subsystem.  You
may see a hard copy by asking a member of the Intrex staff for it.*
/text2/ *Text is available only for the individual parts of this document (that is, for
articles or chapters) which were separately documented.*
/text3/ *Request a hard copy of text from a member of the Intrex staff.*
/text5a/ *Hard copy is found at library with code name *
/text5b/ *.  See Part15.11 of the Guide for explanation of code.*
$text6$ * This document is not available at the console.  You may obtain a copy
of it from the Microform Service Area at the M.I.T. Engineering Library
(UN 4-6900, x3129).  Please bring the following
fiche number with you:*
$serr$ *An error in the computer files caused zero documents to be retrieved
in searching on the word*
$serr2$ *Avoid using this word for searching if error re-occurs.*
$text5$ *Fiche not available*
/empmes/ *no data*
$warn$ **
$outlaw$ *The number you have used to log in is available only for use for the
Intrex retrieval system at certain times and locations. If you want to use CTSS for other
than Intrex retrieval, you must use another entry number. If you want to use the Intrex
retrieval system, please come to the Engineering Libary between 11 AM and 4 PM on
weekdays or, for special hours or locations, contact Richard Marcus at MIT 35-406, ext. 2340.*
$get1$ *The catalog record for this document can not be retrieved at
this time.  Error code = *
$get2$ *INTREX is unable to print this field*
/doc5/ * NO MATCH*
/doc6/ *MATCH invalid for this request*
/f30.01/ *Conventional*
$f31.33$ *Professional Serial article*
/f31.34/ *Professional journal article*
/f31.35/ *Professional letters journal article*
/f31.36/ *Conference proceedings article*
/f31.23/ *Conference proceedings*
$f31.37$ *Trade journal*
$f31.38$ *Mass media magazine*
$f31.44$ *Abstract*
/f31.53/ *Report*
$f31.05$ *Bibliography*
/f36.05/ *English*
$f36.06$ *French*
$f36.07$ *German*
$f66.01$ *Professional*
/f66.04/ *Undergraduate*
$f66.07$ *Layman*
/f65.20/ *Theoretical*
```

```
/f65.16/ *Proposal*
/f65.19/ *Essay*
/f65.05/ *Experimental*
/f65.04/ *Report on a development or application*
/f65.02/ *Theoretical and experimental*
/f65.14/ *Non-critical review*
/f65.18/ *Critical review*
/f20.01/ *Personal author*
$f20.02$ *Corporate author*
$f20.03$ *Title*
   /f36.19/ *Russian*
/f36.00/ **
/f02.0a/ *Radiofrequency, microwave and optical spectroscopy of
liquids and solids (Professor Benedek)*
/f02.0b/ *High temperature metallurgy (Professor Grant)*
/f02.0c/ *Microwave and quantum magnetics (Professor Epstein)*
/f02.0d/ *Casting and solidification (Professor Flemings)*
/f02.0e/ *Structural materials (Professor McGarry)*
/f02.0f/ *Transportation*
$f02.0g$ *Hybrid computing structures (Professor Dertouzos)*
/f02.01/ *Librarian*
/f02.02/ *Faculty*
/f02.03/ *Researcher*
/f02.04/ *Graduate student*
```

APPENDIX E

SUBROUTINE LINKAGES

SUBROUTINE	CALLED BY:
ANDER	AND., NAME, SEARCH
AND.	CLP*
ASCINT	IN., NUMBER, OUT., RANGE, TABLE
ASCIT6	MONTOR
ASCITC	BCDASC, LOCMES, PREP,.C.ASC
ASCTSS	CTSIT, CTSIT6, TRASH
ASIDE	GETLIN, LISTEN, MONTIM, MONTOR, PUTS, SEARCH, SUMOUT, SUPER, TYPEIT, WRT
ASSET	ASIDE
ATLCLN	STCLN*
ATSCRN	SEARCH
AUTHOR	CLP*
BCDASC	DROP, ERRGO, IFSRCH, INITYP, LISFIL, LIST, LISTSL, MONTOR, QUIT, SUMOUT, USE
BCD[illegible]C	PREP, SYSGEN
BFCLOS	ERRGO, INIDSK, PREP, REND, TABLE
BFOPEN	INIDSK, PREP, REND, TABLE
BFREAD	PREP, REND, TABLE
BFWRIT	PUTS, SUMOUT
BUFFER	BFOPEN, CHKSAV, CONDIR, CONNAM, INIRES, LISFIL, LISTSL, LONG, MOVEIT, NAME, QUIT, SAVE, SHORT, SUMOUT, SYSGEN, USE
BUFSCN	ATSCRN, ANDER*
BZEL	CHKNAM, QUIT
CALLIT	ANDER, CLEANP, CLP, DROP, INIFIX, INITDB, LIST, NUMBER, RESTOR, SUPER, USE, BUFSCN, ATLCLN
CHFILE	SYSGEN

* Indicates procedure makes call via CALLIT.

SUBROUTINE	CALLED BY:
CHKNAM	AND., NAME, SAVE
CHKNUM	ASCINT, INFO
CHKSAV	SAVE, USE,* LIST *
CHNCOM	IFSINT, QUIT, REND, SYSGEN
CLEAN	AUTHOR, SUBJ., TITLE, NUMBER,* RESTOR*
CLFILE	GETLIN
CLOSE	ANDER, CLFILE, CHKSAV, CONDIR, CONNAM, DYNAMO, FSO, GETINT, IFSINT, IFSRCH, INIRES, INIVAR, LISFIL, LISTSL, LONG, NAME, QUIT, SAVE, SEARCH, SHORT, SUMOUT, SYSGEN, TIME, USE
CLP	SUPER
CNTLOC	FREC, FREE, FRET, FREZ
COMARG	DYNAMO, INIVAR
COMENT	CLP*
COMPAR	SPCTRN, TABLK
COMPUL	IFSRCH, IN., MONTOR, OUT., RANGE, SPCTRN, TABLK
CONDIR	LISTSL, USE*
CONNAM	DROP
COPY	AUTHOR, EVAL, GETEND, GETFLD, IN., NEXITM, SEEMAT, STEM, S.T., TABLE, TRANS, TYPEIT
CTSIT6	AND., CHKNAM, INFO, QUIT, SIGNIN
DEC1	NEXITM, PUTS, STEM, TRANS, TYPEIT
DEFBC	GETFLD, IFSRCH, INIRES, SEARCH
DELFIL	CONNAM, DROP, SUMOUT, SYSGEN, USE
DELIST	AUTHOR, IFSRCH, NUMBER, RESTOR, SUBJ., TITLE
DERBC	TRETRI
DIST	EVAL, FSO, SPCTRN
DNSORT	NUMBER
DORMNT	ANDER, CLP, ERRGO, FREC, FREE, FRET, FREZ, PREP, PUTS, QUIT, SHORT, SUPER, TABLE, TYPASH, TYPEIT

SUBROUTINE	CALLED BY:
DROP	CLP*
DRPPTR	AND., NAME, DELIST
DYNAMO	SUPER*
ENDTAB	REND
ERRGO	IFSRCH
EVAL	SUPER*
EXIT	CLP*
FAPDBG	CLP*
FCLEAN	CLEANP, DELIST, NEWPT
FERRTN	SETRTN, SYSGEN
FIELDS	FSO
FILCNT	CHKSAV, CLEANP, CONDIR, CONNAM, GETLIN, IFSINT, INIRES, LISFIL, LISTSL, PREP, REND, TABLE
FIND	NEXITM, SPETRN
FLDNAM	EVAL, FSO, IN., OUT
FRALG	INIAUT, INICON, INIEVL, INIFIX, INIFLD, INIOUT, INIRNG, INIS.T, MONINT, PREP, SIGNIN, SYSGEN, TABLE
FREE	AUTHOR, CONNAM, GETFLD, IFSINT, INIFIX, , INITYP, INIVAR, IN., SYSGEN, TABLE, TYPASH
FRER	CONNAM, GETFLD, TYPASH
FRET	ASCITC, CHKNAM, CLP, CONNAM, EVAL, FCLEAN, FRALG, GETFLD, GETINT, GETLIN, IFSINT, IFSRCH, INFO, INIFIX, , INITYP, NEXITM, NUMBER, PREP, QUIT, RANGE, REND, SUBJ., S.T., TABLK, TITLE, TYPASH, TYPEIT, .C.ASC
FREZ	AUTHOR, ASCIT, ASCITC, ASCIT6, BUFSCN, CLP, CTSIT, CTSIT6, EVAL, GETFLD, GETLIN, IFSRCH, INIPOT, INIS.T, INITDB, MONINT, NEXITM, NUMBER, PREP, RANGE, REND, SEARCH, SUBJ., S.T., TABLE, TABLK, TITLE
FRESET	SYSGEN
FRMRAL	RANGE
FSIZE	SUPER

SUBROUTINE	CALLED BY
FSO	SUPER*
FSOCLN	FSO, LISTEN*
FSTATE	FILCNT, GETLIS, INIDSK, INITYP, PUTS, SEARCH, SYSGEN, USE
FWAIT	ANDER, BFOPEN, BFWRIT, GETLIS
GET	ASCINT, AUTHOR, CAPASC, CHKNUM, COMPAR, COMPUL, CTSIT, CTSIT6, FIND, GETLIN, IFSRCH, IN., NAM5, NEXITM, OUT., SIGNIN, STEM, TABLK, TYPEIT, .C.ASC
GET6	ASCIT, ASCITC, ASCIT6, PREP
GET12	GETLIN
GETBRK	LISTEN, PUTS, TYPEIT
GETCOM	COMARG, SYSGEN
GETEND	EVAL
GETFLD	FSO
GETIME	GETLIN, GETTM, MONTIN, WHEN
GETINC	GETFLD
GETINT	FSO, FSOCLN
GETLIN	CLP,* SYSGEN
GETLIS	ANDER, FSO, NUMBER
GETMEM	FREE, FRET, SIZE
GETP	FSO, INXCON
GETSET	GETFLD
GETTAB	GETFLD
GETTM	INIMON
GETWRD	INITYP
GIVTAB	IFSINT
GNAM	COMARG
GO	CLP*
IFSET	IFSRCH, SEARCH
IFSINT	INIVAR
IFSRCH*	SEARCH
INC	ASCINT, ASCIT, ASCITC, ASCIT6, CTSIT, CTSIT6, EVAL, FSO, GETFLD, GETLIN, INTASC, MATCH, NEXITM, OCTASC, SEEMAT, SPCTRN, STEM, TABLE, TABLK, TRANS, TRASH, TYPEIT

SUBROUTINE	CALLED BY:
INC1	CAPASC, CHKNUM, COMPAR, COMPUL DROP, FIND, FSO, GETFLD, IN., NAM5, NEXITM, OUT., PUTS, SEEMAT, TABLE, TRANS, .C. ASC
INC6	ASCIT, ASCITC, ASCIT6, CTSIT, PREP, TRASH,
INC12	GETLIN
INCHAR	SPCTRN, TABLK
INDENT	FSOCLN, FSO
INFO	CLP*
INIAUT	SEGINT
INICON	INIFIX
INIDSK	DYNAMO, FSO, GETLIN, MONTOR, SUMOUT
INIEND	IFSINT
INIEVL	INIT2
INIFIX	SUPER*
INIFLD	SEGINT
INI. C.	INIFIX
INIMON	DYNAMO
ININT	DYNAMO
INIOUT	INIT2
INIRES	DYNAMO
INIRNG	SEGINT
INIS. T	INIT2
INITDB	SYSGEN
INITYP	DYNAMO, GETLIN, INFO, LONG, SHORT
INIT2	INIFIX
INIVAR	SUPER*
INIVRB	SEGINT
INMON2	SEGINT
INTASC	GETFLD, SEEMAT, TRANS, TYPEIT
IN.	CLP*
INXCON	DYNAMO
IODIAG	ERRGO
ISARG	NEXITM
ISARGV	NEXITM, TYPEIT, .C. ASC

SUBROUTINE	CALLED BY:
JOBTM	MONTIM
KILFAP	DYNAMO
LDOPT	DYNAMO, QUIT
LEGFLD	IN., OUT.
LIBRY	CLP*
LINKUP	CALLIT
LISFIL	LIST*
LIST	CLP*
LISTSL	LIST*
LISTEN	TYPEIT, INTONE
LOCMES	ANDER, IFSINT, IFSRCH, INIDSK, INIFIX, INIVAR, LISTEN, MONTOR, PUT, PREP, REND, SHORT, STRCH, SUMOUT, TABLE, TYPEIT
LOCSEC	IFSRCH
LONG	CLP*
LOOKUP	CLP, NAME
MAINBD	FRALG, INI.C.
MATCH	FSO
MONINT	INIFIX
MONTIM	CALLIT, GETLIN, INIMON, MONTOR, SUPER, TYPEIT
MONTOR	CLP,* INIMON
MOVEIT	USE*
NAM5	IFSRCH (LOCSEC), REORD (MEADIR)
NAME	CLP*
NAP	GETLIN
NEWPT	FSOCLN
NEXITM	AND., AUTHOR, CLP, DROP, INFO, IN., LIST, MONTOR, NAME, NUMBER, OUT.
NOT.	CLP*
NUMBER	CLP*
OCABC	SYSGEN
OCTASC	ERRGO, LISTEN, PUTS, TYPEIT
OCTTOI	FREE, FRET
OPEN	BFOPEN, CHKSAV, CONDIR, CONNAM, DYNAMO, GETINT, GETLIS, IFSINT, INIRES, INITYP, LISFIL, LISTSL, LONG, MOVEIT, NAME, OPFILE, SAVE, SHORT, SUMOUT, SYSGEN, TRETRI, USE

SUBROUTINE	CALLED BY:
OPFILE	DYNAMO, GETLIN, SENTRY
OR.	CLP*
OTBL	TABLK
OUT.	CALLIT
PREP	INIFIX*
PRT12	TYPASH
PUT	ASCIT, ASCIT6, CAPASC, CHKNUM, EVAL, GETFLD, GETLIN, INTASC, IN., NEXITM, OCTASC, OUT., PUTS, SEEMAT, SIGNIN, TABLE, TRANS, .C.ASC
PUTINC	GETFLD
PUTOUT	TRASH
PUTS	TYPEIT (internal)
PUT6	CTSIT, CTSIT6, PREP, TRASH
QUIT	GO, CLP*
RANGE	CLP*
RDFILE	ANDER
RDFLX	SYSGEN
RDFLXA	GETLIN, INIVAR
RDWAIT	ANDER, CALLIT, CHKSAV, CONDIR, CONNAM, FSO, GETINT, GETLIS, IFSINT, IFSRCH, INITYP, LISFIL, LISTSL, MOVEIT, NUMBER, SAVE, SUMOUT, SYSGEN, TRETRI, USE
REND	INIEND
RESTOR	CLP*
RJUST	CHKNUM, QUIT, SYSGEN
RNGNAM	EVAL, RANGE
RPRIME	TABLE
RSCLCK	DYNAMO
SAVBRL	PUTS, TYPEIT
SAVE	CLP*
SEARCH	SUPER*
SEEMAT	CLP,* SUPER,*
SEGINT	SYSGEN
SENTRY	LINKUP
SETBCD	GETLIN, PRT12, TRETRI
SETBLP	AND., SEARCH
SETBRK	ININT, INTONE, TYPEIT

SUBROUTINE	CALLED BY:
SETFUL	GETLIN, PRT12, TRETRI
SETMEM	FREE, SYSGEN
SETRTN	INIRES
SETSYS	DYNAMO, QUIT
SETWRD	DYNAMO, INIDSK, LONG, OPFILE, QUIT, SHORT, TYPEIT
SHIFT	STEM
SHORT	CLP*
SIGNIN	SUPER*
SIGN2	SUPER*
SIZE	INIFIX, SUPER
SLEEP	NAP
SPCTRN	FSO
STANDL	FSO
STBL	TABLK
STEM	S.T
STRACC	SEGINT
SUBJ.	CLP*
SUMOUT	MONTOR, QUIT
SYSGEN	SUPER
S.T	SUBJ., TITLE
TABENT	ANDER, FSO, IFSRCH, NUMBER
TABLE	INIFIX*
TABLK	SPCTRN
TBSRCH	NEXITM
TESTMO	TYPEIT
TIME	CLP*
TIMEIN	(variable)
TITLE	CLP*
TOTTIM	(variable)
TOUT	(variable)
TRANS	MONTIM, SUMOUT
TRASH	TYPASH
TRETRI	FSO
TRFILE	CLEANP, CONDIR, CONNAM, DELIST, INIRES, NAME, QUIT
TSSASC	ASCIT, ASCITC, ASCIT6, GETLIN

SUBROUTINE	CALLED BY:
TYPASH	INIDSK, PUTS, TYPEIT
TYPEIT	ANDER, AND., AUTHOR, CHKNUM, CLP, CONNAM, DROP, ERRGO, EVAL, EXIT, FSO, GETFLD, GETLIN, IFSINT, IFSRCH, INFO, INIFIX, INIVAR, IN., LISFIL, LIST, LISTSL, MONTIM, MONTOR, NAME, NUMBER, OUT., PREP, QUIT, RANGE, REND, , SEARCH, SEEMAT, SHORT, SIGNIN, SIGN2, SPCTRN, SUBJ., SUMOUT, SUPER, S.T, TABENT, TABLE, TIME, TITLE, USE
USE	CLP*
VSRCH	LOOKUP, STEM
WAIT	GETLIN
WFLX	CNTLOC, FREE, FRET, PUTOUT, TYPASH
WFLXA	CNTL FREE, FRET
WHEN	MONTOR
WHOAMI	FSO, INXCON, MONTOR,
WITH.	CLP*
WRFILE	BFWRIT
WRFLX	CALLIT, SYSGEN
WRFLXA	CALLIT, CLP, OUT., PRT12S, SENTRY, SYSGEN, TRETRI, WFLXA
WRHGH	TABLK
WRT	COMENT, LIBRY
WRWAIT	ANDER, CHKSAV, CONDIR, CONNAM, FSO, IFSRCH, INIRES, MOVEIT, QUIT, SAVE, SUMONT
.C.ASC	INIAUT, INICON, INIEVL, INIFLD, INIOUT, INIRNG, INIRES, INIS.T, INIVRB, MONINT, SAVE

APPENDIX F

SUBROUTINE LINKAGES

SUBROUTINE	CALLS
ANDER	BUFSCN*, CLOSE, DORMNT, FWAIT, GETLIS, LOCMES, RDFILE, RDWAIT, TABENT, TYPEIT, WRWAIT
AND.	ANDER, CHKNUM, CTSIT6, DRPPTR. NEXITM, SETBLP, TYPEIT
ASCINT	CHKNUM, GET, INC
ASCIT	FRER, GET6, INC, INC6, PUT, TSSASC
ASCITC	FRET, FREZ, GET6, INC, INC6, TSSASC
ASCTTS	no calls
ASIDE	ASSET
ASSET	no calls
ATLCLN	no calls
ATSCRN	BUFSCN
AUTHOR	CLEANP, COPY, DELIST, FREE, FREZ, GET, NEXITM, TYPEIT
BCDASC	ASCITC
BCDEC	no calls
BFCLOS	no calls
BFCODE	no calls
BFOPEN	BUFFER, OPEN, FWAIT
BFREAD	no calls
BFWRIT	WRFILE, FWAIT
BUFFER	no calls

* -Indicates procedure called via CALLIT

SUBROUTINE	CALLS
BUFSCN	WRWAIT
BZEL	no calls
CALLIT	AND., ASCINT, CHKNAM. DROP, EVAL, FAPDBG, FCLEAN, GETLIN, LINKUP, LIST, MONTIM, NAME, NOT., NUMBER, OR., OUT., RDWAIT, RESTOR, SUBJ., TITLE, USE, WRFLX, WRFLXA
CAPASC	PUT, GET, INC1
CHACAP	no calls
CHANGE	no calls
CHFILE	no calls
CHKNAM	BZEL, CTSIT6, FRET
CHKNUM	PUT, GET INC1, TYPEIT
CHKSAV	BUFFER, CLOSE, FILCNT, TRFILE
CLEANP	DRPPTR, FILCNT, TRFILE, FCLEAN, FRET, ATLCLN*
CLFILE	CLOSE
CLOSE	no calls
CLP	CALLIT. DORMNT, FRET, FREZ, LOOKUP, NEXITM, TYPEIT, WRFLXA
CNTLOC	WFLX, WFLXA, OCTTOI
COMARG	GETCOM, GNAM
COMENT	WRT
COMPAR	GET, INC1
COMPUL	GET, INC1
CONDIR	BUFFER , CLOSE, OPEN, RDWAIT, TRFILE, WRWAIT

SUBROUTINE	CALLS
CONNAM	BUFFER, CLOSE, DELFIL, FILCNT, FRER, FRET, OPEN, RDWAIT, TRFILE, TYPEIT, WRWAIT
COPY	no calls
CTSIT	ASCTSS, FRER, GET, INC, INC6, PUT6
CTSIT6	ASCTSS, FREZ, GET, INC, PUT6
DEAD	no calls
DEC	no calls
DEFBC	no calls
DELBC	no calls
DELFIL	no calls
DELIST	FCLEAN, TRFILE, DRPPTR
DERBC	no calls
DIFF	no calls
DIST	no calls
DNSORT	no calls
DORMNT	no calls
DROP	BCDASC, CHKNAM, CHKSAV*,CONDIR*, DELFIL, INC1, NEXITM, TYPEIT
DRPPTR	no calls
DYNAMO	CLOSE, COMARG, INIDSK, INIMON, ININT, INIRES, INITYP, INXCON, KILFAP, LDOPT, OPEN, OPFILE, RSCLCK, SETSYS, SETWRD
ENDTAB	no calls

SUBROUTINE	CALLS
ERRGO	BDCASC, BFCLOS, DORMNT, IODIAG, OCTASC, TYPEIT
EVAL	COPY, DIST, FLDNAM, FRET, FREZ, GETEND, INC, PUT, RNGNAM, TYPEIT
EXIT	TYPEIT
FAPDBG	no calls
FCHECK	no calls
FCLEAN	FRET
FERRTN	no calls
FIELDS	no calls
FILCNT	FSTATE
FIND	GET, INC1
FLDNAM	no calls
FRALG	FRET, MAINBD
FREC	CNTLOC, DORMNT
FRED	no calls
FREE	CNTLOC , DORMNT, GETMEM, OCTTOI SETMEM, WFLX, WFLXA
FRER	no calls
FRESET	no calls
FRET	CNTLOC , DORMNT, GETMEM, OCTTOI, WFLX, WFLXA
FREZ	CNTLOC., DORMNT, FREE
FSIZE	no calls
FSO	CLOSE, DIST, FIELDS, FLDNAM, FRALG, FSOCLN, GETFLD, GETINT, GETLIS, GETP, INC, INC1, INDENT, INIDSK, MATCH, RDWAIT, SPCTRN, STANDL, TABENT, TRETRI, TYPEIT, WHOAMI, WRWAIT

SUBROUTINE	CALLS
FSCOLN	GETINT, INDENT, NEWPT
FSTATE	no calls
FTRACE	no calls
FWAIT	no calls
GET	no calls
GET6	no calls
GET12	no calls
GETBLP	no calls
GETBRK	no calls
GETCOM	no calls
GETEND	COPY, INIEND
GETFLD	COPY, DEFBC, FREE, FRER, FRET, FREZ, GETINC, GETSET, GETTAB INC, INC1,INTASC, PUT, PUTINC, TYPEIT
GETIME	no calls
GETINC	no calls
GETIN	CLOSE, FRET, OPEN, RDWAIT
GETLIN	ASIDE, CLFILE, FILCNT, FRET, FREZ, GET, GET12, GETINC, INC, INC12 INIDSK, INITYP, NAP, OPFILE, PUT, RDFLXA, SETBCD, SETFUL, TSSASC, TYPEIT, WAIT
GETLIS	FSTATE, OPEN, RDWAIT
GETMEM	no calls
GETOPT	no calls
GETP	no calls
GETSET	no calls
GETSYS	no calls

SUBROUTINE	CALLS
GETTAB	no calls
GETTIM	no calls
GETTM	GETIME
GETWRD	no calls
GNAM	no calls
GIVTAB	no calls
GO	QUIT
IFSET	no calls
IFSINT	CHNCOM, CLOSE, FILCNT, FREZ, FRET, GIVTAB, INIEND, LOCMES, OPEN, RDWAIT, TYPEIT
IFSRCH	BCDASC, BUFSCN, CLOSE, COMPUL, DEFBC, DELIST, ERRGO, FRET, FREZ, GET, IFSET, LOCMES, NAM5, RDWAIT, TABENT, TYPEIT, WRWAIT
INC	no calls
INC1	no calls
INC6	no calls
INC12	no calls
INCHAR	no calls
INDENT	no calls
INFO	CHKNUM, CTSIT6, FRET, INITYP, NEXITM, TYPEIT
INIAUT	FRALG, .C.ASC
INICON	FRALG, .C.ASC
INIDSK	BFCLOS, BFOPEN, FSTATE, LOCMES, SETWRD, TYPASH
INIEND	no calls

SUBROUTINE	CALLS
INIEVL	FRALG, .C. ASC
INIFIX	FRALG, FREE, FRET, INICON, INIT2, INI. C., LOCMES, MONINT, PREP* SIZE, TABLE*, TYPEIT
INIFLD	.C. ASC
INI. C.	MAINBD
INIMON	GETTM, MONTIM
ININT	SETBRK
INIOUT	FRALG. C. ASC
INIPUT	FREE, FRET, FREZ
INIRES	BUFFER, CLOSE, DEFBC, FILCNT OPEN, SETRTN, TRFILE, WRWAIT, .C. ASC
INIRNG	FRALG, .C. ASC
INIS. T	FRALG, FREZ, .C. ASC
INITDB	FREZ
INITYP	FREE, FRET, FSTATE, SETWRD GETWRD, OPEN, RDWAIT, CLOSE
INIT2	INIEVL, INIOUT, INIS. T
INIVAR	CLOSE, COMARG, FREE, IFSINT, LOCMES, RDFLXA, TYPEIT
INIVRB	.C. ASC
INMON2	no calls
INT ASC	INC, PUT
INTONE	SETBRK, GETBRK, LISTEN
INTTWO	GETBRK,
IN.	ASCINT, COMPUL, COPY, FLDNAM, FREE, GET, INC1, LEGFLD, NEXITM, PUT, WHOAMI

SUBROUTINE	CALLS
INXCON	GETP, WHOAMI
IODIAG	no calls
ISARG	no calls
ISARGD	no calls
ISARGP	no calls
ISARGV	no calls
JOBTM	no calls
KILF[illegible]P	no calls
KILNBK	no calls
LDOPT	no calls
LEGFLD	no calls
LIBRY	WRT
LNKUP	SENTRY
LISFIL	BCDASC, BUFFER, CLOSE, FILCNT, OPEN, RDWAIT, TYPEIT
LIST	BCDASC, LISTSL*, LISFIL*, CHKSAV*, CHKNAM NEXITM, TYPEIT
LISTSL	BCDASC, BUFFER, CLOSE, CONDIR, FILCNT, OPEN, RDWAIT, TYPEIT
LISTEN	ASIDE, GETBRK, LOCMES, OCTASC, MONTIM, FSOCLN*
LOCMES	ASCITC
LOCSEC	GET, NAM 5, DEFBC, OPEN
LONG	BUFFER, CLOSE, INITYP, OPEN, SETWRD
LOOKUP	VSRCH
MAINBD	no calls
MATCH	INC, COMPAR, COMPUL
MONINT	FRALG, FREZ, . C. ASC

SUBROUTINE	CALLS
MONTIM	ASIDE, GETIME. JOBTM. TYPEIT
MONTOR	ASCIT6, ASIDE, BCDASC, COMPUL, INIDSK, MONTIM, NEXITM, SUMOUT, TYPEIT, WHEN, WHOAMI
MOVEIT	BUFFER, OPEN, RDWAIT, WRWAIT
NAM5	GET, INC1
NAME	ANDER, BUFFER, CHKNAM, CLOSE, DRPPTR, LOOKUP, NEXITM, OPEN, TRFILE, TYPEIT
NAP	SLEEP
NEWPT	FCLEAN, WRWAIT, TABENT, CLOSE,
NEXITM	COPY, DEC1, FIND, FRET, FREZ, GET, INC, INC1, ISARG, ISARGV PUT, TBSRCH
NOT.	no calls
NUMBER	ASCINT, CLEANP*, DELIST, DNSORT, FRET, FREZ, GETLIS, NEXITM, RDWAIT, TABENT, TYPEIT
OCABC	no calls
OCDBC	no calls
OCLBC	no calls
OCRBC	no calls
OCTASC	INC, PUT
OCTTOI	no calls
OPEN	no calls
OPFILE	OPEN, SETWRD
OR.	no calls
OTBL	no calls
OUTCOM	no calls

SUBROUTINE	CALLS
OUT.	ASCINT, COMPUL, FLDNAM, GET, INC1, LEGFLD, NEXITM, PUT, TYPEIT, WRFLXA
PREP	ASCITC, BCDEC, BFCLOS, FBOPEN, DORMNT, FILCNT, FRALG, FRET, FREZ, GET6, LOCMES, PUT6, TYPEIT
PRT12	SETBCD, SETFUL, WRFLXA
PUT	no calls
PUTINC	no calls
PUTOUT	INC6, PUT6
PUTS	ASIDE, BFWRIT, DEC1, DORMNT, FSTATE, GETBRK, INC1, LOCMES, OCTASC, PUT, SAVBRK, TYPASH
PUT6	no calls
QUIT	BCDASC, BUFFER, BZEL. CHNCOM, CLOSE, CTSIT6, DORMNT, FRET, LDOPT, NEXITM, RJUST, SETSYS, SETWRD, SUMOUT, TRFILE, TYPEIT, WRWAIT
RANGE	ASCINT, COMPUL, FRET, FREZ, NEXITM, RNGNAM, TYPEIT
RDFILE	no calls
RDFLX	RDFLXA
RDFLXA	no calls
RDWAIT	no calls
REND	BFCLOS, BFOPEN, BFREAD, CHNCOM, ENDTAB, FILCNT, FRET, FREZ, LOCMES, TYPEIT
RESTOR	CLEANP*, DELIST
RJUST	no calls
RNGNAM	no calls
RPRIME	no calls

SUBROUTINE	CALLS
RSCLCK	no calls
RSOPT	no calls
RSTRTN	no calls
SAVBRK	no calls
SAVE	BUFFER, CHKNAM, CHKSAV, CLOSE, NEXITM, OPEN, RDWAIT, WRWAIT, .C.ASC
SEARCH	ANDER, ASIDE, ATSCRN, CLOSE, DEFBC, FREZ, FSTATE, IFSET, IFSRCH, LOCMES, NAM5, SETBLP, TYPEIT
SEEMAT	COPY, INC, INC1, INTASC, PUT, TOTTIM, TYPEIT
SEGINT	INIAUT, INIFLD, INIRNG, INIVRB, INMON2, STRACE
SENTRY	no calls
SETBCD	no calls
SETBLP	no calls
SETBRK	no calls
SETFRE	no calls
SETFUL	no calls
SETMEM	no calls
SETNBK	no calls
SETNCV	no calls
SETOPT	no calls
SETP	no calls
SETRTN	no calls
SETSYS	no calls
SETWRD	no calls

SUBROUTINE	CALLS
SHIFT	no calls
SHORT	BUFFER, CLOSE, DORMNT, INITYP, OPEN, SETWRD, TYPEIT
SIGNIN	CTSIT6, FRALG, GET, NEXITM, PUT, TYPEIT
SIZE	CETMEM
SLEEP	no calls
SPCTRN	COMPAR, COMPUL, DIST, FIND, INC, INCHAR, TABLK, TYPEIT
STANDL	no calls
STBL	no calls
STEM	COPY, DEC1, GET, INC, SHIFT, VSRCH
STOPCL	no calls
STRACC	no calls
SUBJ.	CLEANP, DELIST, ~~FRET~~, FREZ, S. T, TYPEIT
SUMOUT	ASIDE, BCDASC, BFWRIT, BUFFER, CLOSE, DELFIL, INIDSK, LOCMES, OPEN, RDWAIT, TYPEIT, WRWAIT
SUPER	ASIDE, INIFIX*, INIVAR*, DYNAMO*, SIGNIN*, FSO*, FEIZE, MONTIM, SIZE, CLP, DORMNT, SEARCH*, EVAL*, FSIZE, MONTIM, SIZE, SYSGEN, SEEMAT* TYPEIT
SYSGEN	BCDEC, BUFFER, CHFILE, CHNCOM, CLOSE, DELFIL, FERRTN, FRALG, FREE, FRESET, FSTATE, GETCOM, GETLIN, INITDB, OCABC, OPEN, RDFLX, RDWAIT, RJUST, SEGINT, SETMEM, WRFLX, WRFLXA
S. T	COPY, ~~FRET~~, FREZ, NEXITM, STEM, TYPEIT

SUBROUTINE	CALLS
TABENT	TYPEIT
TABLE	ASCINT, BFCLOS, BFOPEN, BFREAD, COPY, DORMNT, FILCNT, FRALG, FREE, FREZ, INC, INC1 LOCMES, NEXITM, PUT, RPRIME, TYPEIT
TABLK	COMPAR, COMPUL, FRET, FREZ, GET, INC, INCHAR, OTBL, STBL, WRHGH
TBSRCH	no calls
TESTMO	no calls
TILOCK	no calls
TIMCHK	no calls
TIME	CLOSE, NEXITM, TYPEIT
TIMEIN	no calls
TIMER	no calls
TIMLFT	no calls
TITLE	CLEANP, DELIST, FRET, FREZ, S. T , TYPEIT
TOTTIM	no calls
TOUT	no calls
TRANS	COPY, DEC1, INC, INTASC, PUT
TRASH	ASCTSS, INC, INC6, GET, PUTOUT, PUT6
TRETRI	DERBC, OPEN, RDWAIT, SETBCD, SETFUL, WRFLXA
TRFILE	no calls
TSSASC	no calls
TYPASH	DORMNT, FREE, FRET, PRT12, TRASH, WFLX

SUBROUTINE	CALLS
TYPEIT	ASIDE, BCDASC, BFOPEN, BFWRIT DEC1, DORMNT, FRET, FSTATE, GET, GETBRK, INC, INC1, INITYP, INTASC, ISARGV, LISTEN, LOCMES, MONTIM, OCTASC, PUTS, SAVBRK, SETBRK, SETWRD, PUT, TESTMO, TYPASH RDWAIT
UPDATE	no calls
USE	BCDASC, BUFFER, CHKSAV*, CLOSE, CONDIR*, DELFIL, FSTATE, MOVEIT*, NEXITM, OPEN, RDWAIT, TYPEIT
VSRCH	no calls
WAIT	no calls
WFLX	WRFLX
WFLXA	WRFLXA
WHEN	GETIME
WHOAMI	no calls
WHOM	WHOAMI
WITH.	no calls
WRFILE	no calls
WRFLX	no calls
WRFLXA	no calls
WRHGH	no calls
WRT	ASIDE, NEXITM
WRWAIT	no calls
.C.ASC	ASCITC, FRET, GET, INC1, ISARGV, PUT

APPENDIX G

GLOSSARY OF INTREX TERMS

TERM	DEFINITION
affix	1) The ending removed from a subject or title word by stemming. 2) The initials of an author.
anding mode	The mode of operation to be used by the Boolean procedure ANDER, which handles all types of Boolean commands in the manner directed by the mode.
attribute	A particular property of a reference word which may be specified by the user to narrow a search request.
augmented pointer	Three consecutive computer words containing the addr ss, length, type, etc. of a reference list.
CL.	A pointer to the Command List array. It is stored in one of the three COMMON words of core accessible to all of Intrex.
combined search	A search request consisting of two or three of the th ee types of primary searches (subject, tit author). These search requests are give n the same command line, separated l lashes (/) and cause an intersection of document numbers of the separate lists.
Command List	A list o user command parameters containing poi ers to the search structure, resultant ref rence list, and output request data.
Command List Pointer	See CL.
common buffer	One of several blocks of 432 consecutive core words whose addresses are in the POT to make them available as I/O buffers from anywhere in Intrex.
condition code	A letter code attached to each ending in the ending table specifying the exceptional conditions in which the ending may <u>not</u> be removed by stemming.

TERM	DEFINITION
current list	A list of Inverted File references resulting from a search, Boolean operation, document command, or restoration of a NAMEd list. It is available for output of catalog information or full text.
data base initialization	That part of the initialization of the Intrex software in which names of the data base files are specified via the time-sharing console.
document count	The number of different document numbers in a reference list (usually lower than the number of references since most documents have several appearances of the same key word.)
Dump File	A file used by Intrex which holds the results of Boolean operations and is kept open for both reading and writing.
ending code	A 12-bit code representing the address in the ending table where a particular ending will be found.
ending table	A table of English word endings which may be removed to leave a common stem more suitable for searching on subjects or titles.
error code	A value returned from a procedure indicating an abnormal condition has occurred during execution of that procedure.
fence	A special computer word, file word, or table entry which contains a distinguishable code signaling the end of the data, (often $777777\ldots7_8$).
find-point	The place in the Inverted File where the search word should be found if it is in the file.
fixed-parameter initialization	The phase of Intrex initialization which does not depend on the particular console, user, mode, date, etc. and so can be done once by Intrex personnel and saved as a semi-initialized system.
full-term shred	(See shred)
Guide Directory	A disk file used by Intrex as an index to the Guide File.
Guide File	A disk file containing the text of the labeled sections of the on-line User's Guide available through the INFO command.

TERM	DEFINITION
held system	An Intrex session which is "resumed" using the word HOLD as an argument causing automatic recycling through the subsystem INXSUB, back to start Intrex again after a QUIT or BEGIN command.
list	An Inverted File entry containing one English word (or character group) with its associated affix codes and references.
list header	A group of three computer words containing data (counts, etc.) pertaining to the Inverted File list which follows it.
list pointer	A single computer word containing the address of an augmented (3-part) pointer to a reference list and with the document count of that list in the decrement portion of the pointer.
long mode	One of the two dialog modes of Intrex which is designed to instruct and inform the user in his interactions with the system.
Message Directory	A disk file used by Intrex as an index to the Message File and which also contains the text of core-stored message components.
Message File	A disk file containing the text of labeled message components used by the Intrex dialog.
Monitor File	A disk file used to record all user/Intrex transactions and related timing data.
Name File	A disk file used by Intrex to hold the reference lists retained by the user's NAME commands.
Parameter Option Table	See POT
password	A code word of up to six characters, which may be specified after the HOLD argument of a "resume Intrex" command, providing a means of escaping from a held system back to CTSS command level.
Password File	A disk file used by Intrex to retain the password of a held system and the dialog mode for use by the subsystem in restarting Intrex.
POT	An array containing a collection of Intrex parameters containing over 40 computer words of assorted data which help to control the operation of Intrex.

TERM	DEFINITION
POT.	See POT pointer.
POT Pointer	A pointer containing the address of the start of the POT array. It is stored in one of the three COMMON words available to all Intrex software.
primary search	A search of the Inverted Files for matches on the search term or author name.
range restriction	Specification by the user of a particular subject term weight (0-5) to be used as a reference selection criterion.
record	1) A catalog entry describing one document in the data base. 2). A disk segment consisting of 432 computer words.
reference	A computer word containing several items of information pertaining to an Inverted File indexing word.
reference list	A contiguous group of reference words which share one or more characteristics.
relloc	The depth into a disk file at which reading or writing is to begin.
resultant reference list	See current list
Save File	A disk file named by the user for the purpose of holding NAMEd lists for future Intrex sessions via use of the SAVE command.
search term	The subject or title phrase or author name given by the user in his search request.
search word	A single English word (or group of characters), delimited by spaces or hyphens, which is part of a search term.
secondary search	An exact-character match of the string of characters given by the user in a RESTRICT command against the characters in the specified field.
section	Part of an Inverted File segment consisting of 432 computer words, or 1 disk record.
section header	A single computer word containing a pointer to the first list in an Inverted File section.
segment	A disk file consisting of part of the set of Inverted Files or Catalog Files.

TERM	DEFINITION
session initialization	The phase of Intrex software initialization which must be performed at the outset of each intrex session. This complements fixed-parameter initialization.
short mode	The dialog mode of Intrex which uses abbreviations and keeps prompting to a minimum.
shred	A unit of data produced by processing the catalog input and consisting of one index term with its accompanying reference.
SST.	A pointer to the System State Table. It is stored in one of the three COMMON words available to all Intrex software.
stemmed shred	A shred produced from a full-term shred by stemming and decomposing into single index words and adding an affix field. These shreds are used to generate or update the Inverted Files.
stemming	The removal of common endings from the words in an index term according to an algorithm and table.
System State Table	A computer word in which each bit position is a Boolean indicator specifying if particular system conditions are true or false.
System State Table Pointer	See SST.

APPENDIX H

DATA BASE GENERATION PROCEDURE

1. Data is entered into the computer in the form of files containing ten catalog records each. This data is typed in from computer consoles, using the edit program QED.[15]

2. The input files are proofread and then corrected using QED.

3. DRYRUN is run to examine the files for errors.

4. Errors found by DRYRUN are corrected using QED.

5. The program WETRUN generates three files and updates two others:

   a. SUBTIT date is generated which contains the subject and title terms (shreds) extracted from field 73 and field 24 of each catalog record.

   b. AUTHOR date is generated which contains the names of the authors which were in field 21 of each record.

   c. Catalog segments with names of the form CRxxx M25100 are generated. Each segment contains about fifty catalog records. The data in these segments has been reformatted to make it more easily accessible by the retrieval program.

   d. The file CATDIR FILE is updated with pointers to the new catalog records.

   e. FICHE DIRECT is updated with the fiche addresses of the new documents.

6. The catalog segments generated by WETRUN are compressed to 55% of their original size. The program MASH does the compression by converting the ASCII character codes to digram codes, which represent two characters in one nine-bit byte.

7. AUTHOR date is processed by SORT to produce the file SORTA date.

8. IFGENA is run to integrate the data in SORTA with the current author Inverted File, thus generating a new set of Inverted File segments, AIxxx date. IFGENA also produces the directories IFDA date and IFTABA date. These directories are used by the retrieval program to locate authors in the Inverted Files.

9. STEMER is used to parse the subject and title terms in SUBTIT date into individual word shreds and then to stem them. These stems are written into the file STEMED date.

10. SORT is a run which uses STEMED date to generate a file of sorted subject and title stems, called SORTS data.

11. IFGENS is used to merge SORTS date with the current subject/title Inverted File, producing the segments SIxxx.date. IFGENS also creates two levels of directories for the Inverted File: IFDS date and IFTABS date.

12. IFTEST is executed to scan the new Inverted Files for format errors.

13. IFLIST is used to list the contents of the Inverted Files creating the ASCII files SUBFIL IFLIST and AUTFIL IFLIST.

APPENDIX I
INDEX TO SUBROUTINES IN CHAPTER III

ACLN 3.2.4.6
AI 3.2.3.5
ANDER 3.3.4.5
AND. 3.3.4.1
ASCINT 3.4.2.10
ASCIT6 3.4.2.7
ASCITC 3.4.2.9
ASCTSS 3.4.2.6
ASIDE 3.1.6.4
ASRCH 3.2.4.5
ASSET 3.1.6.4
ATLCLN 3.2.8.16
ATSCRN 3.2.4.3
AUTHOR 3.2.3.2
BCDASC 3.1.7.4
BC 3.5.6.1
B 3.5.2.4
BFOPEN 3.5.2.1
BFREAD 3.5.2.2
BRWRIT 3.5.2.3
BUFFER 3.5.1.3
BUFSCN 3.3.4.7
BZEL 3.5.6.6
CALLIT 3.1.9.2
CHFILE 3.5.3.1
CHKNAM 3.3.2.4
CHKNUM 3.4.2.11
CHKSAV 3.3.3.2
CLEANP 3.2.4.6
CLFILE 3.1.3.3
CLOSE 3.5.1.2
CLP 3.2.1.2
CNTLOC 3.4.1.4
COMARG 3.5.7.15
COMENT 3.2.9.2
COMPAR 3.4.4.12
COMPUL 3.4.4.13
CONDIR 3.3.3.5
CONNAM 3.3.2.8
COPY 3.4.4.9
CTSIT6 3.4.2.8
DEC1 3.4.4.19
DEFBC 3.5.6.2
DELFIL 3.5.3.2
DELIST 3.2.4.7
DERBC 3.5.6.3
DIST 3.4.4.10
DNSORT 3.4.5.1
DORMNT 3.5.7.1
DROP 3.3.2.7
DRPPTR 3.2.4.8
DYNAMO 3.1.3.1
ENDTAB 3.1.2.15
ERRGO 3.1.10.1
EVAL 3.2.6.2
EXIT 3.1.4.3
FAPDBG 3.4.5.2
FCLEAN 3.2.4.9
FERRTN 3.5.4.1
FIELDS 3.1.2.12
FILCNT 3.4.3.1
FIND 3.4.4.11
FLDNAM 3.1.2.10
FRALG 3.4.1.10

FREE 3.4.1.1
FRER 3.4.1.6
FRESET 3.4.1.8
FRET 3.4.1.2
FREZ 3.4.1.3
FSIZE 3.4.1.7
FSO 3.2.8.1
FSOCLN 3.2.8.5
FSTATE 3.5.3.3
FTRACE 3.4.1.9
FWAIT 3.5.1.9
GATP 3.2.3.3
GET 3.4.4.1
GET6 3.4.4.4
GET12 3.4.4.7
GETBRK 3.5.7.4
GETBUF 3.3.4.5
GETCOM 3.5.7.14
GETEND 3.2.6.3
GETFLD 3.2.8.6
GETIME 3.5.7.22
GETINC 3.4.4.16
GETINT 3.2.8.4
GETLIN 3.2.1.3
GETLIS 3.3.4.6
GETMEM 3.5.7.7
GETP 3.6.1.5
GETSET 3.4.4.15
GETSYS 3.5.7.18
GETTAB 3.2.8.7
GETIME 3.5.7.22
GETTM 3.5.7.23
GETWRD 3.5.7.11
GIVTAB 3.1.2.16
GNAM 3.5.7.19
GO 3.1.4.1
IFSET 3.1.2.14
IFSINT 3.1.2.14
IFSRCH 3.2.5.1
INC 3.4.4.2
INC1 3.4.4.18
INC6 3.4.4.5
INC12 3.4.4.8
INCHAR 3.1.7.6
INDENT 3.1.7.5
INFO 3.2.9.7
INIAUT 3.2.3.1
INICON 3.2.1.1
INIDSK 3.1.6.2
INIEND 3.1.2.15
INIEVL 3.2.6.1
INIFIX 3.1.2.3
INIFLD 3.2.8.3
INI.C. 3.4.2.2
INIMON 3.1.6.1
ININT 3.1.8
INIOUT 3. . 7.
INIRES 3.3.2.1
INIRNG 3.2.8.14
INIS.T. 3.2.2.1
INITDB 3.1.2.2
INITYP 3.1.7.1
INIT2 3.1.2.4
INIVAR 3.1.2.13
INIVRB 3.2.9.1
INMON2 3.1.6.1
INTASC 3.4.2.3
INTONE 3.1.8.2
INTTWO 3.1.8.3
IN. 3.2.7.3

INXCON 3.1.3.2
INXSUB 3.1.10.3
IODIAG 3.5.4.2
ISARG 3.6.1.3
ISARGV 3.6.1.3
JOBTM 3.5.7.21
KILFAP 3.1.3.4
LDOPT 3.5.7.16
LEGFLD 3.1.2.9
LIBRY 3.2.9.3
LINKUP 3.1.9.5
LISFIL 3.3.3.7
LIST 3.3.2.6
LISTSL 3.3.3.6
LISTEN 3.1.8.4
LN 3.2.3.4
LOCMES 3.1.7.3
LOCSEC 3.2.5.2
LONG 3.2.9.5
LOOKUP 3.2.1.5
MAINBD 3.1.9.4
MATCH 3.4.4.14
MEADIR 3.2.4.4
MONINT 3.1.5.1
MONTIM 3.1.5.2
MONTOR 3.1.6.3
MOVEIT 3.3.3.4
NAM5 3.2.5.3
NAME 3.3.2.2
MATAFF 3.2.5.4
NAP 3.4.5.3
NCOM 3.5.7.25
NEWPT 3.2.8.2
NEXITM 3.2.1.4
NOT. 3.3.4.4
NUMBER 3.3.1.1
OCABC 3.5.6.4
OCTASC 3.4.2.4
OCTTOI 3.6.1.6
OPEN 3.5.1.1
OPFILE 3.1.3.3
OR. 3.3.4.3
OTBL 3.2.8.12
OUT. 3.2.7.2
PREP 3.1.2.5
PRT12 3.1.7.9
PUT 3.4.4.3
PUTINC 3.4.4.17
PUTOUT 3.1.7.8
PUT6 3.4.4.6
QUIT 3.1.4.4
RANGE 3.2.8.15
RDFILE 3.5.1.4
RDFLX 3.5.5.2
RDFLXA 3.5.5.1
RDWAIT 3.5.1.5
REND 3.1.2.15
REORD 3.2.4.4
RESTOR 3.3.2.5
RJUST 3.5.6.5
RNGNAM 3.1.2.8
RPRIME 3.1.2.7
RSCLCK 3.5.7.20
SAVBRK 3.5.7.6
SAVE 3.3.3.1
SCLS 3.5.7.24
SEARCH 3.2.4.1
SEEMAT 3.2.9.8
SEGINT 3.1.2.1
SENTRY 3.1.9.3

SETBCD	3.5.5.6
SETBRK	3.5.7.5
SETFUL	3.5.5.5
SETMEM	3.5.7.8
SETRTN	3.1.10.2
SETSYS	3.5.7.17
SETWRD	3.5.7.10
SHIFT	3.4.5.6
SHORT	3.2.9.6
SIGNIN	3.1.4.2
SIZE	3.4.1.5
SLEEP	3.5.7.2
SPCTRN	3.2.8.9
SSRCH	3.2.4.2
STANDL	3.1.2.11
STBL	3.2.8.11
STCLN	3.2.4.6
STEM	3.2.2.5
STRACC	3.4.5.7
STRCH	3.2.4.2
SUBJ.	3.2.2.2
SUMOUT	3.1.5.4
SUPER	3.1.1.1
SYSGEN	3.1.9.1
S.T	3.2.2.4
TABENT	3.3.2.3
TABLE	3.1.2.6
TABLK	3.2.8.10
TBSRCH	3.4.5.4
TESTMO	3.4.5.8
TIME	3.1.5.3
TIMEIN	3.4.5.10
TITLE	3.2.2.3
TOTTIM	3.4.5.10
TOUT	3.4.5.10
TRANS	3.1.5.5
TRAC	3.4.5.9
TRASH	3.1.7.8
TRETRI	3.2.8.8
TRFILE	3.5.1.8
TSPOT	3.4.5.9
TSRCH	3.2.4.2
TSSASC	3.4.2.5
TYPASH	3.1.7.7
TYPEIT	3.1.7.2
USE	3.3.3.3
VSRCH	3.4.5.5
WAIT	3.5.7.3
WFLX	3.6.1.1
WFLXA	3.6.1.2
WHEN	3.4.5.11
WHOAMI	3.5.7.9
WITH.	3.3.4.2
WRFILE	3.5.1.6
WRFLX	3.5.5.4
WRFLXA	3.5.5.3
WRHGH	3.2.8.13
WRT	3.2.9.4
WRWAIT	3.5.1.7
.C.ASC	3.4.2.1

REFERENCES

1. C. F. J. Overhage, Project Intrex - A Brief Description, Project Intrex, M. I. T., Cambridge, Mass.

2. Project Intrex - An Overview for Users, Electronic Systems Laboratory, M. I. T., Cambridge, Mass.

3. J. F. Reintjes, "System Characteristics of Intrex", Proceedings of the Spring Joint Computer Conference, 1969, pp. 457, 459.

4. R. S. Marcus, P. Kugel, and R. L. Kusik, "An Experimental Computer-Stored Augmented Catalog of Professional Literature", Proceedings of the Spring Joint Computer Conference, 1969, pp. 461, 473.

5. D. R. Knudson and S. N. Teicher, "Remote Text Access in a Computerized Library Information Retrieval System", Proceedings of the Spring Joint Computer Conference, 1969, pp. 475, 481.

6. D. R. Haring and J. K. Roberge, "A Combined Display for Computer Generated Data and Scanned Photographic Images", Proceedings of the Spring Joint Computer Conference, 1969, pp. 483, 490.

7. D. Griffin, "Compressing and Catalog Record Data Base to Save Storage", Intrex Memorandum ISR-44, October 27, 1969.

8. J. B. Lovins, "Development of a Stemming Algorithm", Mechanical Translation, Vol. 11, Nos. 1 and 2, March and June 1968.

9. R. L. Kusik, A File Organization for the Intrex Information Retrieval System, Report ESL-TM-415, Electronics Systems Laboratory, M. I. T., Cambridge, Mass., January 1970.

10. R. E. Goldschmidt, File Design for Computer-Resident Library Catalogs, Report ESL-R-451, Electronic Systems Laboratory, M. I. T., Cambridge, Mass., June 1971.

11. A. Benenfeld, "Input/Output Representation of Special Characters", Intrex Memorandum 4, Project Intrex, M. I. T., Cambridge, Mass.

12. N. Goto, "A Translator Program for Displaying a Computer Stored Set of Special Characters," Report ESL-R-429, Electronic Systems Lab., M.I.T., Cambridge, Mass., July 1970.

13. AED-0 Manual, M.I.T. Information Processing Center, Cambridge, Mass.

14. TIP Sort Package, Report TIP-TM-106, Project TIP, M. I. T., Cambridge, Mass.

15. CTSS Programmer's Manual, M.I.T. Information Processing Center, Cambridge, Mass.

16. MULTICS Programmer's Manual, Project MAC, M.I.T., Cambridge, Mass.