

AD-783 409

MACHINE INDEPENDENT DATA MANAGEMENT  
SYSTEM (MIDMS) SYSTEM SPECIFICATIONS

Defense Intelligence Agency  
Washington, D. C.

1 July 1974

DISTRIBUTED BY:

**NTIS**

National Technical Information Service  
U. S. DEPARTMENT OF COMMERCE  
5285 Port Royal Road, Springfield Va. 22151

AD 783409

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Machine Independent Data Management System (MIDMS) System Specifications		5. TYPE OF REPORT & PERIOD COVERED User Documentation
7. AUTHOR(s) Defense Intelligence Agency (DIA)		6. PERFORMING ORG. REPORT NUMBER
8. PERFORMING ORGANIZATION NAME AND ADDRESS Defense Intelligence Agency ATTN: Information Processing Division (DS-5) Washington, D.C. 20301		9. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Same as 9		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE 1 July 1974
		13. NUMBER OF PAGES 224
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; unlimited distribution		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
MIDMS	Subsystem	Fixed Set
File Structuring	Subroutine	Variable Set
Librarian	Retrieval	Module
File Maintenance	Output	
Language Processor	Periodic Set	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) This document describes the MIDMS system specifications. Each module of the system is described in detail to include its subsystem structure, subprogram identification and description, flow charts, and error messages. Differences between the IBM and Honeywell versions of MIDMS are documented.		

Reproduced by  
NATIONAL TECHNICAL  
INFORMATION SERVICE  
U S Department of Commerce  
Springfield VA 22151

MACHINE INDEPENDENT DATA MANAGEMENT SYSTEM

(MIDMS)

SYSTEM SPECIFICATIONS

*Enclosures A, B, C. + D*

## FOREWORD

This manual has been developed by the Defense Intelligence Agency (DIA) for technical purposes. It does not reflect either explicitly or implicitly official DIA policy or intelligence matters. Its purpose is to provide a detailed description of the Machine Independent Data Management System (MIDMS) programs. DIA assumes no system installation, program maintenance, or system operation responsibility, nor is DIA responsible for the data upon which the system operates.



# TABLE OF CONTENTS

	PAGE
CHAPTER 1 - FILE STRUCTURING - <i>AD-783406</i>	
1. Subsystem Structuring .....	1-1
2. COBOL Data Division Entries .....	1-3
3. FS Subprogram Identification and Description .....	1-22
a. FSX (Supervisor) .....	1-22
b. FSSNX (Word Scan) .....	1-29
c. FSPRX (Print/Error Subroutines) .....	1-33
d. FSJBX (Job Card) .....	1-36
e. FSEDX (EDITS) .....	1-39
f. FSSEX (Subroutine and Table) .....	1-41
g. FSFLX (Field Card) .....	1-44
h. FSIOX (Input Output Function) .....	1-50
i. FSDTX (Management Date) .....	1-53
j. FSGOX (FIELD Card Continued) .....	1-55
k. FSCRX (GROUP Card) .....	1-57
l. FSVSX (VSET Card) .....	1-64
m. FSENX (ENDFS Card) .....	1-66
n. FSDDX (COBOL Data Division) .....	1-70
4. File Structuring Error Messages .....	1-74
5. Quantitative Limits .....	1-84
CHAPTER 2 - LIBRARIAN - <i>AD-783406</i>	
1. Subsystem Structure .....	2-1
2. Subprogram Identification and Description .....	2-1
a. LBLB .....	2-2
b. LBCMPRS .....	2-13
c. LBEXPAND .....	2-14
d. LBMOVEC .....	2-15
e. LBJBNAME .....	2-18
f. LBENQ .....	2-18
g. LBDEQ .....	2-19
h. LBRNAME .....	2-19
3. Error Messages .....	2-20
CHAPTER 3 - FILE MAINTENANCE (FM) - <i>AD-783407</i>	
Overview	
1. General .....	3-1
2. File Maintenance (FM) Subprograms .....	3-3
a. FM .....	3-3
b. FMIOX .....	3-5
c. FMSCN .....	3-6
Section I - File Maintenance Language Processor (FMLP)	
1. Overview .....	3-I-1
2. FMLP Subprograms	
a. FMLP .....	3-I-3
b. FMLP2 .....	3-I-4
c. FMLPVAL .....	3-I-5

## Section II - Ordinary Maintenance Language Processor (OMLP)

1. Subsystem Structure .....	3-II-1
2. COBOL Entries and Subroutines .....	3-II-3
a. DSDOL Entries .....	3-II-3
b. VAL-LIT-AREA Entries .....	3-II-4
c. Subroutines .....	3-II-7
(1) OMLPX .....	3-II-7
(2) OMLPFIL .....	3-II-11
(3) OMLPGRP .....	3-II-19
(4) OMLPREC .....	3-II-23
(5) OMLPFLD .....	3-II-26
(6) OMLPINT .....	3-II-28
(7) OMLPOPR .....	3-II-32
(8) OMLPOP2 .....	3-II-36
(9) OMLPOP3 .....	3-II-39
(10) OMLPOP4 .....	3-II-41
(11) OMLPRNG .....	3-II-44
(12) OMLPVAL .....	3-II-46
(13) OMLPRO .....	3-II-51
(14) OMLPSBR .....	3-II-54
(15) OMLPPSS .....	3-II-56
(16) OMLPWRP .....	3-II-59
(17) OMLPWRT .....	3-II-61
(18) OMLPWRP .....	3-II-62
3. Error Messages .....	3-II-63

## Section III - File Maintenance (Ordinary) Input Processor (FMIP)

1. Overview .....	3-III-1
2. File Maintenance (Ordinary) Input Processor (FMIP)	
Subprograms	
a. FMIPX .....	3-III-5
b. FMIPCD .....	3-III-13
c. FMIPREC .....	3-III-26
d. FMIPUN .....	3-III-44
e. FMIPVAL .....	3-III-57
f. FMIPTRN .....	3-III-117
g. FMIPBIN .....	3-III-130
h. FMIPSRT .....	3-III-132
3. IP Error Messages .....	3-III-133

## Section IV - File Maintenance Maintenance Proper (FMP)

1. Overview .....	3-IV-1
2. FMP Subprograms .....	3-IV-3
a. FMMPX .....	3-IV-3
b. MPD1 .....	3-IV-30
c. MPD2 .....	3-IV-49
d. OMMPX .....	3-IV-62
e. OMMPPIO .....	3-IV-125
f. OMMPTRN .....	3-IV-128

	PAGE
g. LMPX .....	3-IV-146
h. MPSRTX .....	3-IV-147
i. FMPSRT .....	3-IV-148
j. FMMPMRG .....	3-IV-149
k. OMOVL .....	3-IV-156
3. File Maintenance Confirmations and Error Messages ...	3-IV-158
Section V - Logical Maintenance	
1. Subsystem Structure .....	3-V-1
2. Subprogram Identification and Description .....	3-V-8
a. LMLP .....	3-V-8
b. LMLPASN .....	3-V-11
c. LMLPATC .....	3-V-13
d. LMLPBLD .....	3-V-16
e. LMLPCNG .....	3-V-19
f. LMLPDDS .....	3-V-23
g. LMLPDEF .....	3-V-25
h. LMLPDEL .....	3-V-29
i. LMLPFLD .....	3-V-32
j. LMLPFMT and LMLPFMT1 .....	3-V-42
k. LMLPCEN .....	3-V-46
l. LMLPCRP .....	3-V-47
m. LMLPMLT .....	3-V-50
n. LMLPMOV .....	3-V-52
o. LMLPNM0 .....	3-V-55
p. LMLPNM1 .....	3-V-57
q. LMLPNM2 .....	3-V-60
r. LMLPNM3 .....	3-V-62
s. LMLPPG1 .....	3-V-64
t. LMLPPG2 .....	3-V-67
u. LMLPPRT .....	3-V-69
v. LMLPPUT .....	3-V-72
w. LMLPRFD .....	3-V-75
x. LMLPRLN .....	3-V-78
y. LMLPRMN1 .....	3-V-81
z. LMLPRMS .....	3-V-86
aa. LMLPRST .....	3-V-89
ab. LMLPRT1 .....	3-V-94
ac. LMLPRT2 .....	3-V-98
ad. LMLPSCN .....	3-V-102
ae. LMLPSPO .....	3-V-106
af. LMLPSP1 .....	3-V-111
ag. LMLPSTE .....	3-V-114
ah. LMLPSUB .....	3-V-116
ai. LMLPTAB .....	3-V-122
aj. LMLPVPT .....	3-V-125
ak. LMLPWF0 .....	3-V-127
al. LMLPWP1 .....	3-V-129
am. LMLPWP2 .....	3-V-133
an. LMLPWP3 .....	3-V-136

	PAGE
ap. LMLPWP5 .....	3-V-142
aq. MLMPZNO .....	3-V-144
ar. LMLPZN1 .....	3-V-146
as. LMLPZN2 .....	3-V-150
at. LMLPZN3 .....	3-V-152
3. LM Error Messages .....	3-V-154
CHAPTER 4 - RETRIEVAL AND OUTPUT DOCUMENTATION PART I - AD-783 408	
a. Subsystem Structure .....	4-1
(1) Modules and Subroutines .....	4-1
(a) COBOL Programs .....	4-1
(b) ALC Subroutines .....	4-2
b. Subprogram Identification and Description .....	4-8
(1) GENO .....	4-8
(2) GEN1 .....	4-8
(3) GEN1A .....	4-12
(4) GEN2 .....	4-12
(5) GEN2X .....	4-18
(6) GEN4X1 .....	4-20
(7) GEN4X2 .....	4-22
(8) GEN4X3 .....	4-23
(9) GEN3A .....	4-24
(10) GEN3 .....	4-33
(10.1) GEN3B .....	4-37
(11) GEN4 .....	4-37.2
(12) GEN4A .....	4-43
(13) GEN5 .....	4-45
(14) GEN5A .....	4-44
(15) GEN6 .....	4-45
(16) GEN6A .....	4-51
(17) GENAA .....	4-51
(18) GEAB .....	4-52
(19) GEAC .....	4-52
(20) GEAD .....	4-53
(21) GEAG .....	4-54
(22) GEAL .....	4-54
(23) GEAM .....	4-55
(24) GEAN .....	4-56
(25) GEAP .....	4-56
(26) GEAS .....	4-57
(27) GEAT .....	4-58
(28) GEAX .....	4-58
(29) GEAZ .....	4-59
c. Module Error Messages .....	4-61
CHAPTER 5 - RETRIEVAL AND OUTPUT DOCUMENTATION PART II - AD-783 408	
a. Program Flowcharts .....	5-1
(1) GENO .....	5-1
(2) GEN1 .....	5-4.2
(3) GEN1A .....	5-13
(4) GEN2 .....	5-14
(5) GEN2X .....	5-44
(6) GEN4X1 .....	5-45

	PAGE
(7) GEN4X2 .....	5-51
(8) GEN4X3 .....	5-52
(9) GEN3A .....	5-55
(10) GEN3 .....	5-73
(10.1) GEN3B .....	5-108.1
(11) GEN4 and GEN4A .....	5-109
(12) GEN5 .....	5-120
(13) GEN5A .....	5-121
(14) GEN6 and GEN6A .....	5-125
b. Program Narrative	
(1) GEN0 .....	5-138
(2) GEN1 .....	5-139.2
(3) GEN1A .....	5-149
(4) GEN2 .....	5-150
(5) GEN2X .....	5-185
(6) GEN4X1 .....	5-186
(7) GEN4X2 .....	NONE
(8) GEN4X3 .....	5-195
(9) GEN3A .....	5-199
(10) GEN3 .....	5-216
(10.1) GEN3B .....	5-264.1
(11) GEN4 and GEN4A .....	5-265
(12) GEN5 .....	5-285
(13) GEN5A .....	NONE
(14) GEN6 and GEN6A .....	5-288
(15) GEAA .....	5-311
(16) GEAB .....	5-314
(17) GEAC .....	5-315
(18) GEAD .....	5-136
(18.1) GEAE .....	5-136.1
(19) GEAG .....	5-317
(20) GEAL .....	5-319
(21) GEAM .....	5-320
(22) GEAN .....	5-321
(23) GEAP .....	5-322
(24) GEAS .....	5-324
(25) GEAT .....	5-325
(26) GEAX .....	5-326
(27) GEAZ .....	5-328
ENCLOSURE A - USER-WRITTEN SUBROUTINES .....	A-1
ENCLOSURE B - SPECIAL OPERATORS AND CONVERT ROUTINES	
1. Circle Search (CIR2SP) .....	B-1
2. Polygon Search .....	B-13
3. Route Search Conversion Subprogram (RTCVS) ...	B-19
4. Route Search Special Operator .....	B-24
5. Date Conversion Subprogram (CDATS) .....	B-26
6. Coordinate Conversion Subprogram (CRDFS) .....	B-40
7. Coordinate Conversion Subprogram (CRD6S) .....	B-41
8. Coordinate Conversion Subprogram (CRDGS) .....	B-41

	PAGE
9. Coordinate Conversion Subprogram (CRD7S) .....	B-43
10. Country Code Conversion Subprogram (CTY1S) .....	B-43
11. Comparison of Mark III and MIIMS Geographic Operators and Convert Routines .....	B-44
12. Route Search Special Operator (RTS3X) .....	B-45
13. Route Search Conversion Module (RTC3X) .....	B-57
ENCLOSURE C - ANCILLARY SYSTEM ROUTINES	
Section 1 - IBM	
1. ABGET .....	C-1-1
2. CALLIO .....	C-1-1
3. COMABSY .....	C-1-2
4. COMALL .....	C-1-4
5. COMARAYS .....	C-1-7
6. COMLIST .....	C-1-9
7. COMNUMS .....	C-1-11
8. COMREC .....	C-1-13
9. DATESUB .....	C-1-14
10. EXPNSP .....	C-1-14
11. LINK .....	C-1-15
12. LMLOOK .....	C-1-16
13. LMTABGEN .....	C-1-21
14. LOAD .....	C-1-22
15. LOADTAB .....	C-1-23
16. MOCHA .....	C-1-24
17. MOVALF .....	C-1-27
18. MOVCMF .....	C-1-29
19. MOVCON .....	C-1-31
20. MOVNUM .....	C-1-32
21. MOVRAY .....	C-1-34
22. MUVE .....	C-1-34
23. OPR34 .....	C-1-36
Section 2 - Honeywell	
1. BIBCS .....	C-2-1
2. COMLST .....	C-2-4
3. COMRAY .....	C-2-7
4. MOVNUM .....	C-2-11
5. MOVPAK .....	C-2-14
6. MOVRAY .....	C-2-16
7. OPR34 .....	C-2-18
8. PUTPSC .....	C-2-20
9. RDPSCS .....	C-2-22
10. WTPSCS .....	C-2-24
11. YYDDD .....	C-2-26
ENCLOSURE D - Honeywell Differences	
1. File Structuring .....	D-1
2. File Maintenance .....	D-1
3. Logical Maintenance .....	D-5
4. Special Operators .....	D-7
5. Retrieval and Output .....	D-8

## User-written Subroutines

1. The function of user-written subroutines in MIDMS is to permit actions to be performed which cannot easily be defined in the normal MIDMS languages. There are three distinct types of user-written subroutines in MIDMS: conversion subroutines, validity checking subroutines, and special operator subroutines. Once they are written, these subroutines can be referenced in File Maintenance, Retrieval, and Output by only their name and operands. However, writing these subroutines requires knowledge of the calling sequence by which data will be passed between MIDMS and the subroutine.

2. Each type of subroutine uses a single parameter (seven subparameters) calling sequence, containing the same fields but whose contents differ somewhat depending on the category of subroutine. The COBOL Data Division statements for these seven subparameters are as follows. While the field names may be different in every program (MIDMS programs and user-written subroutines), the identical field sequence, length, and PICTURE must be used. IN-DATA and OUT-DATA may be coded as groups with subfields for significant data items, but each of these areas must contain exactly 360 characters. In some machines, the term "COMPUTATIONAL" may have to be modified, so that IN-LENGTH, OUT-LENGTH, and EXIT-FLAG will be in binary form. The Honeywell 600/6000 series compilers use the term "COMPUTATIONAL-1" instead of "COMPUTATIONAL".

01	USER-CALLING-SEQUENCE.		
02	IN-LENGTH	PICTURE S9(6)	COMPUTATIONAL.
02	OUT-LENGTH	PICTURE S9(6)	COMPUTATIONAL.
02	EXIT-FLAG	PICTURE S9(6)	COMPUTATIONAL.
02	IN-DATA	PICTURE X(360).	
02	OUT-DATA	PICTURE X(360).	
02	FIELD-NAME	PICTURE X(8).	
02	FILE-NAME	PICTURE X(8).	

3. Definitions of the kinds of user-written subroutines and their methods of handling the seven subparameters are as follows:

a. Conversion Subroutines. File Maintenance, Retrieval, and Output use conversion subroutines to change data from one format to another. The seven subparameters used for passing information between MIDMS programs (or MIDMS-generated logic packages) and user-written conversion subroutines are:

(1) Input Length (IN-LENGTH) - a binary field containing the actual size of the significant portion of the Input Data field. The values of this field is inserted by the calling (MIDMS) program.

ENCLOSURE A

(2) Output Length (OUT-LENGTH) - a binary field containing the actual size of the significant portion of the Output Data field. The value of this field is inserted by the conversion subroutine.

(3) Exit Flag (EXIT-FLAG) - a binary field with data supplied by the subroutine for indicating to the calling program whether or not proper information has been given to the subroutine. The legal values for the Exit Flag field are:

- 1 for normal exit
- 2 for data error exit

(4) Input Data (IN-DATA) - a field containing 360 characters. The significant portion of this field is left justified in the area. The number of significant characters is specified to the subroutine by the Input Length field. The data in this field is inserted by the calling program.

(5) Output Data (OUT-DATA) - a field containing 360 characters. The significant portion of this field is left justified in the area. The number of significant characters is specified to the MIDMS program by the Output Length field. The data is inserted by the subroutine. The Retrieval and Output compilers permit conversion subroutines to produce an error message of up to 62 characters when parameters on query and report statements are not in the proper format. To indicate that an error message is present, the conversion subroutine must set Exit Flag to 2, place the size of the message (0 for no message) in Output Length, and insert the message in the Output Data field. Output conversion errors located for the CMOVE operation will not produce any error messages -- the data will be outputted in an unconverted form when Exit Flag is 2.

(6) Field Name (FIELD-NAME) - an eight character area containing the mnemonic of the data file field being converted. This information is provided by the calling program.

(7) File Name (FILE-NAME) - an eight character field containing the name of the file for which data is converted. This information is provided by the calling program.

b. Validity Checking Subroutine. The Ordinary Maintenance portion of File Maintenance uses a validity checking subroutine in Input Processor for the purpose of examining an input data field to determine if its format or value is in accordance with the programmed criteria. Its use is specified by the SUBCHK parameter of the field card of a Data Source Description (DSD). The seven subparameters used for passing information between File Maintenance Input Processor (FMIP) and user-written validity checking subroutines are:

ENCLOSURE A



(1) Input Length (IN-LENGTH) - a binary field containing the actual size of the significant portion of the Input Data field. The value of this field is inserted by FMIP.

(2) Output Length (OUT-LENGTH) - a binary field which is not applicable to checking subroutines.

(3) Exit Flag (EXIT-FLAG) - a binary field with data supplied by the subroutine indicating to FMIP whether or not Input Data has valid information. The legal values for Exit Flag are:

- 1 for valid data
- 2 for invalid data

(4) Input Data (IN-DATA) - a field containing 360 characters. The significant portion of this field is left justified in the area. The number of significant characters is specified to the subroutine by the Input Length field. The data in this field is inserted by FMIP.

(5) Output Data (OUT-DATA) - a 360 character field which is not applicable to checking subroutines.

(6) Field Name (FIELD-NAME) - an eight character area containing the mnemonic of the data file field being checked. The data is inserted by FMIP.

(7) File Name (FILE-NAME) - an eight character field containing the name of the file for which data is being checked. This information is provided by FMIP.

c. Special Operator Subroutines. Retrieval uses a special operator subroutine to determine whether or not a data field in a MIDMS file has some relationship to the information specified in the Retrieval statement as a search parameter. This "relationship" is determined by whatever has been programmed into the subroutine and either a hit or no hit condition is found, based on the criteria incorporated into the special operator subroutine. Subroutines may send "useful information" from their calculations back to Retrieval for eventual printing during Output processing. The seven subparameters used for passing information between Retrieval and user-written special operator subroutines are:

(1) Input Length (IN-LENGTH) - a binary field containing the size of the field from the MIDMS file which is being examined. The value of Input Length is inserted by Retrieval.

(2) Output Length (OUT-LENGTH) - a binary field which is used for two different purposes:

(a) Upon entrance to the special operator subroutine, Output Length is the number of characters in the search parameter

specified in the Retrieval statement. This value is provided by Retrieval.

(b) Upon completion of the subroutine, Output Length must be changed to the size of information to be transferred from the subroutine to Retrieval. If no data is passed, Output Length must be set to zero.

(3) Exit Flag (EXIT-FLAG) - a binary field with data supplied by the subroutine indicating to Retrieval whether the data field has the programmed relationship (i.e. hits) or does not have the relationship (i.e. does not hit). The legal values for Exit Flag are:

- 1 for hit
- 2 for no hit
- 3 for system error
- 4 for data error

The system error should be reserved for serious inconsistencies among the parameters because it will cause Retrieval to abort the job. The data error flag indicates that the file data value is incorrect and will cause Retrieval to print an error message specifying the location of the invalid data. If Output Length is not set to zero by the subroutine, an error description in Output Data will also be printed by Retrieval.

(4) Input Data (IN-DATA) - an area containing 360 characters. The data from the MIDMS field is left justified in this area. This information is inserted by Retrieval.

(5) Output Data (OUT-DATA) - a 360 character area which is used for two different purposes:

(a) Upon input to the special operator subroutine, Output Data contains the search parameter as it was specified in the Retrieval statement. This information is left justified in Output Data. The size of the search parameter is placed in Output Length. This information is inserted by Retrieval.

(b) Upon completion of the subroutine, Output Data becomes the information to be transferred from the subroutine to Retrieval. The subroutine must left justify the data in Output Data and place the size of the transferred data in Output Length. If no data is transferred, Output length must be set to zero. Output Length and Output Data are ignored when Exit Flag is set to 2 (no hit) or 3 (system error). When Exit Flag is 1, the information will be accepted by Retrieval only if a RECEIVE statement has been specified. When Exit Flag is 4, the information is treated as an error message and will be printed by Retrieval.

(6) Field Name (FIELD-NAME) - an eight character area containing the mnemonic of the data file field being queried. This information is provided by the Retrieval program.

(7) File Name (FILE-NAME) - an eight character area containing the name of the file which is being searched by Retrieval. This information is provided by Retrieval.

4. User-written subroutines may be programmed in COBOL, a combination of COBOL and FORTRAN, or assembly language.

a. COBOL.

(1) For the IBM 360, a COBOL subroutine must have the Data Division statements specified in paragraph 2 placed within the LINKAGE SECTION. Any additional work areas needed for the COBOL subroutine are to be placed in the WORKING-STORAGE SECTION. In addition to the LINKAGE SECTION, an IBM 360 user-written COBOL subroutine must contain a series of standard statements to effect proper linkage with the calling program.

(a) In the Identification Division, the program identity should be composed of the subroutine's entry point with a "P" suffix (xxxxSP).

PROGRAM-ID. 'SUBRSP'.

(b) In the Procedure Division, the entry and return statements are as follows:

```
ENTER LINKAGE.  
ENTRY 'SUBRS' USING USER-CALLING-SEQUENCE.  
ENTER COBOL.  
....  
DATA-ERROR. MOVE 2 TO EXIT-FLAG.  
GO TO LEAVE-SUB.  
NORMAL-EXIT. MOVE 1 TO EXIT-FLAG.  
LEAVE-SUB. ENTER LINKAGE.  
RETURN.  
ENTER COBOL.
```

The entry point name (SUBRS in the example) is used within MIDMS as the reference to the subroutine. It must be exactly five alphanumeric characters, starting with a letter and ending with "S". In the example, DATA-ERROR and NORMAL-EXIT are used as paragraph names, thus implying that this particular subroutine is used as a conversion subroutine. There is no requirement that these paragraph names be used or that EXIT-FLAG be set in a separate paragraph from the processing which determines the value to be placed in EXIT-FLAG.

(2) For the Honeywell 600/6000, a COBOL subroutine must have the Data Division statements specified in paragraph 2 placed within the WORKING-STORAGE SECTION along with any additional work areas. In addition to the appropriate Data Division statements, a Honeywell 600/6000 user-written COBOL subroutine must contain a series of standard statements to effect proper linkage with the calling program.

(a) In the Identification Division, the program identity should be composed of the subroutine's entry point with a "P" suffix (xxxxSP).

PROGRAM-ID. SUBRSP.

(b) In the Procedure Division, the entry and exit statements are as follows:

```
ENTRY SUBRS USING USER-CALLING-SEQUENCE
                        GIVING USER-CALLING-SEQUENCE.
....
LEAVE-SUB.
EXIT PROGRAM.
```

The entry point name (SUBRS in the example) is used within MIDMS as the reference to the subroutine. It must be exactly five alphanumeric characters, starting with a letter and ending with "S". The parameter USER-CALLING-SEQUENCE must be specified twice because it contains both the input to the subroutine and the results from the subroutine and Honeywell COBOL requires separate identification of input and output parameters. The EXIT PROGRAM statement must be the last executed by the subroutine.

b. COBOL and FORTRAN. Due to the character orientation of the subroutine parameters, FORTRAN cannot directly handle the calling sequence provided by MIDMS. Therefore, the subroutine must be subdivided into two parts: a COBOL module to act as interface between MIDMS and the FORTRAN routine and a FORTRAN module for the mathematical processing requirements of the subroutine.

(1) COBOL. As far as MIDMS is concerned, the COBOL portion of the COBOL-FORTRAN subroutine is the entire subroutine. That is, all the requirements for a user-written COBOL subroutine specified in paragraph 4.a. must be followed by the COBOL module. Except for those rules, the detailed processing of the COBOL module will usually be oriented to character handling preprocessing and post-processing needs of the FORTRAN mathematical routines. This may include, but is not necessarily limited to, defining of fields within IN-DATA and OUT-DATA, converting fields between decimal digits and either binary or

floating point forms of the number, or the insertion of error messages into OUT-DATA (Retrieval and Output only) upon signal from the FORTRAN subroutine. To execute the FORTRAN subroutine, an IBM 360 COBOL program needs the following statements:

```
ENTER LINKAGE.  
CALL 'fortsub' USING p1 p2 ... pn.  
ENTER COBOL.
```

Honeywell 600/6000 COBOL uses only the following statement:

```
CALL forsub USING p1 p2 ... pn.
```

In the CALL statement, "fortsub" represents the name on the FORTRAN SUBROUTINE statement. No specific list of USING parameters is shown because it is completely dependent on the requirements of the FORTRAN program.

(2) FORTRAN. The FORTRAN portion of the COBOL-FORTRAN subroutine has virtually no specific requirements as far as MIDMS is concerned, except to perform whatever actions are needed to carry out its user defined functions and, together with the COBOL module, provide the information expected by MIDMS. The FORTRAN module must begin with a SUBROUTINE statement and terminate its processing with RETURN.

```
SUBROUTINE forsub (p1,p2,...,pn)  
...  
RETURN
```

The subroutine name "fortsub" is referenced by the COBOL module in the COBOL CALL statement. It cannot be the same as either the COBOL PROGRAM-ID or the COBOL program entry point name. Parameters p1,p2, ...,pn must have corresponding parameters in the USING list of the COBOL CALL statement. The number of parameters depends on the amount of information passed between the FORTRAN and COBOL modules of the combined subroutine.

(3) Figure A-1 shows the linkage structure between MIDMS and the COBOL-FORTRAN subroutine.

c. Assembly Language. Since assembly language is strictly machine dependent, its use must be severely limited to situations where COBOL programs are extremely inefficient in time or core storage or where the requirement to transfer the subroutine to another computer is considered remote. With these restrictions, the machine independent nature of the MIDMS can be continued throughout its applications.

ENCLOSURE A

(1) For the IBM 360 Operating System, an assembly language subroutine uses the standard linkage conventions:

(a) The entry point is normally derived from either a START or a CSECT statements. Since it is referenced in MILMS, this name must be five characters long, starting with a letter and ending with "S".

(b) Register 1 contains the address in core of a word containing the address of the first position of USER-CALLING-SEQUENCE. This latter address, to be referenced as HOP, must be offset by a certain number of bytes to locate each of the seven subparameters specified in paragraph 2.

1. IN-LENGTH is at location HOP.
2. OUT-LENGTH at HOP+4.
3. EXIT-FLAG at HOP+8.
4. IN-DATA at HOP+12.
5. OUT-DATA at HOP+372.
6. FIELD-NAME at HOP+732.
7. FILE-NAME at HOP+740.

(c) Register 13 contains the address of the save area in the MIDMS program for holding the contents of the registers upon entry to the subroutine. The registers must be restored before returning to MIDMS.

(d) Register 14 contains the address in MIDMS to which the subroutine will return.

(e) Register 15 contains the entry point address in the subroutine.

(f) For further information consult the IBM manuals on the 360 Operating System, especially "Supervisor and Data Management Services" (GC28-6646).

(2) For the Honeywell 600/6000, an assembly language (GMAP) subroutine uses the following standard linkage conventions:

(a) The entry point is derived from a SYMDEF statement. Since it is referenced in MIDMS, this name must be five characters long, starting with a letter and ending with "S".

ENCLOSURE A

(b) Register 1 contains the address in MIDMS to which the subroutine will return.

(c) The upper half of the second word beyond the address in register 1 contains the address of the first word of USER-CALLING-SEQUENCE. This latter address, to be referenced as HOP, must be offset by a certain number of words to locate each of the seven parameters specified in paragraph 2.

1. IN-LENGTH is the word at location HOP.
2. OUT-LENGTH is at HOP+1.
3. EXIT-FLAG is at HOP+2.
4. IN-DATA begins at word HOP+3.
5. OUT-DATA begins at HOP+63.
6. FIELD-NAME begins at HOP+123.
7. FILE-NAME begins at character 2 (bit 12) of HOP+124.

5. Under the IBM 360 Operating System all user-written subroutines are placed in load module form on MIDMS.APPLOAD, a partitioned data set used as a subroutine library. The following actions are required to place a subroutine on MIDMS.APPLOAD:

a. Compile the subroutine using LOAD option (assumed in COBOL and FORTRAN). For COBOL and FORTRAN, //SYSLIN DD specifies the data set (DSNAME=&LOADSET) for the object modules from the compiler; //SYSGO DD serves the same purpose for the assembler. The following Job Control Language (JCL) cards may be used to compile a COBOL subroutine:

```
//jobname JOB acct,prgmr-name,MSGLEVEL=1
// EXEC COBFC,PARM=LOAD
//COB.SYSLIN DD DSNAME=&LOADSET,DISP=(MOD,PASS),
// SPACE=(TRK,(50,50)),UNIT=2314
//SYSIN DD *
      COBOL Statements
/*
```

COBFC is a catalogued procedure which supplies most of the needed JCL for compiling COBOL. Compiling FORTRAN is similar except that FORTGC is the catalogued procedure and FORT is placed before SYSLIN.

ENCLOSURE A

```
//      EXEC   FORTGC,PARM=LOAD
//FORT.SYSLIN DD DSN=&LOADSET,DISP=(MOD,PASS),
//      SPACE=(TRK,(50,50)),UNIT=2314
//SYSIN DD *
//      FORTRAN Statements
/*
```

Assembler language subroutines need the following JCL for compilation (ASMFC is the catalogued procedure):

```
//      EXEC   ASMFC,PARM=LOAD
//ASM.SYSGO DD DSN=&LOADSET,DISP=(MOD,PASS),
//      SPACE=(TRK,(50,50)),UNIT=2314
//SYSIN DD *
//      Assembler Language Statements
/*
```

b. Link edit the subroutine. //SYSIMOD DD specifies the load module library MIDMS.APPLOAD. //SYSLIN DD is the input of &LOADSET, the object module data set from the compilers, and should have DISP=(OLD,DELETE) whenever more subroutines will be compiled during the same job. After SYSLIN should be the concatenated data set, // DD DDNAME=SYSIN to indicate that there are control cards for linkage editor. //SYSLIB DD specifies libraries to resolve calls produced by COBOL (SYS1.COBLIB) and FORTRAN (SYS1.FORTLIB).

```
//SYSIN DD *      requires three cards
//      ALIAS      xxxxS
//      NAME      xxxxSP
/*
```

The ALIAS xxxxS is the same as the entry point for the COBOL subroutine: xxxxSP is the PROGRAM-ID. For assembler language programs, ALIAS is the entry point of the subroutine and the xxxxSP form on the NAME card is for compatibility with COBOL subroutines. The JCL required to link edit a subroutine is as follows:

```
//      EXEC PGM=IEWL,PARM='XREF,LIST,LET',REGION=98K
//SYSPRINT DD SYSOUT=A
//SYSLIN DD DSN=&LOADSET,DISP=(OLD,DELETE)
//      DD DDNAME=SYSIN
//SYSIMOD DD DSN=MIDMS.APPLOAD,DISP=OLD,UNIT=2314,
//      VOLUME=SER=VOLUME
//SYSUT1 DD UNIT=(SYSDA,SEP=(SYSIMOD,SYSLIN)),
//      SPACE=(1024,(200,20))
//SYSLIB DD DSN=SYS1.COBLIB,DISP=SHR
```

ENCLOSURE A



```
//      DD DSN=SYS1.FORTLIB,DISP=SHR
//SYSIN DD *
      ALIAS SUBRS
      NAME  SUBRSP
/*
```

6. Under the Honeywell 600/6000 General Comprehensive Operating Supervisor (GCOS), user-written subroutines are placed in system loadable form on a dynamic user library, a random file used as a subroutine library.

a. A dynamic user library (sometimes identified by its file codes Q\* or \*\*) must be reconstructed for every new subroutine and cannot be maintained in random form. Therefore, each user should have one or more dynamic libraries, as required by the various MIDMS applications.

b. A dynamic user library can be used to hold user-written subroutines, MIDMS-supplied subroutines that serve the purpose of user subroutines but are widely applicable (such as coordinate conversion or geographic special operator subroutines), user defined lookup tables, and single file logic packages.

c. The same dynamic user library must contain any of the above types of members needed during the execution of a single activity because only one \*\* file is permitted at any time. This means that:

(1) The \*\* file for procedures MLMG and MLMX must at least contain the conversion subroutine and tables referenced by the logic package.

(2) The \*\* file for the MFM procedure must at least contain the conversion and validity subroutines needed by Input Processor plus any single file logic package specified in Maintenance Proper plus the conversion subroutines and tables referenced by the logic package.

(3) The \*\* file for the MRTOP procedure must at least contain the conversion and special operator subroutines for Retrieval as well as the conversion subroutines and tables for Output.

d. The following actions are required to place a subroutine into a dynamic user library:

(1) Compile the subroutine using the DECK option (assumed in COBOL, FORTRAN, and GMAP but explicitly required for the FORTY card referencing the Series 6000 FORTRAN compiler). Each object deck will be output on file code C\*. Ordinarily, a permanent file should

ENCLOSURE A

be specified for each C\* file so that the dynamic user library could be built during the same job. By omitting a C\* Job Control Language (JCL) card, the object decks will be punched and a separate job will be required to incorporate the subroutine module(s) into the dynamic user library. The following JCL cards may be used to compile one module of a subroutine:

```

$      COBOL (or FORTRAN or GMAP or FORTY00DECK)
$      PRMFL  C*,W,S,user01/OBJECT/subrs
        compiler source deck

```

(2) After all subroutine modules have been compiled, it is necessary to construct an object library file containing System Editor (SYSEDIT) control cards, General Loader (GLOAD) control cards, and object deck images of the required subroutine modules. The normal GCOS method is to build and maintain an object library using the Object Library Editor. Since object deck images stored in permanent files cannot be referenced by \$ SELECT cards when using the GCOS FILEDITT program, a special MIDMS object library generator program (OBGEN) will be used. The procedure MIDMS/PROCLIB/MOB contains the JCL needed to execute the OBGEN program.

(a) The following is the normal deck set up for the MOB procedure:

```

$      SELECT  MIDMS/PROCLIB/MOB
$      DATA   CD,,COPY
        object library control cards
$      ENDCOPY

```

(b) The object library control cards input by file code CD consist of groupings of SYSEDIT and GLOAD control cards and object decks necessary to construct a dynamic user library. \$ SELECT cards may be used to reference existing card image files which already contain some of the necessary control cards or object cards. The SELECTed files are copied onto the object library file by OBGEN. Since the GCOS Generalized Input (GEIN) program (which normally interprets control cards) does not examine \$-cards appearing after a \$ DATA COPY card, use of \$ SELECT files is limited by OBGEN to two levels (i.e., the CD input file can contain a \$ SELECT for a file which also contains a \$ SELECT). There is no limit on the number of \$ SELECT cards in the input deck or in first level selected files.

(c) Each user-written subroutine to be placed in the dynamic user library will normally be defined with the following control cards. The \$ SYSLD card includes the name used in MIDMS to reference the subroutine.

ENCLOSURE A

```

$      SYSLD   CATALOG=subrS,RELOC,MASTER
$      OPTION  NOSETU,NOFCB,SYMREF
$      LOWLOAD
$      SELECT  user01/OBJECT/subrs
(other $ SELECT cards or object decks needed to form
a loadable subroutine)
$      ENTRY   subrS (entry point name in program,
                      normally same as SYSLD catalog
                      name)
$      EXECUTE
$      ENDL

```

(d) Old object libraries saved in permanent files can be used in the construction of a new object library. Each can be referenced by a \$ SELECT card since SYSEdit control cards, GLOAD control cards, and object deck images are already contained on the separate object libraries. It is the user's responsibility to insure that the collection of all items on the new object library, from whatever source, does not contain any duplicate names (SYSLD catalog names).

(e) A full description of the capabilities of the MOB procedures is provided in Chapter 2 of DIAM 65-9-11, "Machine Independent Data Management System (MIDMS) Job Control Language Procedures and Installation Guide."

(3) After the object library is built, the System Editor is used to construct a new dynamic user library.

(a) The following cards are used when the MOB procedure has previously been applied as described in paragraph (2):

```

$      SYSEdit INITIALIZE
$      FILE     R*,R1R
$      PRMFL    Q*,W,R,user01/DYNAMIC/filename

```

(b) The R\* file input to SYSEdit defines the object library produced in the MOB procedure. The Q\* file defines the dynamic user library as a permanent file under the user01 master catalog. This dynamic user library will be referenced as a \*\* file by MLMG, MLMX, MFM, or MRTOP procedures.

ENCLOSURE A

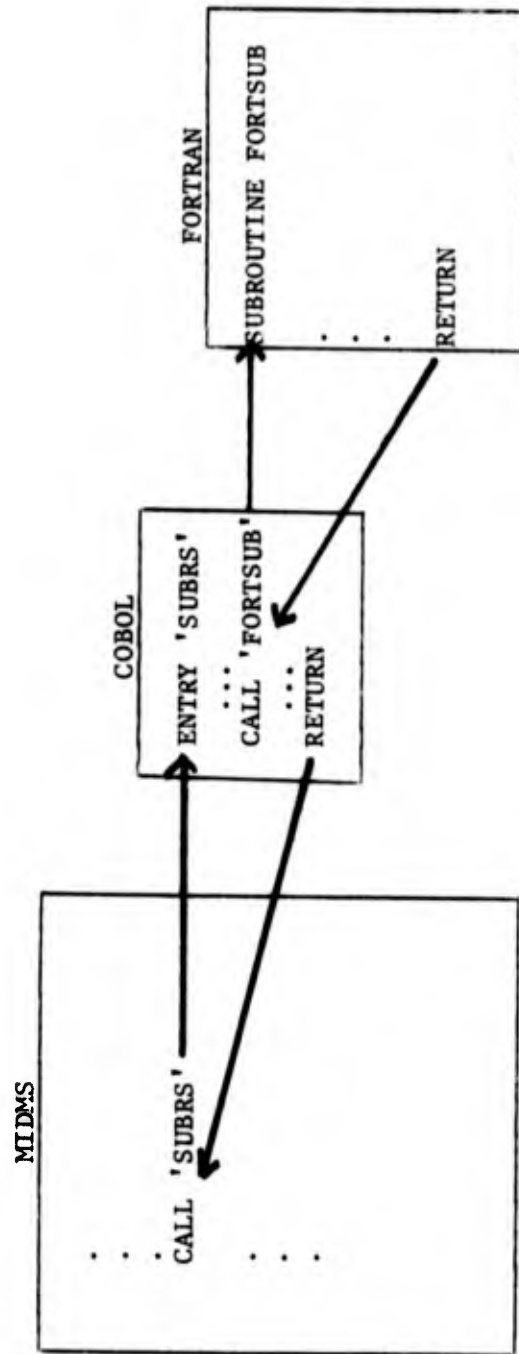


Figure A-1. MIDMS COBOL-FORTRAN Linkage Structure

## Special Operators and Convert Routines

### 1. CIRCLE SEARCH (CIR2SP).

#### a. Function.

(1) Given the coordinates of two geographic points determine if they coincide.

(2) Given the coordinates of a point and a radius and the coordinates of a second point P, determine if P lies within or on the circle and if it does compute the distance and azimuth of P from the center of the circle.

(3) Given the coordinates of two points and two radii determine if the two circles intersect and if they do determine the distance and azimuth from the center of the search circle to the center of the file circle.

#### b. Definitions.

(1) Distance - All distances are in nautical miles.

(2) Azimuth - That angle formed by the rays extending from the center of the circle to both True North and the point, P. It is measured clockwise from True North and has a range, in degrees of 0 Azimuth Angle 360.

#### c. Description of the Input/Output Fields in the Linkage Section.

##### (1) In-Length.

(a) Input - Contains number of characters in the In-Data field.

(b) Output - Not applicable.

##### (2) Out-Length.

(a) Input - Contains number of characters in the Out-Data field.

(b) Output - Contains number of characters in the Out-Data field, equals 16 in the event of a "hit;" zero if there is not a "hit," and 50 + IN-LENGTH in the event of invalid file point coordinates.

ENCLOSURE B

(3) Exit-Flag.

(a) Input - Not applicable.

(b) Output - When two points coincide or a point, P, is inside or on the circle or when two circles intersect, then Exit-Flag contains a value =1. If the file point or circle does not "hit," the search point or circle Exit-Flag contains a value =2. If the file point coordinates are invalid then Exit-Flag contains a value =4.

(4) In-Data.

(a) Input - Can contain coordinates of a point or coordinates of a center of a circle and its radius. The format of a given pair of coordinates is: "Latitude, Longitude." The latitude and longitude may be given in either 15 character external format, 13 character internal format or 11 character internal format. If a radius is given it immediately follows the coordinates of the circle center, and if those coordinates are in 15 character format, the radius is given in 5 characters representing nautical miles and hundredths of a nautical mile, otherwise the radius is given in 4 characters representing nautical miles and tenths of a nautical mile. In all cases the decimal point is assumed after the third character in the radius field.

(b) Output - Not applicable.

(5) Circ-Number.

(a) Input - Contains three digit identifier of user circle or point.

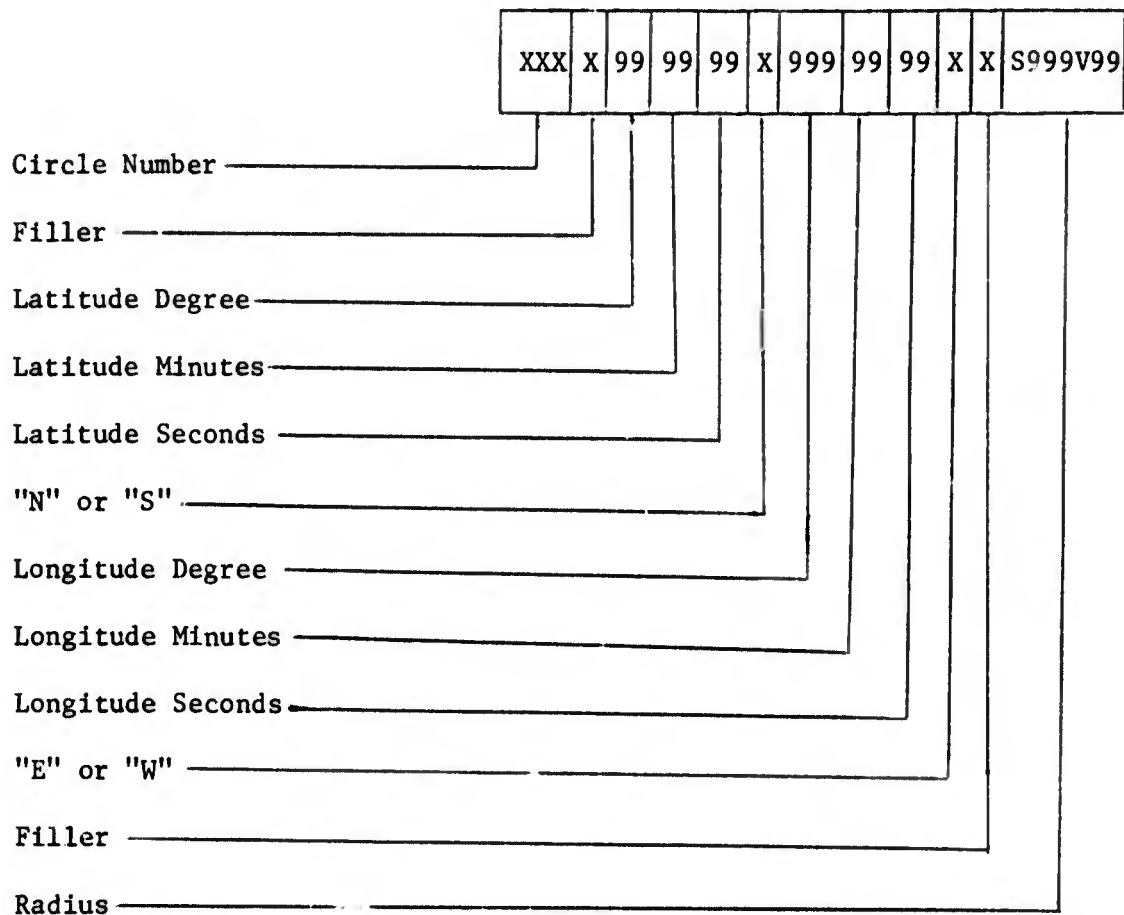
(b) Output - Contains three digit identifier of user circle or point.

(6) Out-Data.

(a) Input - Can contain coordinates of point to be searched or coordinates of center of circle and its radius. The format of a given pair of coordinates is "Latitude, Longitude." Both the latitude and longitude are represented with degrees, minutes, seconds and direction (N-S, E-W), with leading zeros where necessary. When the coordinates of the center of a circle are represented, then

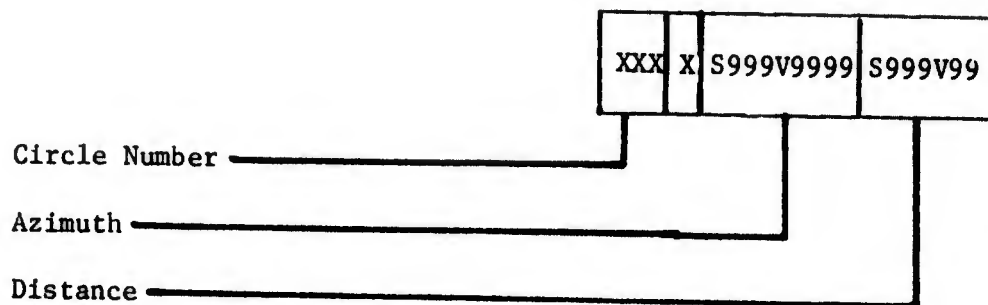
ENCLOSURE B

the radius immediately follows them, is five characters long and has an assumed decimal place between the third and fourth digit positions. Circle identifier, coordinates of center of circle, and radius are user provided and separated by commas.

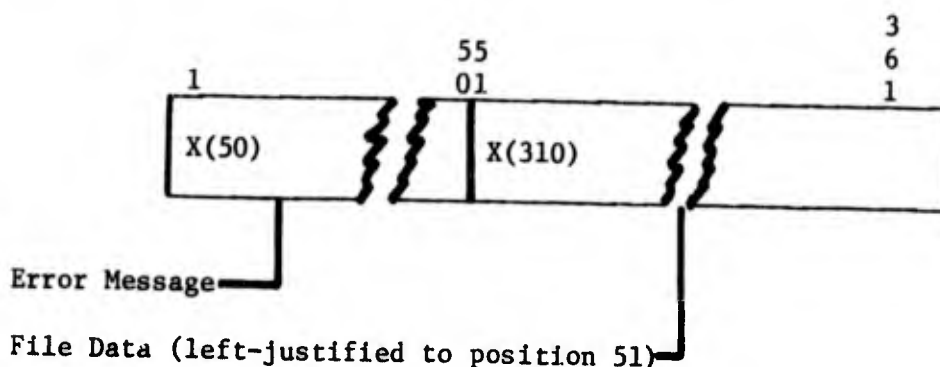


(b) Output - In the case of a point, P, inside or on a circle, then it contains the distance of P from the center of the circle and the azimuth of P. The azimuth is represented first, is seven characters long, and has an assumed decimal place between the third and fourth digit positions. Distance immediately follows the azimuth, is five characters long, and has an assumed decimal place between the third and fourth digit positions.

ENCLOSURE B



In the event of invalid file point coordinates OUT-DATA contains an error message and the file data.



#### d. Narrative Flow.

(1) The subprogram is called by Retrieval using the name "CIR2S." The format of the file data is determined and the file field is converted to binary with 4 decimal places (MAIN-PROGRAM, TEST-13, CASE-15). The coordinates of the search field are converted to the common format and coordinate system and a gross check is performed (GROSS-CHECK).

(2) In the event of a point vs point search the coordinates are compared for equality mod 360 (CHECK-LON). Otherwise the cosine of the average latitude of the circle center and point or circle center and the square of the distance between the above points are computed (CHECK-LON). If the square of the distance is less than or equal to the square of the total allowable radius then the distance is computed (CHECK-LON, SQ-ROOT) and the azimuth is computed (ARC-TANGENT).

(3) The special case of the point and circle center on the same meridian is analyzed separately (SPECIAL-CASE).

ENCLOSURE B



(4) A division error in computing the azimuth is flagged by setting Azimuth to 999 (ERROR-HIT).

(5) If the file point coordinates are invalid, an error message is moved to the output area along with the file coordinates (EDIT-CHK-EXIT).

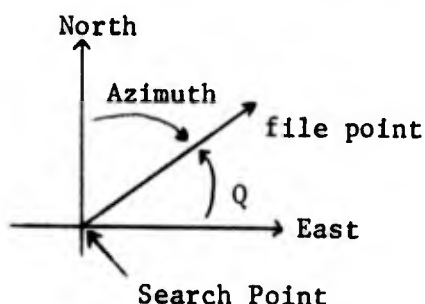
(6) Control is then passed back to Retrieval (ENTER-EXIT).

e. Limitations.

(1) Results are accurate if the radius of the circle or the sum of the radii of the two circles does not exceed 300 nautical miles.

(2) The circle or circles cannot contain either of the two poles.

Method of compute the Azimuth



The tangent of the angle of (FTANA) may be computed by:

$$FTANA = (INLAT-OUTLAT)/DELTA-LON * COSLAT$$

The problem is now to find the arctan of FTANA. The algorithm used in the IBM FORTRAN library subprogram IHCSATAN is used to invert the tangent function (of IBM System/360 FORTRAN IV Library Subprogram, GC28-6596-4).

The algorithm to computer Arctan (X) is as follows:

(a) Reduce to the case  $0 \leq x \leq 1$  by using

$$\arctan \left( \frac{1}{|x|} \right) = \pi/2 - \arctan |x|$$

ENCLOSURE B

(b) Reduce the case  $|X| \leq \tan 15^\circ$  by using  

$$\arctan (X) = 30^\circ + \arctan \left( \frac{\sqrt{3} X - 1}{X + 3} \right)$$

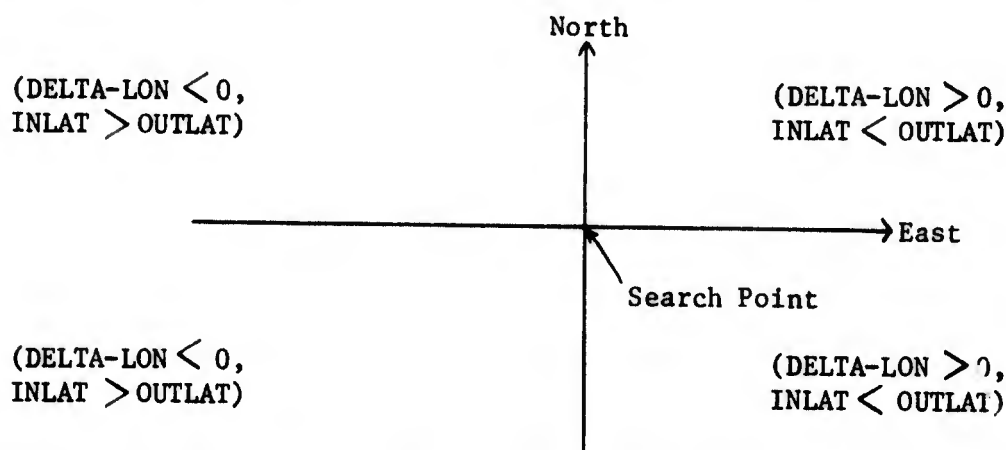
If  $\tan 15^\circ < x \leq 1$  then

$$\left| \frac{\sqrt{3} X - 1}{X + \sqrt{3}} \right| \leq \tan 15^\circ$$

(c) If  $|X| \leq \tan 15$  then

$$\arctan (X) = X \left( .60310579 - .05160454X^2 + \frac{.55913709}{X^2 + 1.4087812} \right)$$

Once the arctangent is found the azimuth is computed based on the quadrant of the file point.

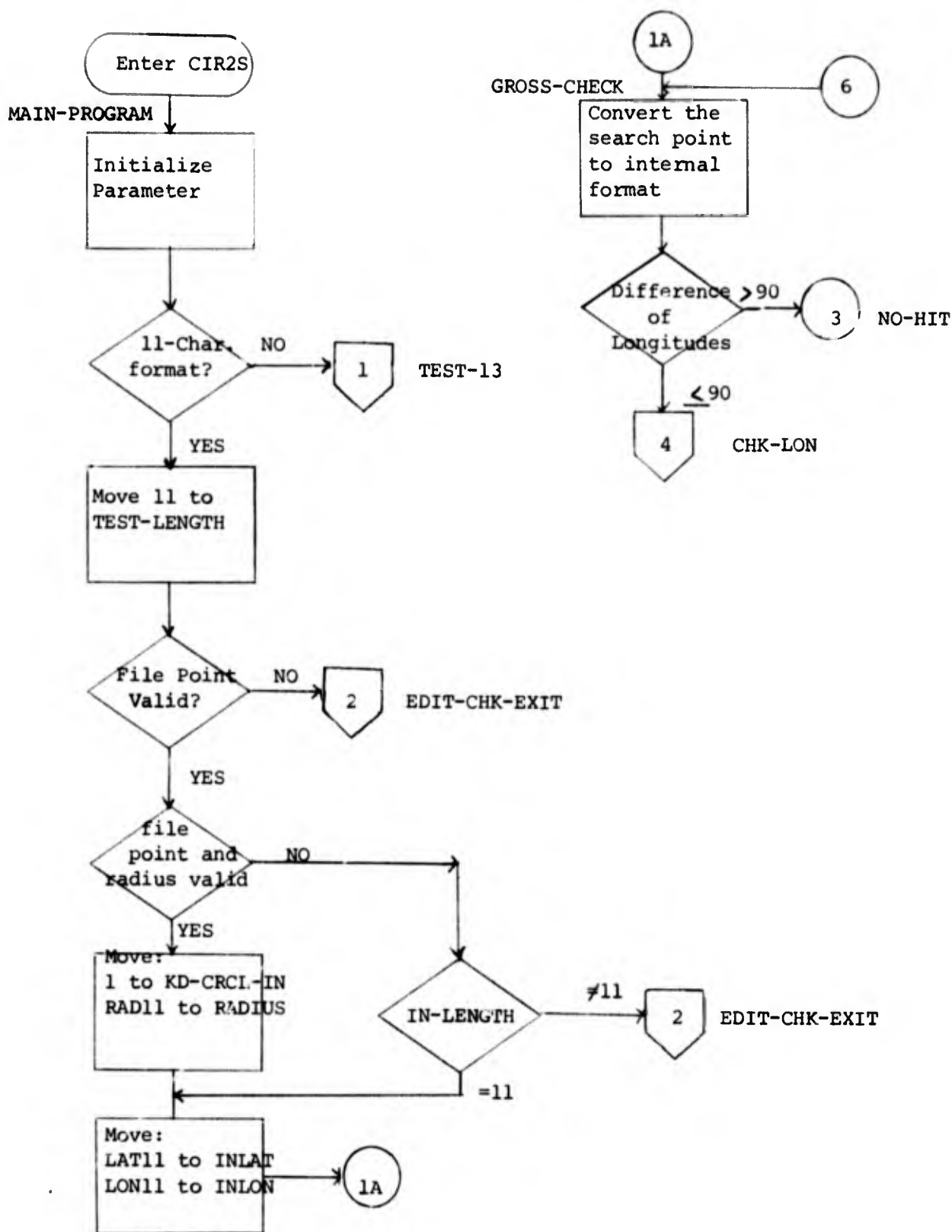


The parameter SYMBOL is used to keep track of the various cases.

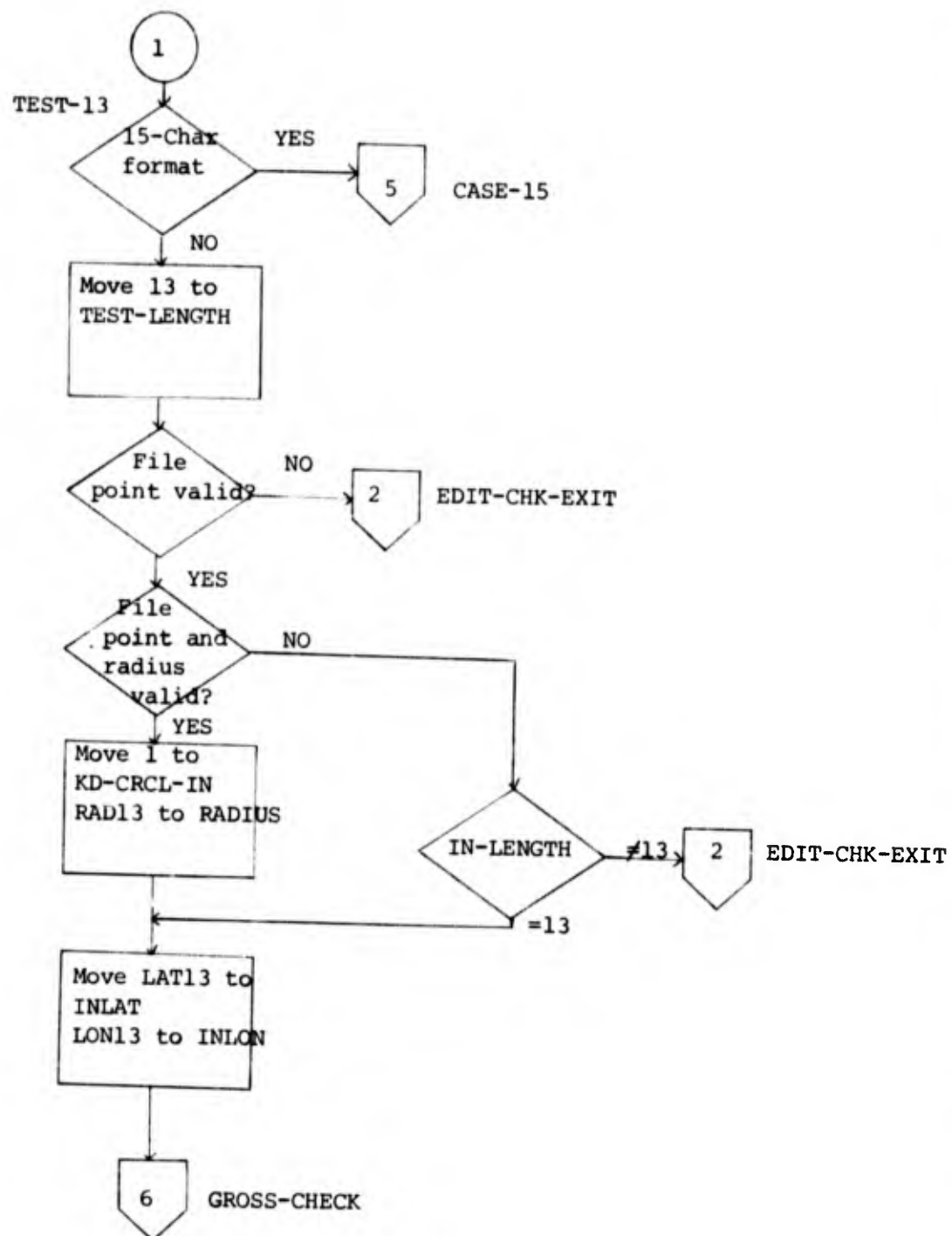
$$\begin{aligned} \text{SYMBOL} = & 0, |\tan \theta| \leq 1; \\ & 1, |\tan \theta| > 1; \\ & 2, |\tan \theta| > 1 \text{ and } \left| \frac{1}{\tan \theta} \right| > \tan 15^\circ \\ & 3, |\tan \theta| \leq 1 \text{ and } |\tan \theta| > \tan 15 \end{aligned}$$

(NOTE: If the search parameter is a point and the file figure is a circle then the azimuth is from the circle center to the search point.)

ENCLOSURE B

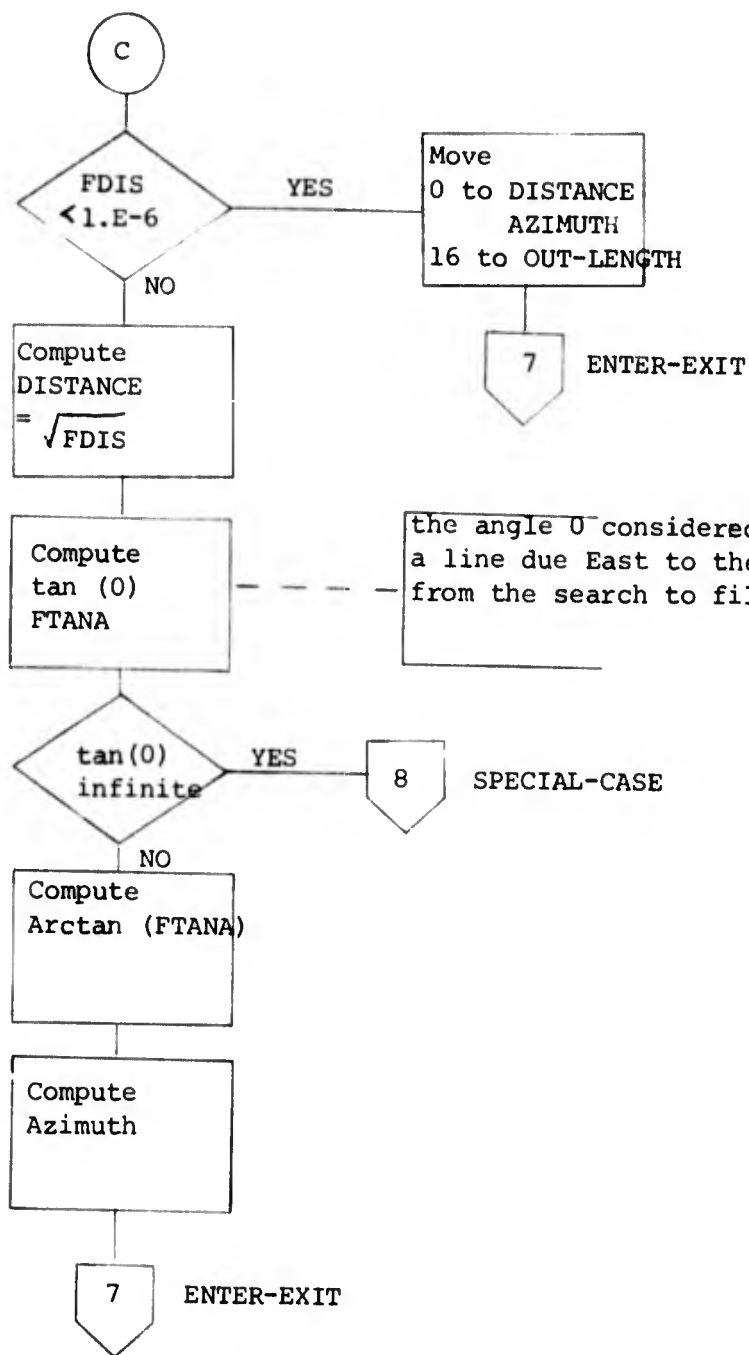


ENCLOSURE B

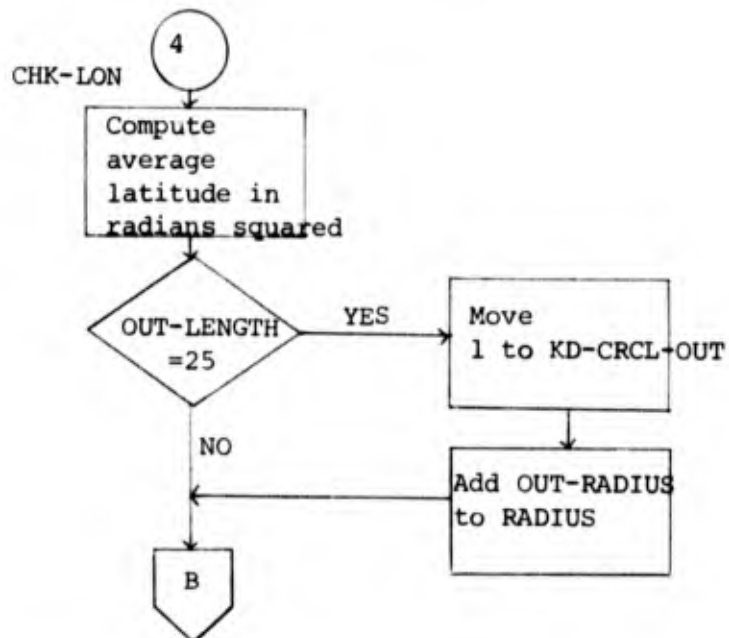
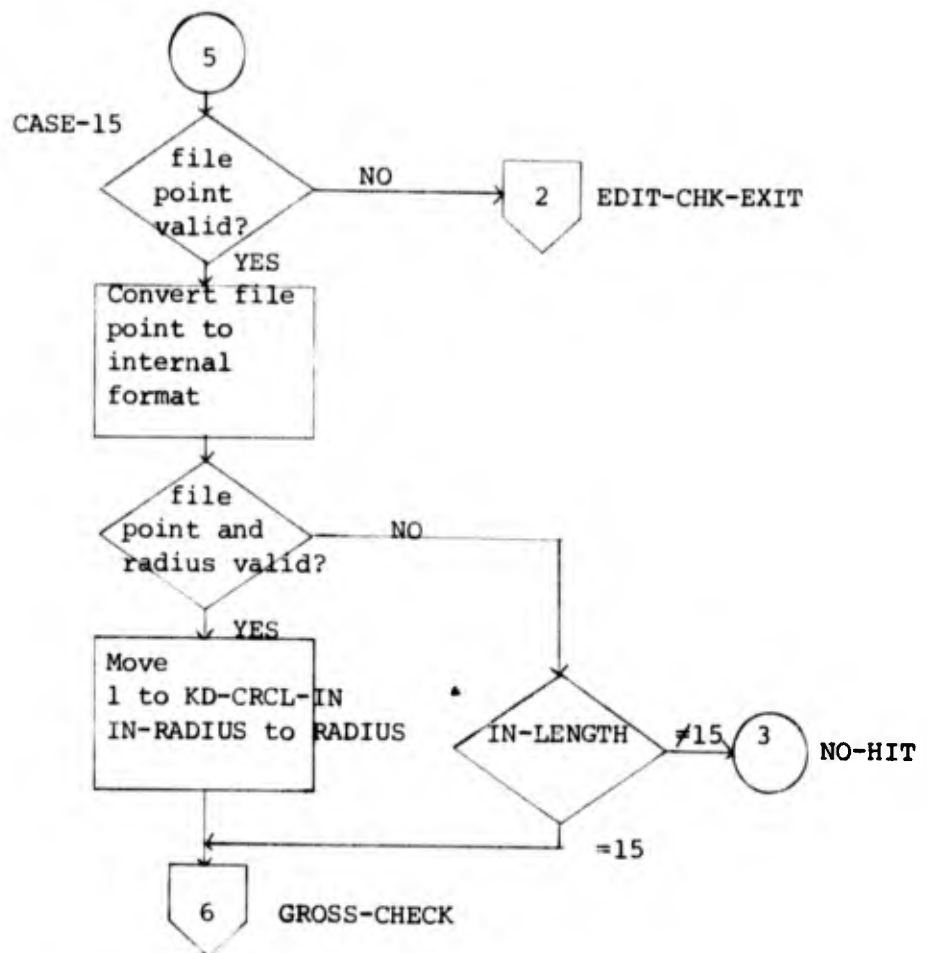


ENCLOSURE B

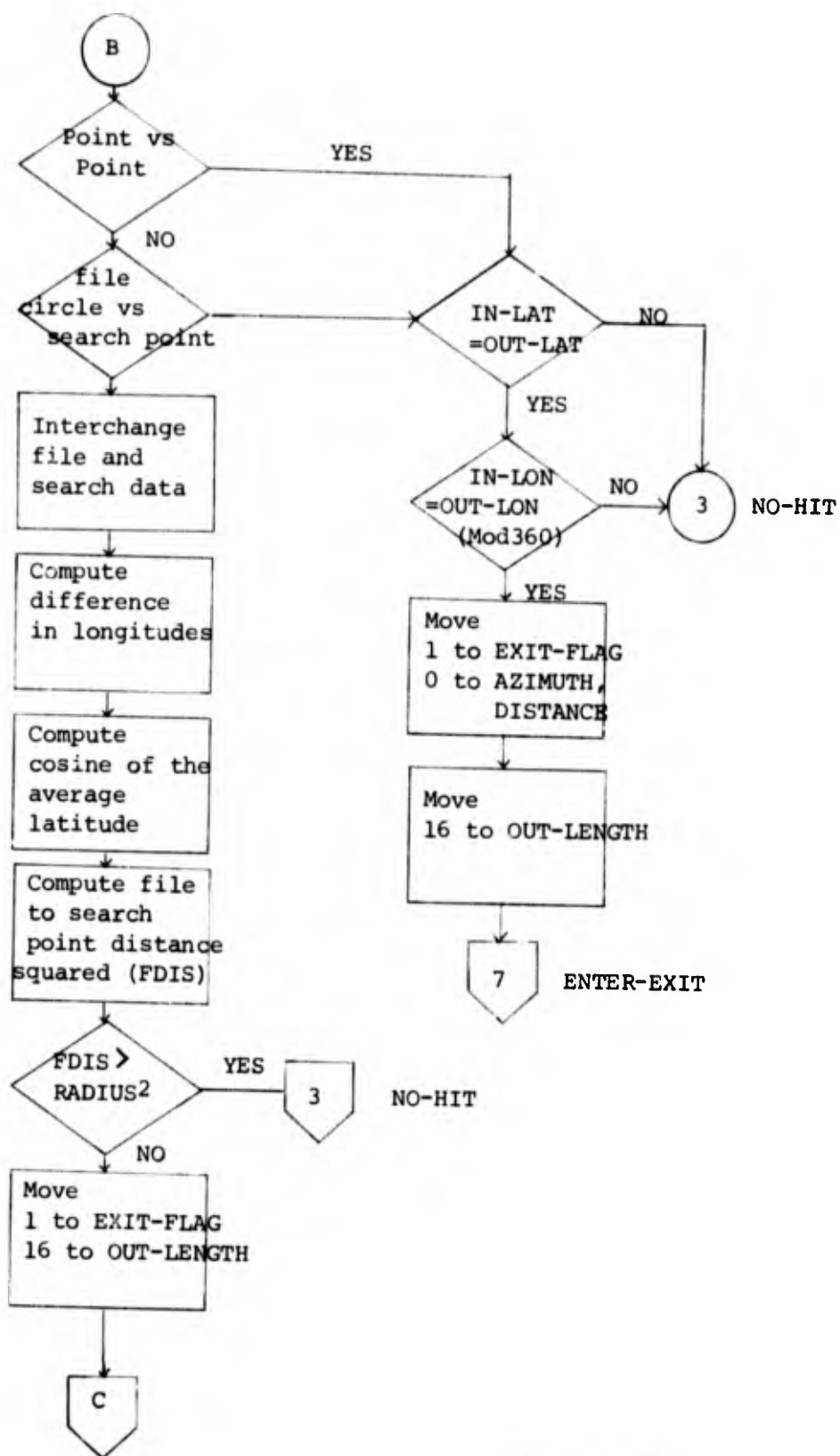
# ARC-TANGENT



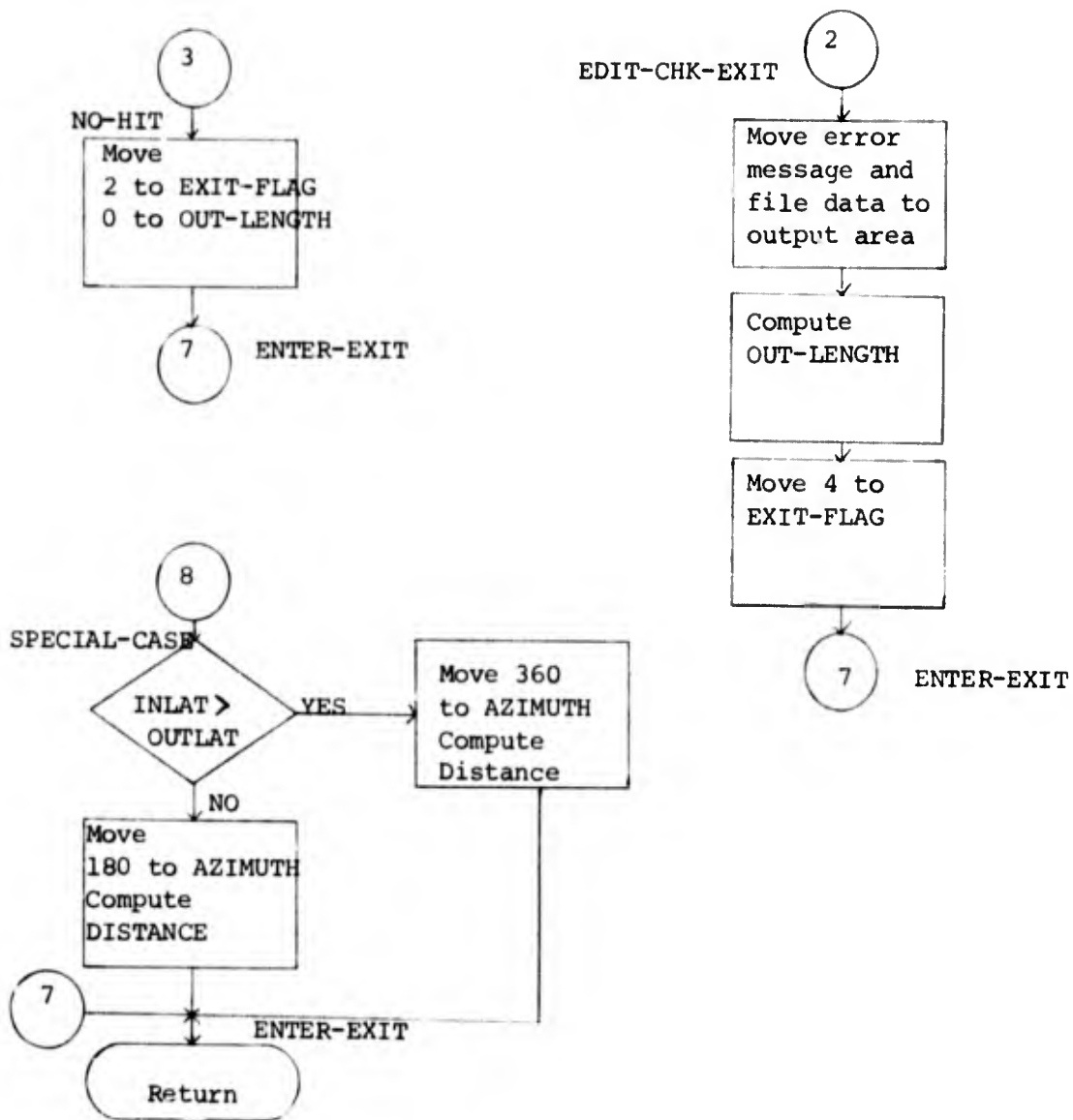
ENCLOSURE B



ENCLOSURE B



ENCLOSURE B



ENCLOSURE B



## 2. POLYGON SEARCH.

a. Function. Given the coordinates of the vertices of a polygon, then:

(1) Given the coordinates of a point P; then determine if P is inside or on the polygon.

(2) Given the coordinates of the vertices of a second polygon; then determine if the two polygons intersect.

(3) In paragraph 2.a.(2), either or both polygons may degenerate into a straight line, or point.

b. Description of the Input/Output Fields in the Linkage Section.

(1) In-Length.

(a) Input - Contains number of characters in the In-Data field.

(b) Output - Not applicable.

(2) Out-Length.

(a) Input - Contains number of characters in the Out-Data field.

(b) Output - Contains the number of characters in the OUT-DATA field, either 0 for a "miss," 3 for a hit or 143, 145, or 147 for an edit check, and 11, 13, or 15 character coordinates.

(3) Exit-Flag.

(a) Input - Not applicable.

(b) Output - When a point, P, is inside or on the polygon; when two polygons intersect; when a straight line and a polygon or two straight lines intersect; when a point and a line or two points intersect, then, Exit-Flag contains a value =1. If the search and file figures do not intersect (i.e., "miss") then EXIT-FLAG =2. If the file data is found to be invalid (i.e., an edit check) then EXIT-FLAG =4.

ENCLOSURE B

(4) In-Data.

(a) Input - Can contain coordinates of a point, or coordinates of vertices of a polygon, or coordinates of end-points of a straight line. The coordinates of a point may be given in 15 character external format, 11 character internal format or 13 character internal format. The coordinates must be contiguous, no commas or blanks to separate them; but with leading zeros where necessary. Input file provided.

(b) Output - Not applicable.

(5) Poly-Number.

(a) Input - Contains three digit identifier of user polygon or point.

(b) Output - Contains three digit identifier of user polygon or point.

(6) Out-Data.

(a) Input - Can contain coordinates of a point, or coordinates of vertices of a polygon, or coordinates of end-points of a straight line. The format of a given pair of coordinates is "Latitude, Longitude," both the latitude and longitude are represented with degrees, minutes, seconds, and direction (N-S, E-W), with leading zeros where necessary (15 character format). User provided with identifier and each pair of coordinates separated by commas.

(b) Output - In case of a "hit" or a "miss" it is not used. In case of invalid coordinates in the file, an error message and the invalid coordinates are returned in this area.

c. Narrative Flow.

(1) The subprogram is called by Retrieval using the name "POL2S." The file field is checked for blanks (CHECK-BLANKS). The format of the file coordinates is determined and the maximum number of vertices in the file polygon is determined (CONVERT-1, TEST-13, CASE-15). The file coordinates are validated and the number of file polygon vertices, which is the number of non-blank contiguous sets of coordinates, is determined (TEST-15A, TEST-11A, TEST-13A). A gross check for intersection is performed (B-11, B-13, B-15, GROSS-CHECK).

(2) The number of vertices in the user supplied search polygon is determined and validated (GROSS-CHECK, TEST1).

ENCLOSURE B

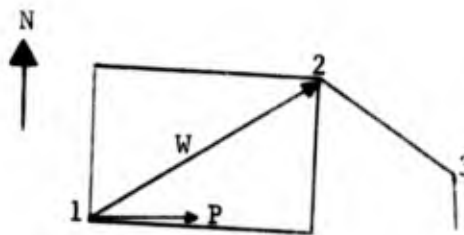
(3) The file coordinates are converted to a common format, binary with four decimal places (AAA, CONVERT-11, CONVERT-13, CONVERT-2). The search coordinates are converted to the standard format (AAA, CONVERT-3).

(4) In the event of a point vs point search the coordinates are tested for equality (AAA).

(5) A check is performed to see whether the In-Data and Out-Data fields will be compared for intersection of a Point-Line, Point-Polygon, Polygon-Polygon, Straight Line-Polygon, Straight Line-Straight Line, in each case initializing values appropriate for that part of the program (AAA).

(6) In the event of a point vs line search the point is tested for equality with an end point of the line. If this fails the vector-cross-product of the line and the vector from the first end point of the line to the point is computed. If the vector-cross-product = 0 then the point is considered to "hit" the line (POINT-LINE, VECTOR-X-PRODUCT, TEST-VCP).

(7) In the event of a Point-Polygon test, the Point is tested for intersection with the polygon by counting the number of times a line drawn from the Point to True North intersects the polygon (CONCAVE-POLYGON, ITERATE, EVALUTE). If the number of intersections is odd, the Point intersects the polygon; if the number of intersections is even, the Point does not intersect the polygon (EVALUATE). Vector Cross Products are used in parts of this testing process (VECTOR-X-PRODUCT), to determine if the point lies South of the side in question.



When the point P is within the square formed by extremes of latitude and longitude of the side in question, the cross product of the vector from vertex 1 to point P, V, and the vector from vertex

ENCLOSURE B

1 to vertex 2, W, is taken. If  $V_xW$  is positive then P lies south of the side  $\overline{12}$ . The special cases of a point lying on the same meridian with a vertex of a polygon, or a side of a polygon are also considered (SPECIAL-CASES, VERY-SPEC-CASE).

(8) In the event of a line versus line search the slope of the file line, CONST-IN, and the slope of the search line, CONST-OUT, are determined, then the longitude of the point of intersection of the two lines, X, is determined (SOLVING-FOR-X). It is then determined if X lies within the longitude boundaries of the two lines, i.e. that the lines intersect and not their extensions (CHECK-X).

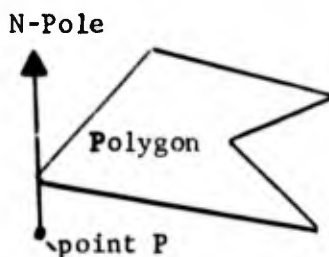
(9) In the event of a line versus polygon search, a test is made to determine if the first end point of the line is within the polygon (CONCAVE-POLYGON, ITERATE, POLY-IN-POLY). If no hit occurs in the first test, then the line is tested for intersection with the sides of the polygon (SOLVING-FOR-X, CHECK-X, BEFORE-IT).

(10) In the event of a polygon versus polygon search, a test is made to determine if vertex 1 of the file polygon is interior to the search polygon (CONCAVE-POLYGON, ITERATE, POLY-IN-POLY). If no hit occurs then the file and search polygons are interchanged and the first test repeated (POLY-IN-POLY). If no hit occurs then all pairs of one side from the file polygon and one side from the search polygon are tested for intersection (SOLVING-FOR-X, BEFORE-IT). The special cases of the line versus line tests are considered (DET-EQ-ZERO, A-NONZERO, PARALLELL-CASE, COLINEAR).

(11) Special cases which may occur in determining if the meridian from a point to the N-pole cuts a side of a polygon are:

(a) The longitude of the point equals the longitude of a vertex (SPECIAL-CASES). The two cases are shown below:

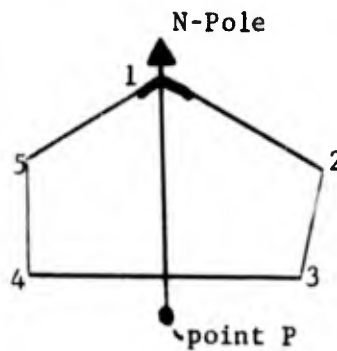
1.



Do not add to the number of intersections.

ENCLOSURE B

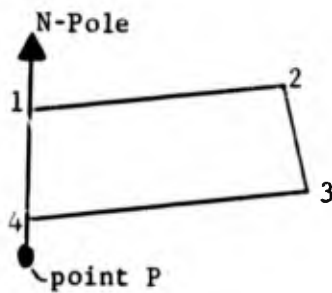
2.



Add 1 to the number of intersections when considering P versus side (1,2).

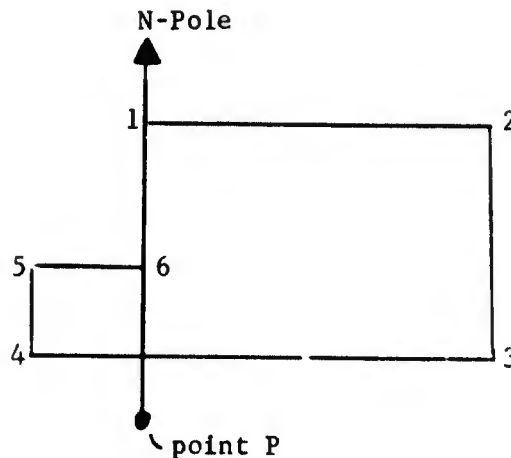
(b) A side of the polygon lies on a meridian through the point (VERY-SPEC-CASE). The two cases are shown below:

1.



In this case add zero to the number of intersections.

2.



In this case add 1 to the number of intersections, P versus side (6,1). (The meridian from P also intersects side (3,4); thus, the total number of intersections is even and P does not lie within the polygon.)

d. Limitations.

(1) Polygons may have interior angles greater than  $180^\circ$ , i.e. can be concave.

(2) Polygon legs cannot be more than 350 nautical miles in length, for accurate results.

(3) The difference between the longitudinal values of any two vertices of a polygon cannot be greater than  $45^\circ$ .

(4) The data base polygon cannot have more than 16 sides.

(5) The user polygon cannot have more than 16 sides.

(6) The polygon cannot contain either of the two poles.

(7) The vertices of the polygon are assumed to be connected in the order that they are read.

ENCLOSURE B

(8) The polygon cannot intersect itself.

(9) Accuracy decreases near the poles (error of approximately 3 miles at 68°N).

3. ROUTE SEARCH CONVERSION SUBPROGRAM (RTCVS).

a. Subsystem Structure. Not applicable.

b. Identification and Description of Subprogram included in the Subsystem.

(1) Abstract of subprogram.

(a) Function. This program is designed to construct a quadrilateral about a route, convert multiple formats into a standard format, and make data error checks on user supplied parameters. If no errors are encountered, data output from this subprogram may be used as input for the Route Search Special Operator.

(b) Calling Sequence, Parameters Received and Passed.

1. In-Length - Describes the number of characters of user supplied leg parameters (see In-Data below). The conversion subprogram utilizes in-length to identify the format in use.

2. Out-Length - Describes the number of characters generated by the subprogram and placed in the Out-Data field.

3. Exit-Flag - A value of "1" placed in Exit-Flag indicates information in Out-Data can be used as input to the special operator; "2" indicates an error and an associated error message is output.

4. In-Data - Parameters describing a leg of a route is input into In-Data. As a minimum, the user supplies the leg number, the starting and ending point of the leg, (in 15 character external format) and the left and right widths (directly or implicitly). Optionally, data to be used for frame or time analysis may be input. Data may be input as one of nine possible formats and may be described uniquely by in-length. Each of the nine formats begins with the leg number, and beginning and ending coordinates separated by commas. (See the documentation of RTC3S for a description of the possible input data formats, they are the same for both programs.)

5. Out-Data - This field, consisting of the conversion subprogram generated data, is used primarily as input to

ENCLOSURE B

the route search special operator. The following data is passed to the special operator; leg number, beginning and ending leg coordinates, conversion code, maximum and minimum widths, four corner points describing quadrilateral about a leg, code indicating option in effect, direction of the leg, cosine of average latitudes, and other supplemental data. The secondary purpose of this field is to pass back to the system error messages indicating errors in user supplied data.

6. Field Name and File Name - These fields are not used for the conversion subprograms.

(c) Narrative Flow and Capabilities. The entry point of the Conversion Program is "RTCVS." The data is checked for appropriate format; coordinates are examined for errors in size (numeric) (ERROR-TEST1, ERROR-TEST2, EXIT1, CONT-PROC). In-length is checked to determine which special case is applicable (CHECK-FMT, A8).

1. CASE 47 - Checks for format error are made, maximum widths (W-MAX) are characterized by "L" and "R" depending on whether that width is left or right. Further checks determine whether frame (CASE 59) or time correlation (CASE 60X) are to be used, otherwise control is passed to the CNVRT section (see below) (CASE-47,A9,A5).

2. CASE 43 - A format error check is made (CASE-43,C9,C1), followed by the computation of the left and right widths from the users supplied focal-length (in), altitude ( $10^3$  ft), and focal-plane (in) where "A" represents 4.5 in and "B", 9.0 (CAL-WIDTH). If frame analysis (CASE 55) or time correlation (CASE 56) are not to be used, control is passed to the CNVRT section (C2).

3. CASE 41 - Only one width is given. After a format error check is made, the width is determined left or right and 0 is placed in the field of the opposite direction (CASE-41). If frame analysis is to be used, a branch to CASE-53 is taken, and if time analysis is to be used, a branch to CASE-54 is taken (H1).

4. CASE 59, CASE 55, and CASE 53 - These cases are associated with frame analysis, values of interval (number of frames across), initial and terminal frames are passed to INTEVAL, INIT-FR, and TERM-FR of the special operator respectively. Control is then passed to the CNVRT section.

ENCLOSURE B

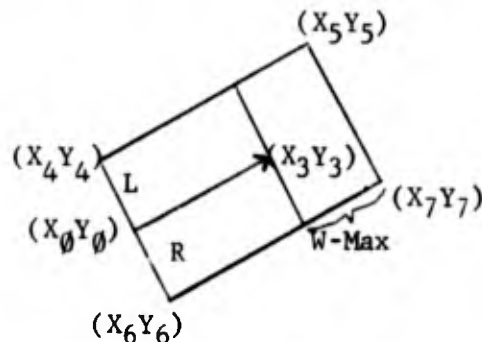


5. CASE 60X, CASE 56, CASE 54, and CASE 60 - Data is moved to appropriate field for use by the special operator for time correlation. One error check is made. Control is passed to the CNVRT section.

6. CNVRT - This section has several parts.

a. The first section CNVRT thru E6 converts coordinate data to decimal format according to one of two possible systems. System 1 makes all western longitude coordinates negative. System 2 converts all western longitudinal coordinates as 360 - longitude. Both systems consider southern latitudes as negative.

b. In E6 cosine is calculated for the average of the leg latitudes. The direction of the leg is determined and control passed to CNST - NORTH, SOUTH, EAST, or WEST, if a special case; or CNST-UNIV if routine. In each case, a quadrilateral is constructed and the corner points are left in out-data for the special operator. The quadrilateral is shown below:



Where:  $(X_0Y_0)$  is the initial point of the leg  $(X_3Y_3)$  is the terminating point of the leg; L is the distance to the left; R is the distance to the right; and W-Max is the greater of L and R. The coordinates  $(X_i, Y_i)$  for all i, are in nautical miles from the equator and the Greenwich meridian.

## (2) Capabilities.

(a) This subprogram can convert leg parameters from nine possible formats to one standard usable format for the route search special operator.

ENCLOSURE B

(b) Coordinates must be in standard 15 character format and are converted to decimal nautical miles.

(c) Implicit widths are calculated from data describing camera, focal length, and altitude, or through omission of one width in width specifications where the missing width is assumed to be zero.

(d) Time analysis requires beginning and ending times in hours, minutes and seconds.

(e) Coordinates are corrected for longitudinal squeeze.

(f) Data checks are made on user queries.

(3) Limitations

(a) All coordinates must be in standard 15 character format.

(b) Accuracy decreases when the leg or its associated data is more than 300 nautical miles.

(c) Leg coordinates cannot extend across the poles.

(d) Beginning and ending times are not continuous over the international date line nor through midnight.

(e) Number of leg is limited by the system (main program) to a maximum of 219 (depending on the format).

(f) Pitch of camera is assumed to be zero.

c. Error Messages.

(1) ERROR IN FORMAT SIZE.

(a) In-length is not the same value as required by the conversion program.

(b) Check for improperly imbedded blanks, missing commas, extra fields, and missing characters in the parameter field (field enclosed with quotations) of the query card.

(2) ERROR IN EVENT POINT COORDINATES.

(a) Coordinates are incorrect.

ENCLOSURE B

(b) Check longitudes for over 180°, latitudes for over 90°; insure characters for direction are correct and in their appropriate positions; check for alphabetic characters in numeric part of the field.

(3) ERROR IN LEG NUMBER.

(a) Leg number is incorrect.

(b) Insure leg number is a three character numeric field with no blanks or commas.

(4) SIZE ERROR.

(a) Probable attempt to divide by zeros has occurred (a rare occurrence).

(b) Check all equations with size error option, display inputs to these equations to determine cause of size error.

(5) ERROR WIDTH SPECIFICATIONS.

(a) Widths (explicit) are incorrect.

(b) Check format to insure that "L" and "R" are used appropriately; that preceding digits are numeric, and both widths are present.

(6) ERROR IN FOCAL WIDTH DATA.

(a) Widths (implicit) are not correct.

(b) Format of focal width is not correct or in cases where only one width is specified, that width is specified incorrectly.

(7) ERROR IN FRAME DATA, ERROR IN TIME DATA.

(a) Class (numeric and alphabetic) of data is incorrect.

(b) Format of focal width is not correct or cases where only one width is specified, that width is specified incorrectly.

(8) XX SIZE ERROR or YY SIZE ERROR.

(a) Probable error in numerator of complex equation.

ENCLOSURE B

(b) Check paragraphs I1 and I3 for data input.

4. ROUTE SEARCH SPECIAL OPERATOR.

a. Subsystem Structure. Not applicable.

b. Identification and Description of Subprogram included in Subsystem.

(1) Abstract of Subprogram.

(a) Function. The Route Search Special Operator passes a coordinate from a file against the data description of a leg to determine whether the coordinate falls within specified parameters. If it does, the coordinates's relative position to the leg, and its frame (optional) or its time (optional) is output with other supplemental data.

(b) Calling Sequence.

1. In-Length - For the special operator, in-length is either 11, 13, or 15 depending on the format of the file coordinates.

2. Out-Length - If the coordinate examined is hit, then a field of 25 characters is output. If the file data is invalid, an error message and the invalid coordinates are returned with a length of 65 characters.

3. Exit-Flag - The value 1 represents a hit, and the value 2 represents a no hit. The value 4 represents invalid file data.

4. In-Data - One 11, 13, or 15 character geographic coordinate is passed from the file to the program.

5. Out-Data - This field is identical to the out-data field of the conversion subprogram. Several data names are different; however, the two data fields are exactly alike. On output, in the event of a hit, a 25 character field or return values is placed in Out-Data. (See the user documentation and documentation RTS3S for a complete description.) In the event of invalid file data, a 50 character error message and the invalid coordinates are returned.

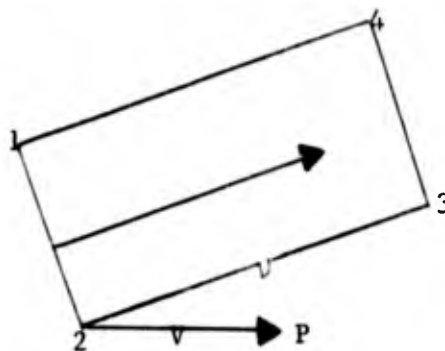
6. Field Name and File Name - The special operator does not utilize these fields.

ENCLOSURE B

c. Narrative Flow and Capabilities. The entry point for the special operator is "RTS2S."

(1) The file coordinate is tested for format (TEST-CONV-SYS, A2, TEST-13, CASE-15) and converted to a standard decimal format.

(2) The file coordinate is tested to determine if it is within the quadrilateral constructed about the route in the conversion subprogram. This test is performed by taking the vector cross product of the vector formed by the side of the quadrilateral and the vector formed by the line from the first vertex of the side in question and the file point. If the result is negative, the point is exterior to the quadrilateral and a "2" is moved to the exit-flag and control is returned to the system. If a hit, control is passed to NORTH-SOUTH-TEST. The process is illustrated below.



V is the vector from the corner No. 2 to the point P. U is the vector from corner No. 2 to corner No. 3. If the vector cross product,  $U \times V$ , is negative then V is to the right of U and therefore, P is not within the quadrilateral.

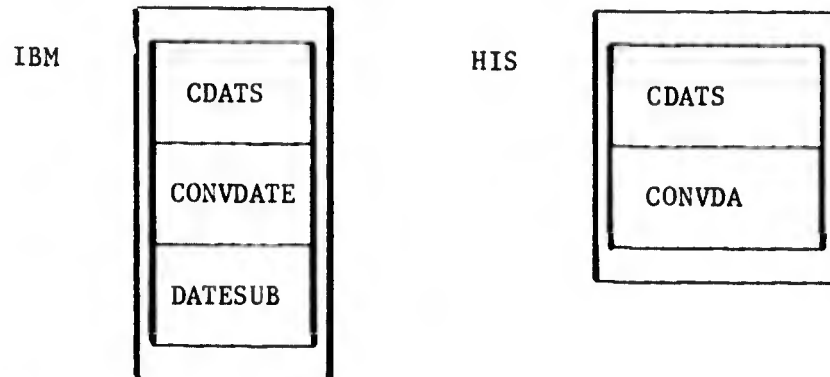
(3) NORTH-SOUTH-TEST thru NE-SE-FLIGHT. Flight direction is determined, distance down track is computed, distance left or right of route is calculated. If time or frame options are in effect (CHK-OPT), time down track (TIME OPTION) or the approximate frame (FRAME-OPTION) is calculated respectively, this generated data is placed in out-data and control is then passed to the system.

d. Error Message. In can the file data is invalid. The message: "FILE DATA INPUT TO ROUTE SEARCH IS INVALID" is returned.

ENCLOSURE B

## 5. DATE CONVERSION SUBPROGRAM (CDATS).

### a. Subsystem Substructure.



CDATS is the executive route and calls CNVTDT by any one of six possible entry points. CONVDATE calls DATESUB if the system date is required.

### b. Identification and Description of Subprogram included in Subsystem.

#### (1) CDATS

##### (a) Abstract of Subprogram.

1. Function. The conversion routine is made up of two COBOL programs. The first provides an interface between the MIDMS module that calls for the conversion and the second one which does the actual conversion. The first contains the standard user calling sequence, and determines, by examination of the first two characters of IN-DATA, which entry point is to be called in the second program.

2. Calling Sequence. It accepts the MIDMS standard user calling sequence composed of:

01	USER-CALLING SEQUENCE		
02	IN-LENGTH	PICTURE	S9(6) Computational.
02	OUT-LENGTH	PICTURE	S9(6) Computational.
02	EXIT-FLAG	PICTURE	S9(6) Computational.
02	IN-DATA	PICTURE	X(360).
02	OUT-DATA	PICTURE	X(360).
02	FIELD-NAME	PICTURE	X(8).
02	FILE-NAME	PICTURE	X(8).

ENCLOSURE B

3. Capabilities. The input is checked to insure that it is numeric. The routine then isolates the first two characters of IN-DATA which the user has specified as the type of input and type of output. The type of output can be specified as:

1	FOR	YYMMDD	(690320)
2	FOR	JULIAN	(69079)
3	FOR	DDMMYY	(200369)
4	FOR	ABRV.	(20 MAR 69)
5	FOR	MILITARY	(20 MARCH 1969)
6	FOR	CIVILIAN	(MARCH 20, 1969)

Type of input can be type 1, 2, or 3. In addition, the input type can be 0 which indicates that the current run date as generated by the system is to be used as the input type. Dependent upon the input code type, the appropriate entry point is called in the second program. Appropriate error checking is done to determine if the input date is within valid bounds. Depending on the output code the desired processing takes place.

4. Limitations. It is the user's responsibility to insure that the first two characters of the field specified by the CONVERT operator contain respectively, the input type and the output type. This can be done by the use of partial notation or in the case of Logical Maintenance by partial notation or GROUP DEFINE.

## (2) CONVDA

(a) Calling Sequence. (NOTE: This is HIS version.)

Entry Point CNONE using RCODE giving RETURN-DATE input is system data (input code 0).

Entry Point YYMMDD using RCODE YOUR-DATE giving CCODE RETURN-DATE input is YYMMDD (input code 1).

Entry Point JULIAN using RCODE YOUR-DATE giving CCODE RETURN-DATE input is Julian date (input code 2).

Entry Point MILDAT using RCODE YOUR-DATE giving CCODE RETURN-DATE input is DD~~MM~~ ~~MM~~YYYY (input code 5).

Entry Point CIVDAT using RCODE YOUR-DATE giving CCODE RETURN-DATE input is MM ~~MM~~DD, ~~MM~~YYY (input code 6).

Entry Point ABRDAT using RCODE YOUR-DATE giving CCODE RETURN-DATE input is DD~~MM~~~~MM~~YY (input code 4).

ENCLOSURE B

Date Code	General Format	Example
0	System date	
1	YYMMDD	730629
2	Julian YYDDD	73180
3	DDMMYY	290673
4	DD <del>MM</del> YY	29 JUN 73
5	DD <del>MM</del> ...M <del>YY</del> YY	29 JUNE 1973
6	MM...M <del>DD</del> , <del>YY</del> YY	JUNE 29, 1973

#### Parameters

RCODE	PICTURE	S9(5)	COMP-1.
CCODE	PICTURE	S9(5)	COMP-1.
YOUR-DATE	PICTURE	X(8).	
RETURN-DATE	PICTURE	X(20).	

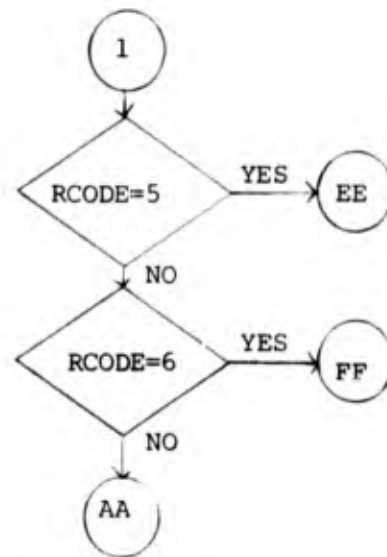
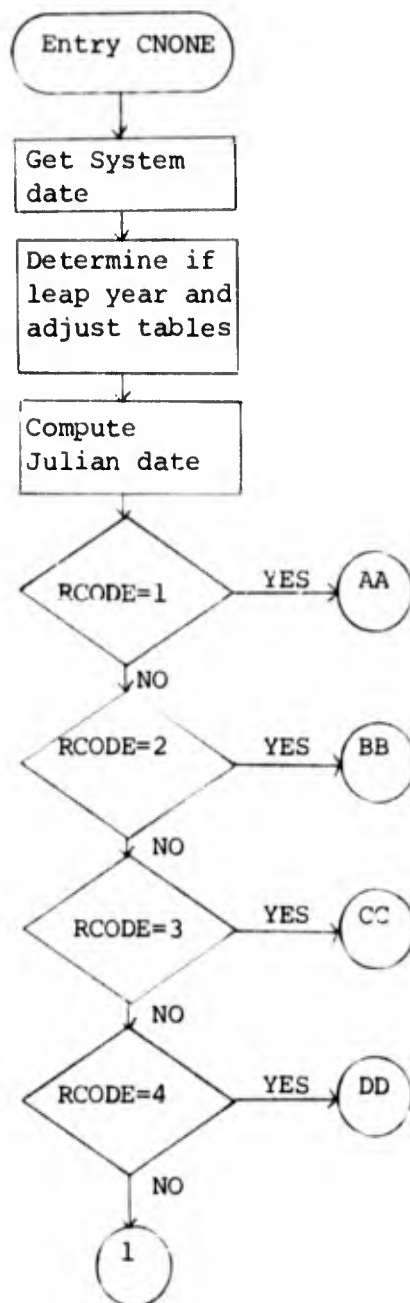
RCODE gives the desired format of the output; it range is 1, 2, 3, 4, 5, 6.

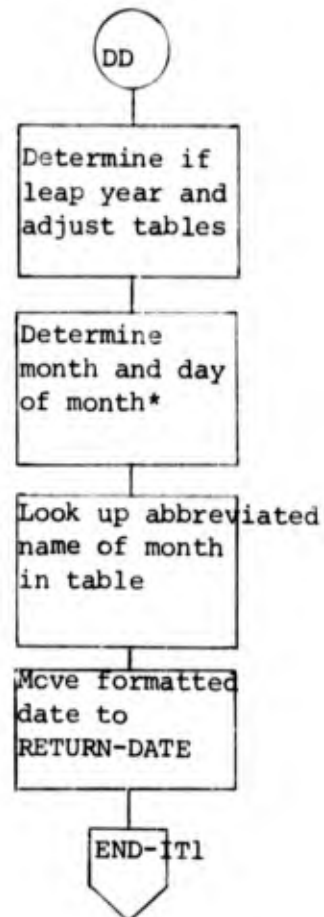
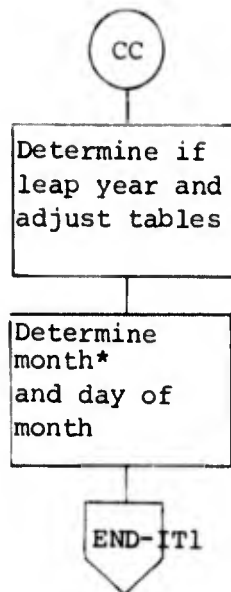
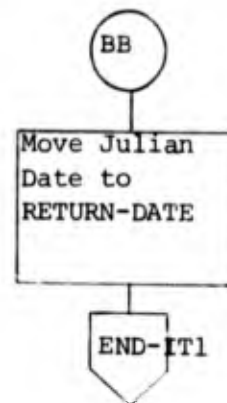
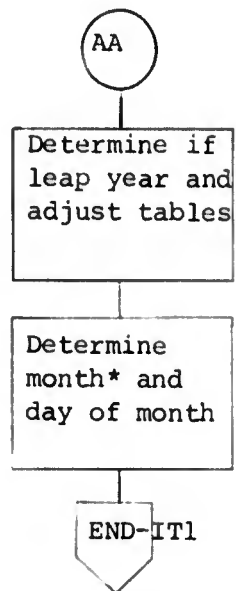
CCODE is zero if the input data is valid and is 9 if the data is invalid.

(b) Flow charts.

ENCLOSURE B

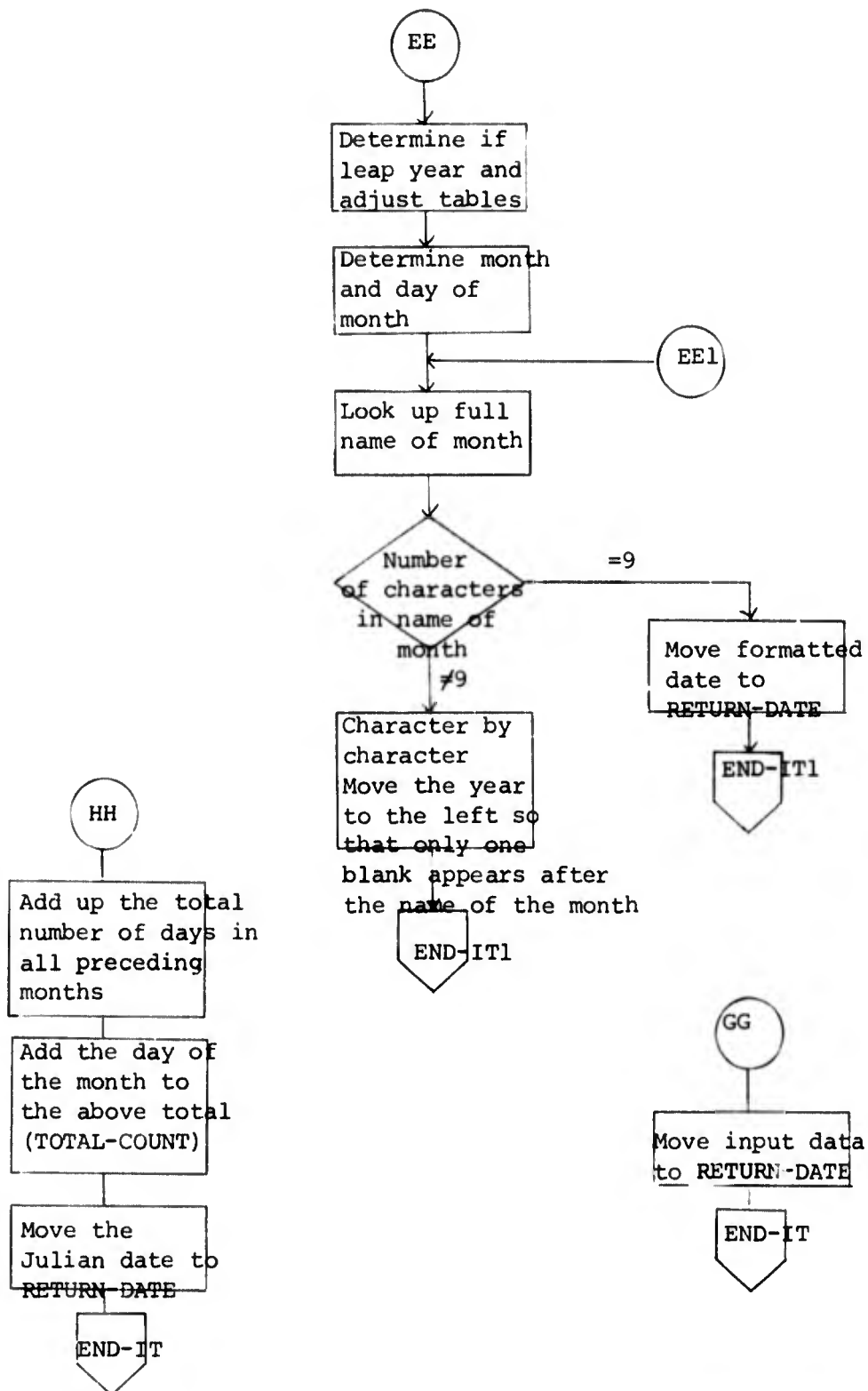




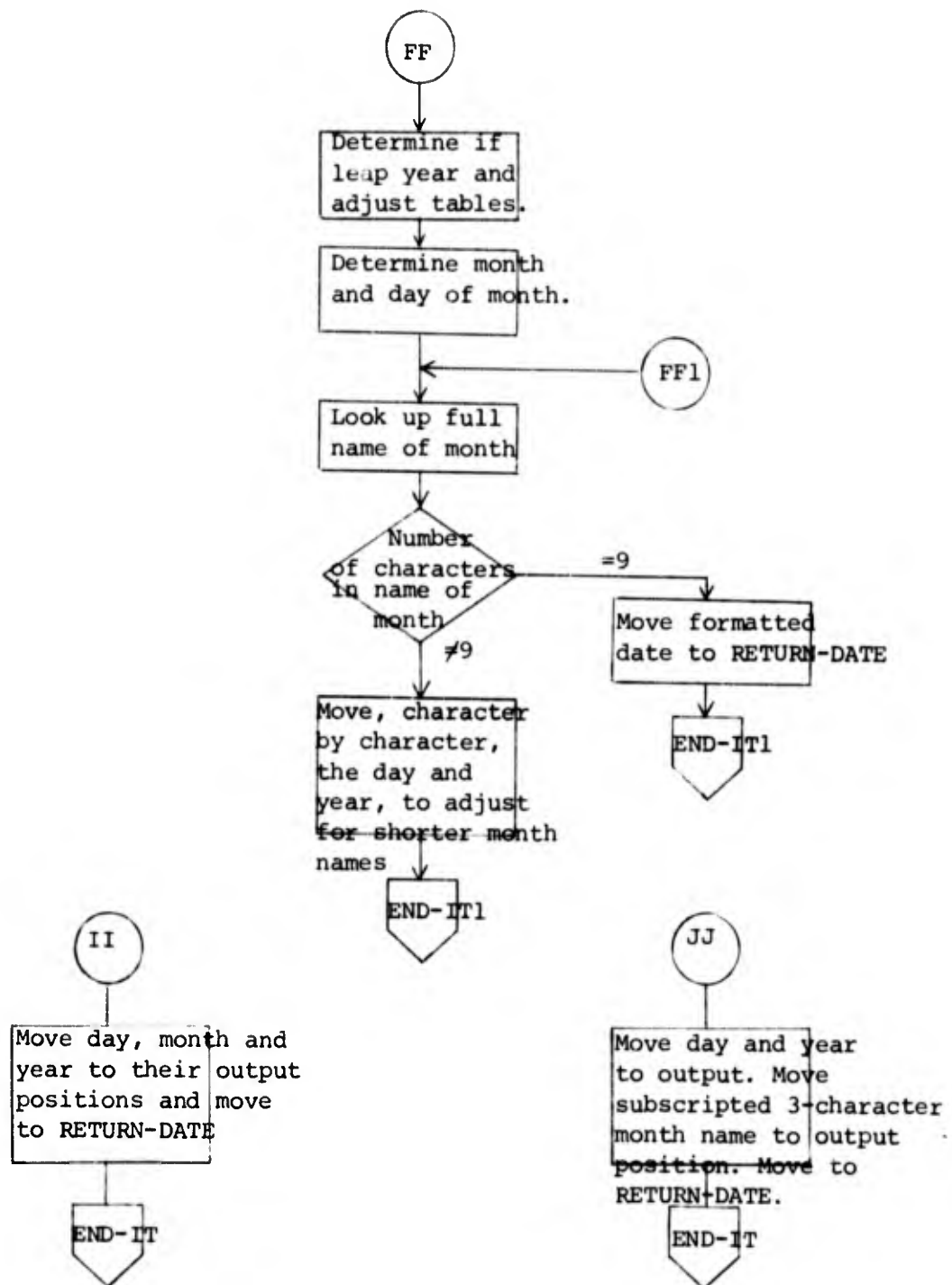


\*Month is used here to mean the numeric month (e.g., 12=December).

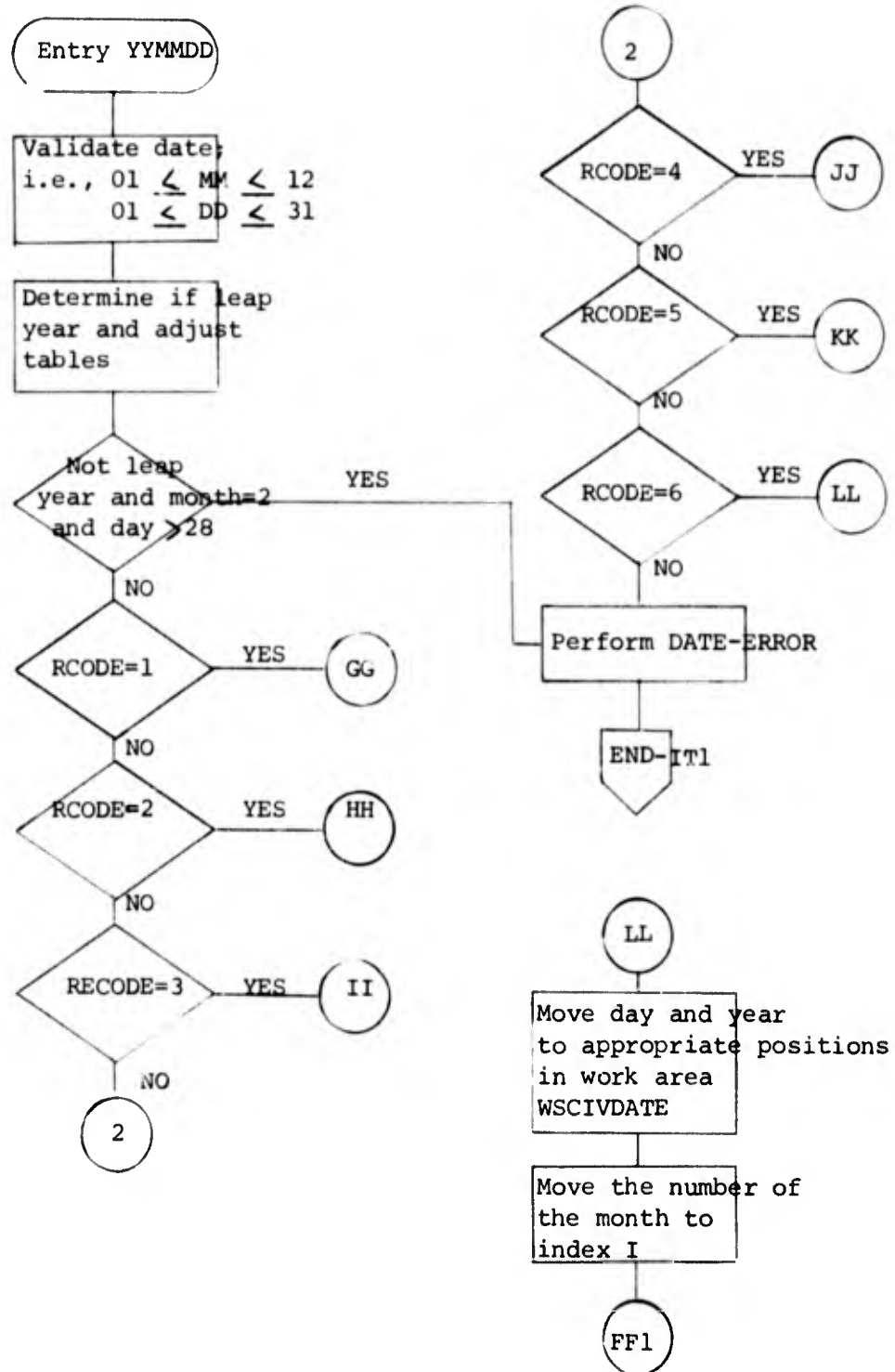
ENCLOSURE B



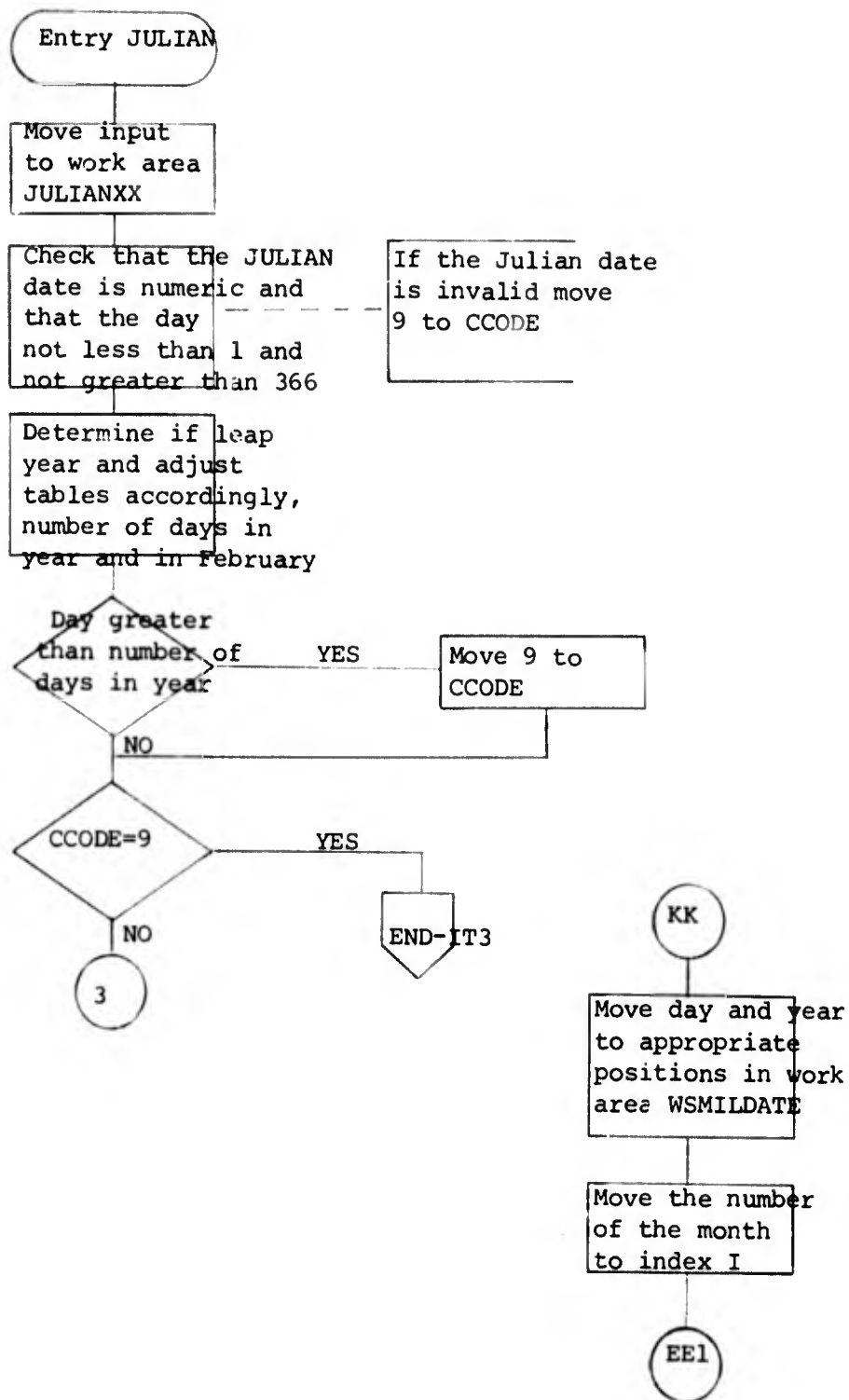
ENCLOSURE B



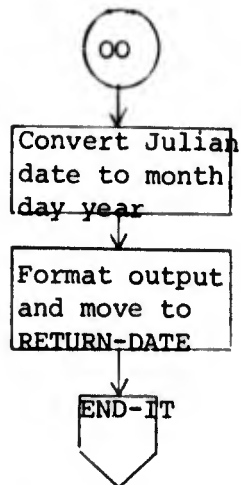
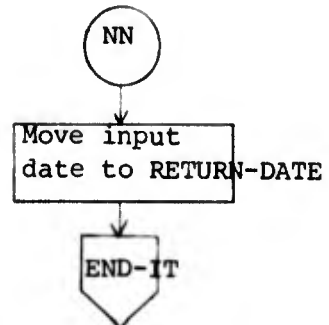
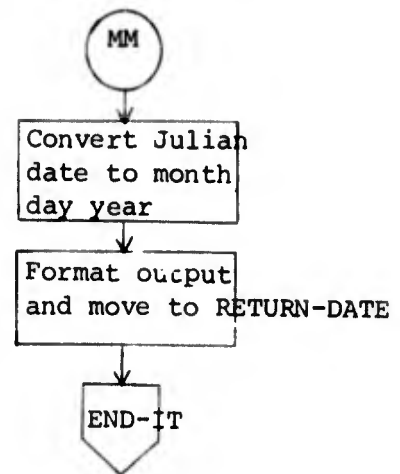
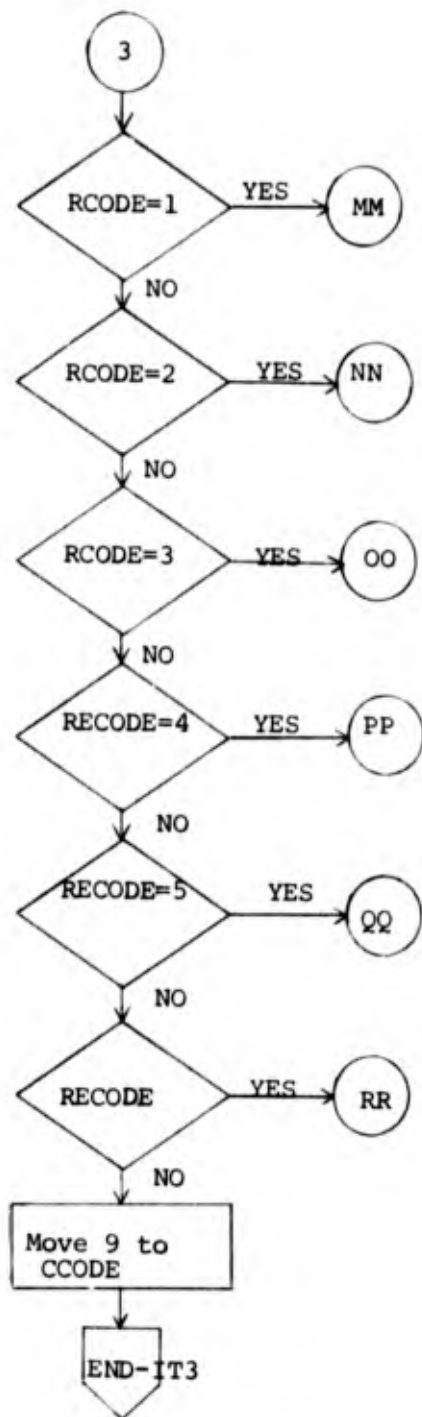
ENCLOSURE B



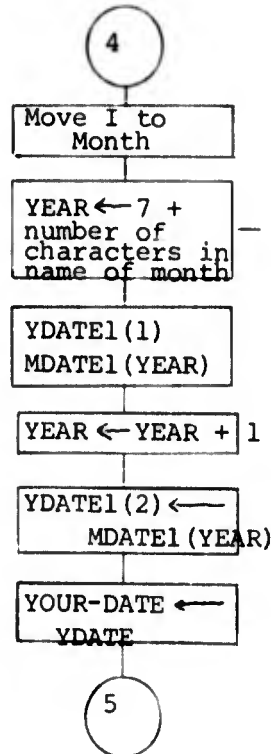
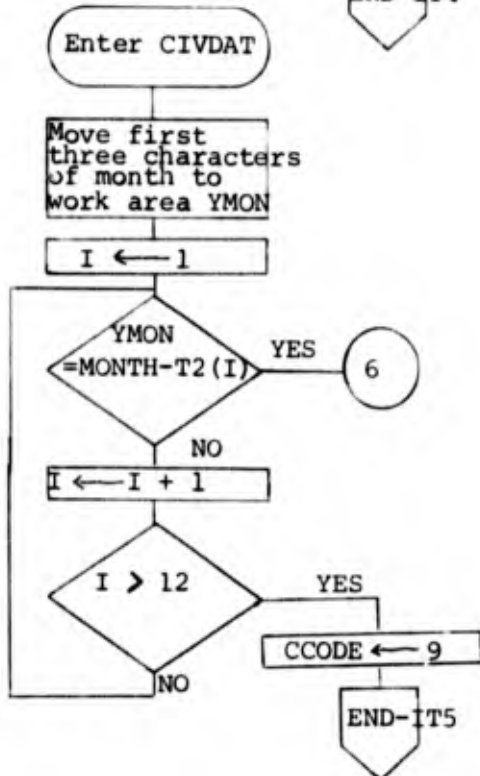
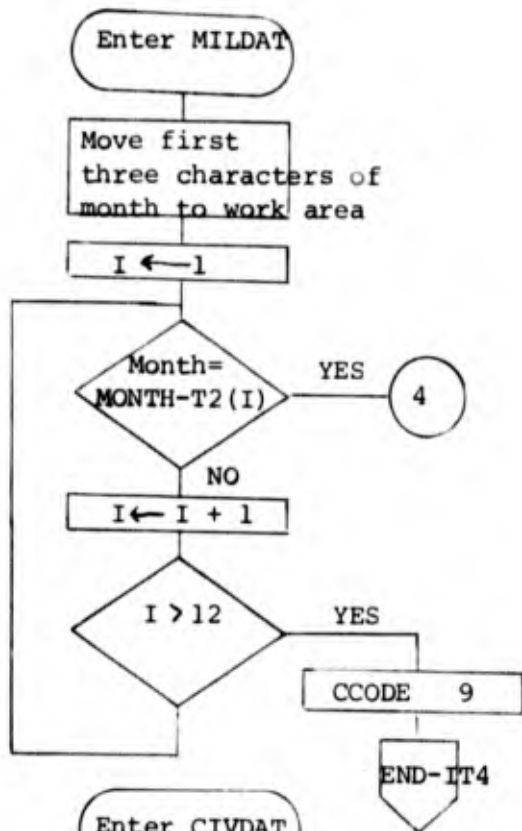
ENCLOSURE B



ENCLOSURE B

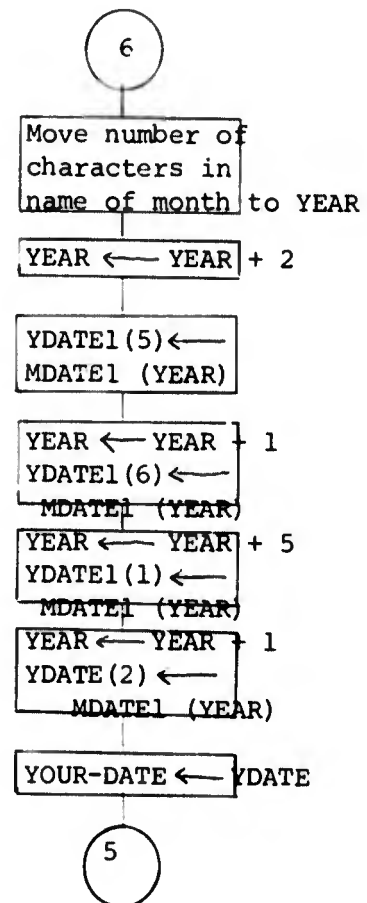


ENCLOSURE B



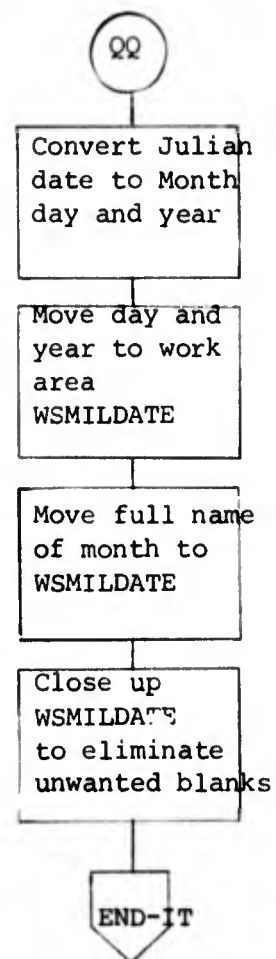
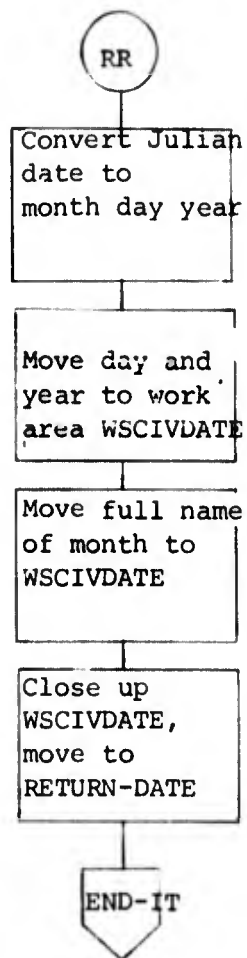
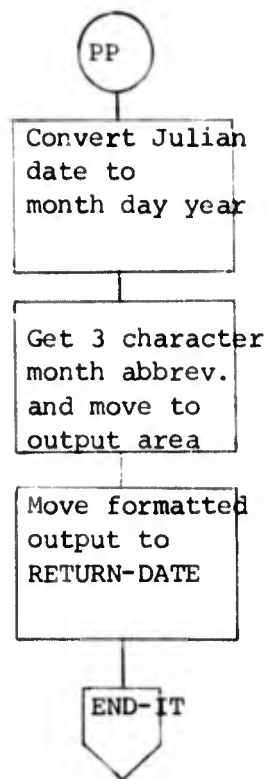
YEAR is a pointer to the third and fourth digit of the year in format 5; e.g., for 20MARCH1973 YEAR = 12 initially pointing at the "7"

MDATE1 contains the input date

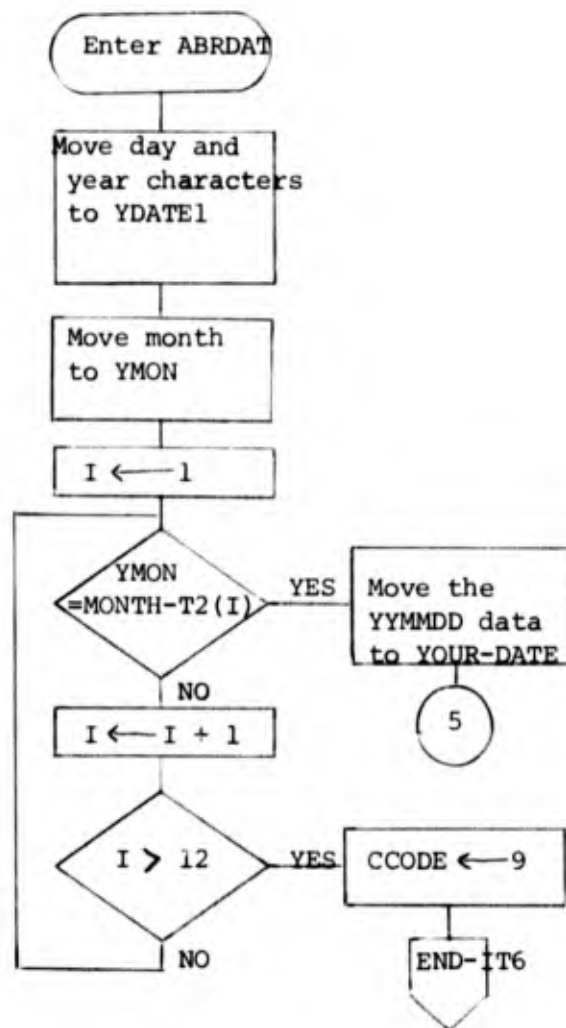


ENCLOSURE B



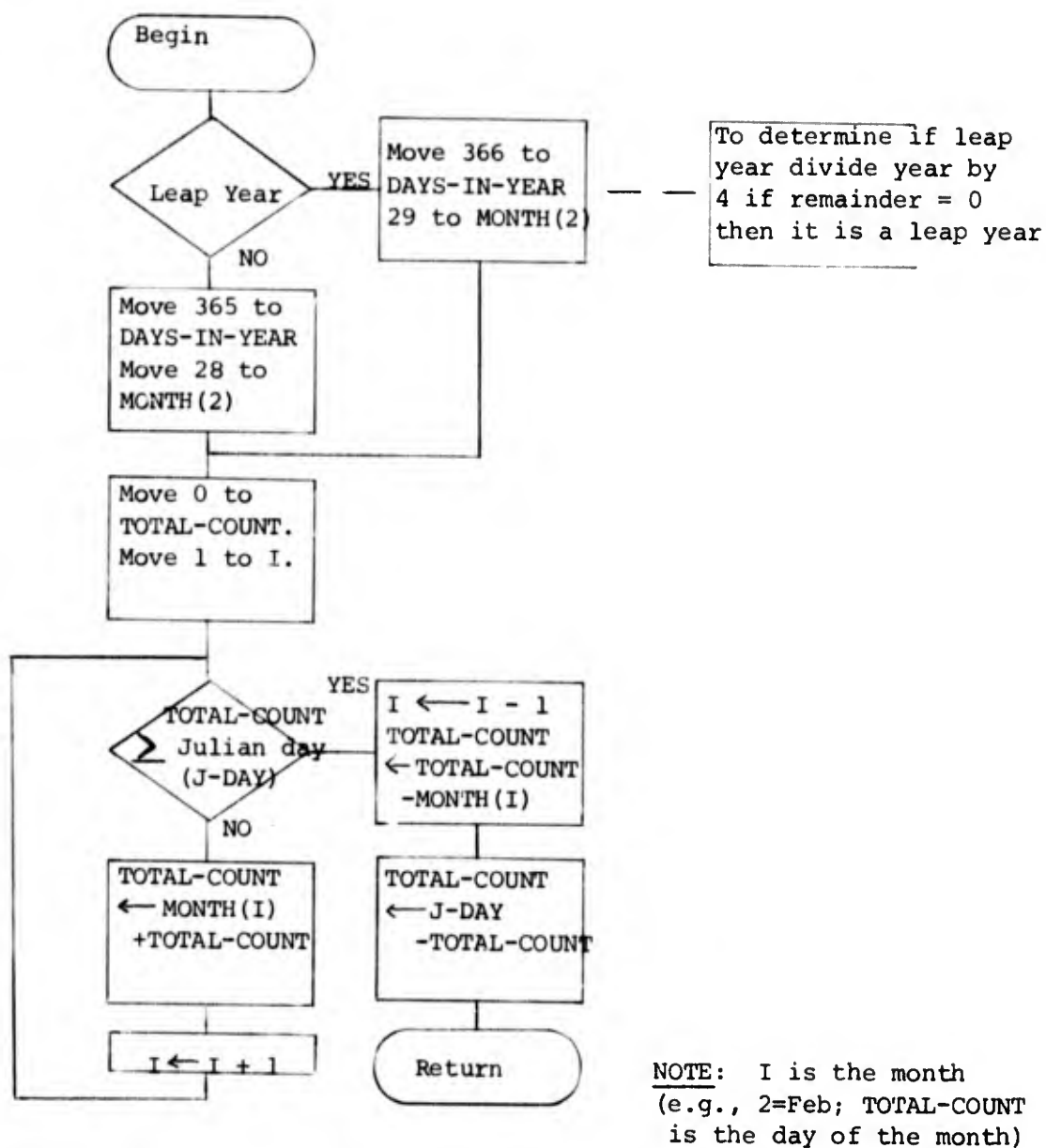


ENCLOSURE B



ENCLOSURE B

# TO CONVERT JULIAN DATE TO MONTH DAY YEAR



ENCLOSURE B

6. COORDINATE CONVERSION SUBPROGRAM (CRDFS).

- a. Subsystem Structure. Not applicable.
- b. Identification and Description of Subprogram included in the Subsystem.

(1) Abstract of Subprogram.

(a) Function. This program will convert the coordinates of one geographic point, from 11 character internal to 15 character external format. This program also makes error checks on the input data.

(b) Calling Sequence Parameters received and passed.

1. IN-LENGTH - Describes the number of characters in the IN-DATA field. It should be equal to 11.

2. OUT-LENGTH - Describes the number of characters generated by the subprogram and placed in the OUT-DATA field.

3. EXIT-FLAG - A value of "1" in EXIT-FLAG indicates a successful execution of this subprogram; a value of "2" indicates an error in the input data.

4. IN-DATA - The coordinates of one geographic point are input to this subprogram in IN-DATA. The coordinates are given in 11 character format, the first 5 characters give the latitude in degrees and thousandths of a degree, with the sign given in the zone position of the 5th byte, the 6th through 11th character give the longitude in degrees and thousandths of a degree with the sign in the zone position of the 11th byte, e.g. 45500105250.

5. OUT-DATA - This field will contain either the converted coordinates or an error message. The converted coordinates are given in 15 character external format: degrees, minutes, seconds, quadrant, degrees, minutes, seconds, quadrant with leading zeroes (North latitude and East longitude are considered positive), e.g. 453000N1051500W.

6. FIELD-NAME and FILE-NAME - These fields are not used.

(c) Narrative Flow. The entry point of the program in 'CRDFS.' The data is tested to determine if it is numeric and valid, it is then converted to the output format.

(2) Capabilities. This program validates and converts the coordinates of a geographic point from 11 character internal format to 15 character external format.

c. Error Messages.

(1) "ERROR COORDINATES NOT NUMERIC."

(a) There is non-numeric data in the input field of IN-DATA.

(b) Check for a blank input field or mispunched data.

(2) "ERROR COORDINATES NOT VALID."

(a) Either the magnitude of the latitude is greater than 90° or the magnitude of the longitude is greater than 180°.

(b) Check the input field for mispunched data.

7. COORDINATE CONVERSION SUBPROGRAM (CRD6S). This subroutine is a modification of CRDFS. It converts the geographic coordinates of a single point from the 13 character internal format to the 15 character external format, e.g.

+  
input 450001052505

output 453000N1051502W.

8. COORDINATE CONVERSION SUBPROGRAM (CRDGS).

a. Subsystem Structure. Not applicable.

b. Identification and Description of Subprograms included in the Subsystem.

(1) Abstract of Subprogram.

(a) Function. This program will convert the coordinates of a single geographic point from 15 character external to 11 character internal format. This program also makes error checks.

(b) Calling Sequence Parameters received and passed.

1. IN-LENGTH - Gives the number of characters in the IN-DATA field. It should be 15.

2. OUT-LENGTH - Gives the number of characters in the OUT-DATA field.

ENCLOSURE B

3. EXIT FLAG - A value of '1' in EXIT-FLAG indicates a successful completion of the subroutine; a value of '2' indicates an error in the input data.

4. IN-DATA - The coordinates of a single geographic point are input into IN-DATA. The coordinates should be given in 15 character external format, i.e. 7 characters for the latitude expressed in degrees, minutes, seconds, and N or S with leading zeros, and 8 characters for the longitude expressed in degrees, minutes, seconds, and E or W with leading zeros, e.g.

45°30'00"N105°15'00"W would be given as

453000N1051500W.

5. OUT-DATA - This field will contain either the converted coordinates or an error message. The converted coordinates are given in 11 character internal format, i.e. 5 characters for the latitude expressed in degrees and thousandths of a degree, with the sign in the zone position of the 5th byte; the 6th through 11th characters give the longitude in degrees and thousandths of a degree with the sign in the zone position of the 11th byte, e.g.

+  
45500105250.

6. FIELD-NAME and FILE-NAME - These fields are not used.

7. Narrative Flow - The entry point of the program is 'CRDGS.' The data is tested to determine if it is numeric and valid, and if it is, then it is converted.

(2) Capabilities. This subroutine can convert the coordinates of one point from 15 character external to 11 character internal format.

c. Error Messages.

(1) "ERROR COORDINATES ARE NOT NUMERIC."

(a) There are non-numeric characters in the degrees, minutes or seconds fields.

(b) Check input field for blanks and non-numeric data.

(2) "ERROR QUADRANTS INVALID."

ENCLOSURE B

(a) The seventh character is not 'N' or 'S' or the fifteenth character is not 'E' or 'W.'

(b) Check the seventh and fifteenth character for punch errors.

(3) "ERROR COORDINATES INVALID."

(a) The magnitude of the latitude is greater than 90°, or the magnitude of the longitude is greater than 180°, or the magnitude of the minutes or seconds is greater than 59.

(b) Checks the above fields for punch errors.

9. COORDINATE CONVERSION SUBPROGRAM (CRD7S). This subroutine is a modification of CRDGS. It converts the geographic coordinates of a single point from 15 character external to 13 characters internal format, e.g.

input 453000N1051502W

+

output 4550001052505.

10. COUNTRY CODE CONVERSION SUBPROGRAM (CTY1S).

a. Subsystem Structure. Not applicable.

b. Identification and Description of Subprograms included in the Subsystem.

(1) Abstract of Subprogram.

(a) 1. Function. The program is designed to accept a 2-character country code, search a table of country names and output a 14-character country name, or an error message if the country name is not found.

(b) Calling Sequence Parameters received and passed.

1. IN-LENGTH - Describes the number of characters in the IN-DATA field. It should be equal to 2.

2. OUT-LENGTH - Describes the number of characters returned by the program in the OUT-DATA field. If the country name is located OUT-LENGTH will be 14, if not OUT-LENGTH will be 31.

ENCLOSURE B

3. EXIT-FLAG - A value of 1 in EXIT-FLAG indicates that the country name has been found. A value of 2 indicates that the country name has not been found.

4. IN-DATA - Contains the two character country code.

5. OUT-DATA - Either contains the 14-character country name or the 31-character error message when the country name is not located.

6. FIELD-NAME and FILE-NAME - These fields are not used for this program.

(c) Narrative Flow. The entry point of this program is CTY1S. A binary search of the country code table is performed (BINARY-TEST, CC-GREATER, CC-LESS). The country name is returned (FOUND-IT). If the country name is not found an error message is returned (CC-ERROR-EXIT).

(2) Capabilities. This subprogram given a 2-character country code performs a binary search of the country code table and returns a country name on error message.

c. Error Message. "COUNTRY CODE NOT FOUND IN TABLE."

(1) The country code is not standard or it has been misspunched.

(2) Check the country code.

#### 11. COMPARISON OF MARK III AND MIDMS GEOGRAPHIC OPERATORS AND CONVERT ROUTINES.

a. Transfer Values.

(1) In Mark III a field must be defined in the FFT to receive transfer values from the special operators.

(2) In MIDMS a field to receive transfer values may be defined in the query or in the FFT.

b. General Geographic Operator.

(1) The capability to compare a file polygon with a search circle for overlap exists in Mark III and not in MIDMS.

ENCLOSURE B



(2) Capabilities in MIDMS and not in Mark III.

(a) The distances left and right of the leg need not be equal.

(b) There may be more than one frame across a leg.

(3) Language Differences.

(a) Default value of leg width.

1. In Mark III it is 20 nm.

2. In MIDMS it is 0 nm.

(b) Transfer Values. The MIDMS Route Search Operator returns a 3 character, user-defined, leg number, as well as a one character sort key (left=2, right=1).

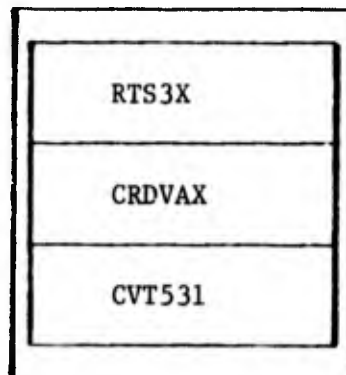
d. Circle Search.

(1) In Mark III the radius of the search circle is limited to 99.9 nm, whereas in MIDMS it is limited to 300 nm for accurate results.

(2) The MIDMS Circle Search Operator can return an identifying number for the search circle or point which hits the file circle on point. This capability is not available in Mark III.

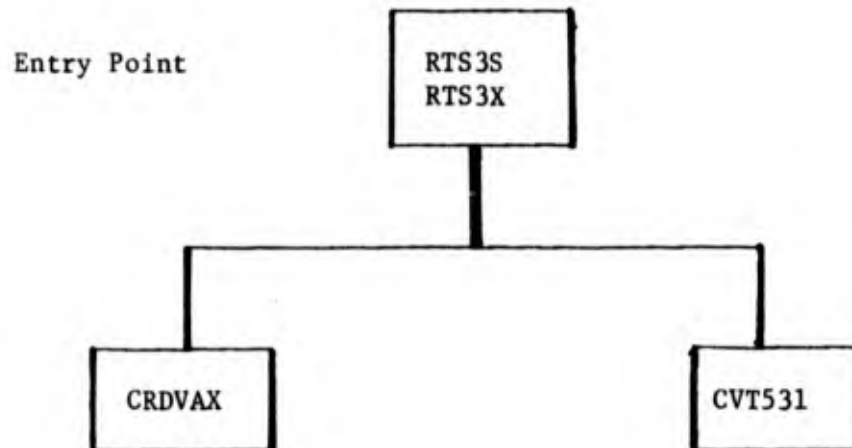
## 12. ROUTE SEARCH SPECIAL OPERATOR (RTS3X).

a. Subsystem Structure. The Route Search Special Operator is depicted below.



ENCLOSURE B

The flow of control is shown below:



The Route Search Special Operator passes a coordinate from a file against the data description of a leg to determine whether the coordinate falls within specified parameters. If it does, the coordinate's relative position to the leg, and its frame (optional) or its time (optional) is output with other supplemental data.

b. Identification and Description of Subprogram.

(1) RTS3X

(a) Abstract Function.

1. Function. Given the coordinates of a geographic point from the file and the parameter defining a leg (from RTC3S), this routine determines if the point lies within the leg rectangle. If the point lies within the leg then the cross-leg and down-leg distance then the cross-leg and down-leg distance are returned along with the frame number (optional) or time (optional).

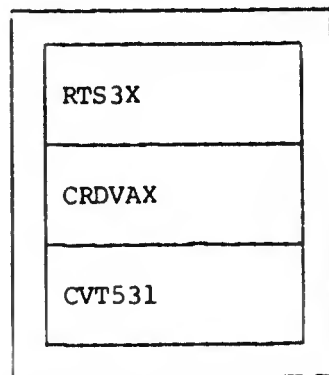
2. Calling Sequence.

Entry point RTS3S using  
USER-CALLING-SEQUENCE GIVING  
USER-CALLING-SEQUENCE.

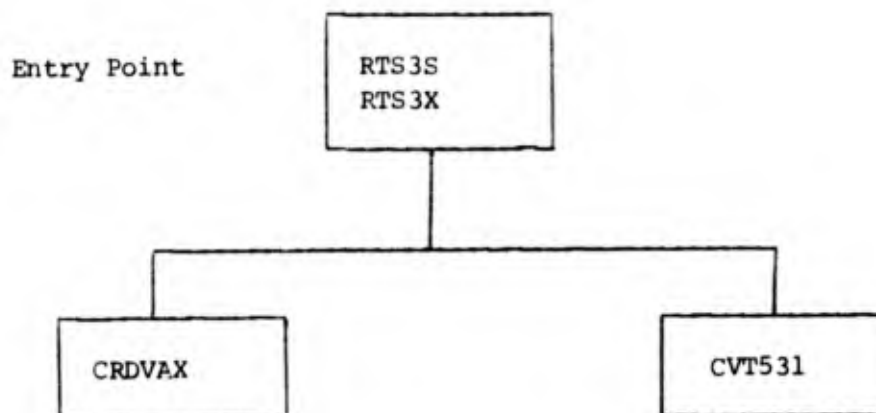
ENCLOSURE B

## Route Search Special Operator (RTS3X)

a. Subsystem Structure. The Route Search Special Operator is depicted below.



The flow of control is shown below:



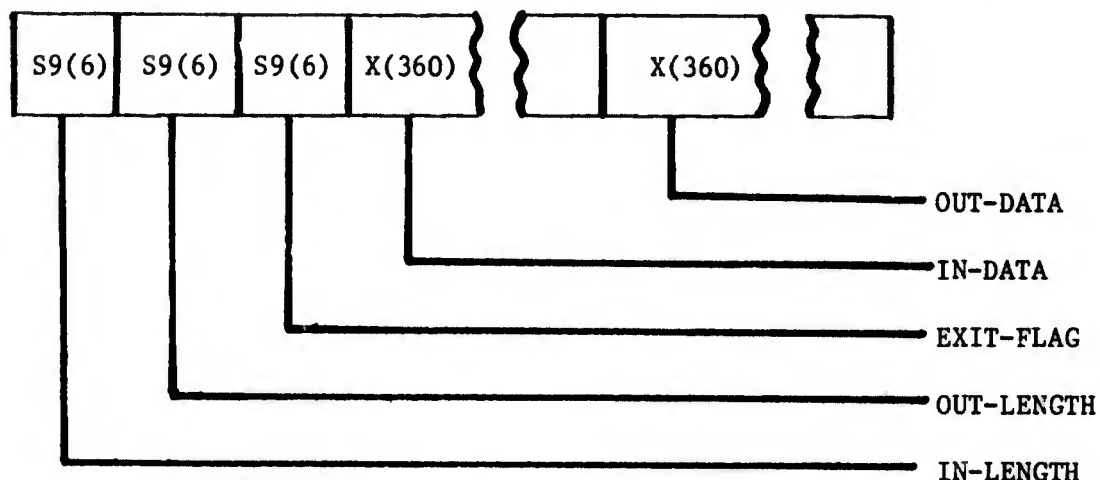
The Route Search Special Operator passes a coordinate from a file against the data description of a leg to determine whether the coordinate falls within specified parameters. If it does, the coordinate's relative position to the leg, and its frame (optional) or its time (optional) is output with other supplemental data.

b. Identification and Description of Subprogram.

(1) RTS3X

(a) Abstract Function.

The formats of the fields in USER-CALLING-SEQUENCE are depicted below:



IN-LENGTH is a binary field, which contains the number of valid characters in the IN-DATA field.

OUT-LENGTH is a binary field, which contains the number of valid characters in the OUT-DATA field.

EXIT-FLAG is a binary field. The program places a code into EXIT-FLAG to indicate the outcome of the execution.

EXIT-FLAG = 1 file point in leg ("hit").  
 2 file points not in leg ("miss").  
 4 file point coordinates are invalid.

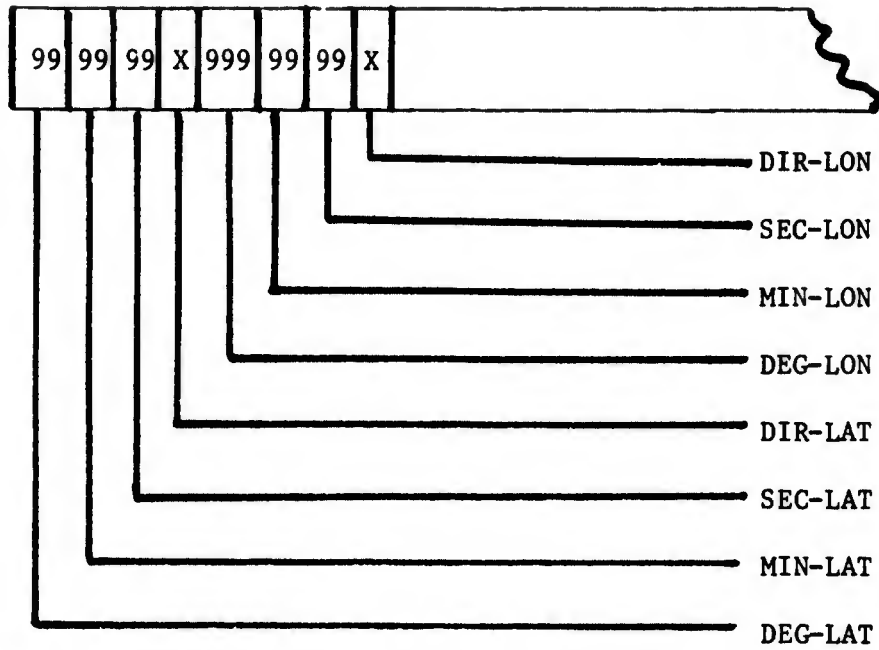
IN-DATA contains the file point coordinates in either 11, 13, or 15 character format as shown below.

ENCLOSURE B

15-char

1

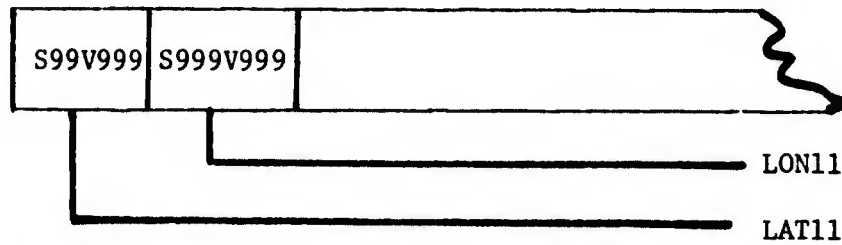
1  
5



11-char

1

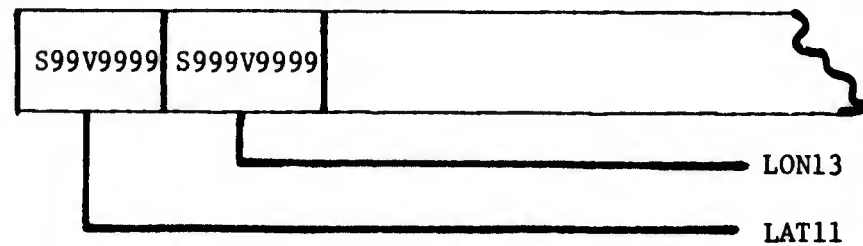
1  
1



13-char

1

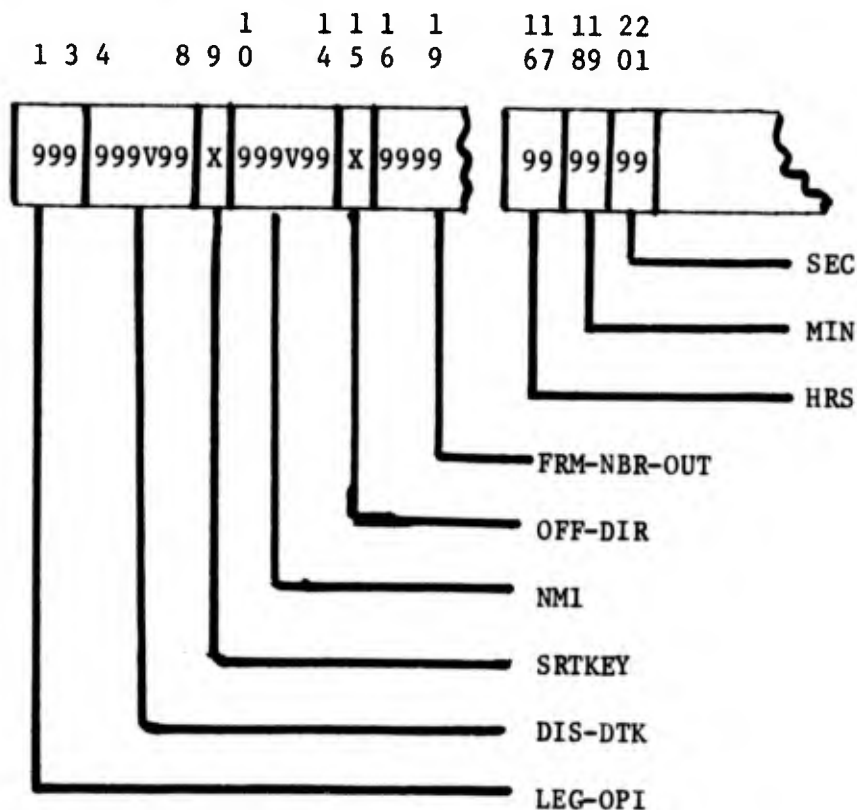
1  
3



ENCLOSURE B

The OUT-DATA field is used to input the leg parameter generated by RTC3S upon entering RTS3S, and to output the generated value or error message upon exiting RTS3S. The format of OUT-DATA upon entering RTS3S is identical to the format upon exiting RTC3S (see description of RTC3S). The format of OUT-DATA upon exiting RTS3S is shown below.

In even of a "hit" (EXIT-FLAG=1),



LEG-OPI is the leg identifier number. (Display)

DIS-DTK is the down-leg distance in nm. (Display)

SRTKEY is the sortkey. (Display)

= 1 for right  
2 for left

NMI is the cross-leg distance in mn. (Display)

ENCLOSURE B

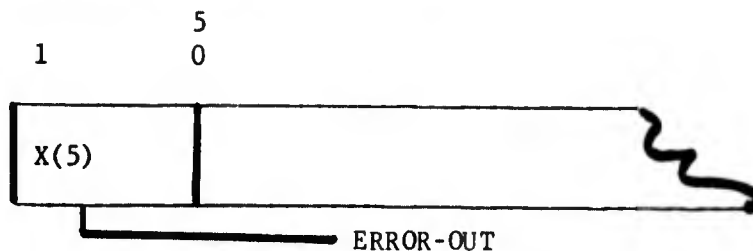
OFF-DIR is the direction of the point  
"L" or "R" for left or right. (Display)

If the frame option is used:

FRM-NBR-OUT is the frame number of the file point. (Display)

If the time option is used, HRS, MIN, and SEC gives the time that the file point would occur along with the leg. (Display)

In the event of invalid coordinates in the file EXIT-FLAG is set to 4 and an error message is placed in OUT-DATA.



(b) Description. The Route Search Special Operator uses a planar model. The coordinates of the file point are projected onto a plane with a cartesian coordinate system with origin at the initial point of the leg. The formulas used are:

$$XTEMP = (X2-X0) * 60.0000 * DEL,$$

and

$$XTEMP=(Y2-40) *60.0000.$$

when:

X2 is the longitude of the file point in degrees.

Y2 is the latitude of the file point in detrees

X0, Y0 are the longitude and latitude of the origin of the leg in degrees.

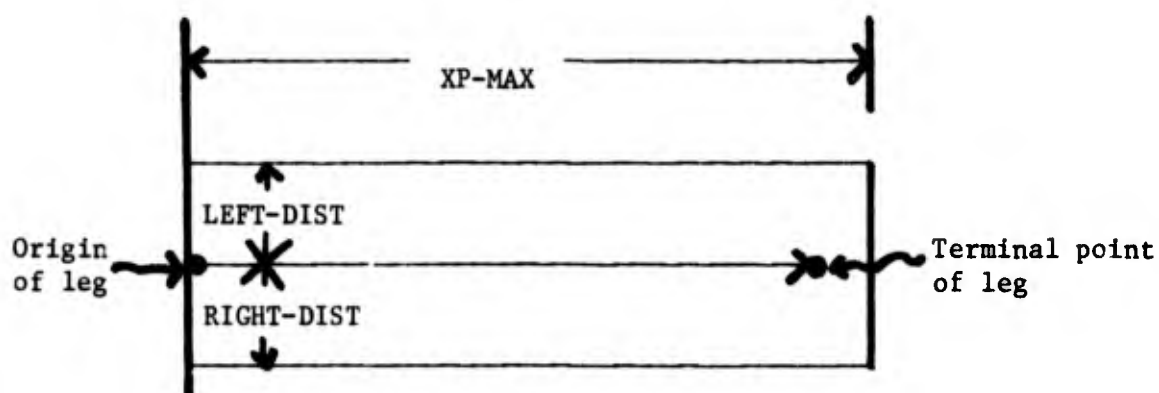
DEL is the cosine of the average latitude.

The coordinates are then rotated to a coordinate system along the leg.  
The formulas used are:

$$XP = XTEMP * COSINE-THETA + YMTEMP * SINE-THETA.$$

The planar model is ecpecter

The picture of leg coordinate frame is amplified below:



Once the coordinates along the leg (XP) and perpendicular to the leg (YP), have been computed they are compared with the limits computed in RTC3S. The limits are:

ENCLOSURE B



$$0 \leq XP \leq XP-MAX$$

$$- RIGHT-DIST \leq YP \leq LEFT-DIST$$

(NOTE: XP and YP are signed numbers, YP positive to the left of the leg negative right of the leg.)

The value of OPT is tested to determine if the leg used the frame-option (OPT=1) or line option (OPT=2).

If the frame option has been taken, a test is made to see if the file point lies in the extension area (i.e.,  $XP \geq LEG-LENGTH$ ) in which case the terminal frame number is returned. If the file point is not in the extension area, the following formula is used to compute the frame number.

$$frame-number = DF + FI * \left[ \frac{XP}{\Delta X} \right] + \left[ \frac{LEFT-DIST-YP}{\Delta Y} \right]$$

where:

BF is the beginning frame number.

X is the length of a frame in the X direction.

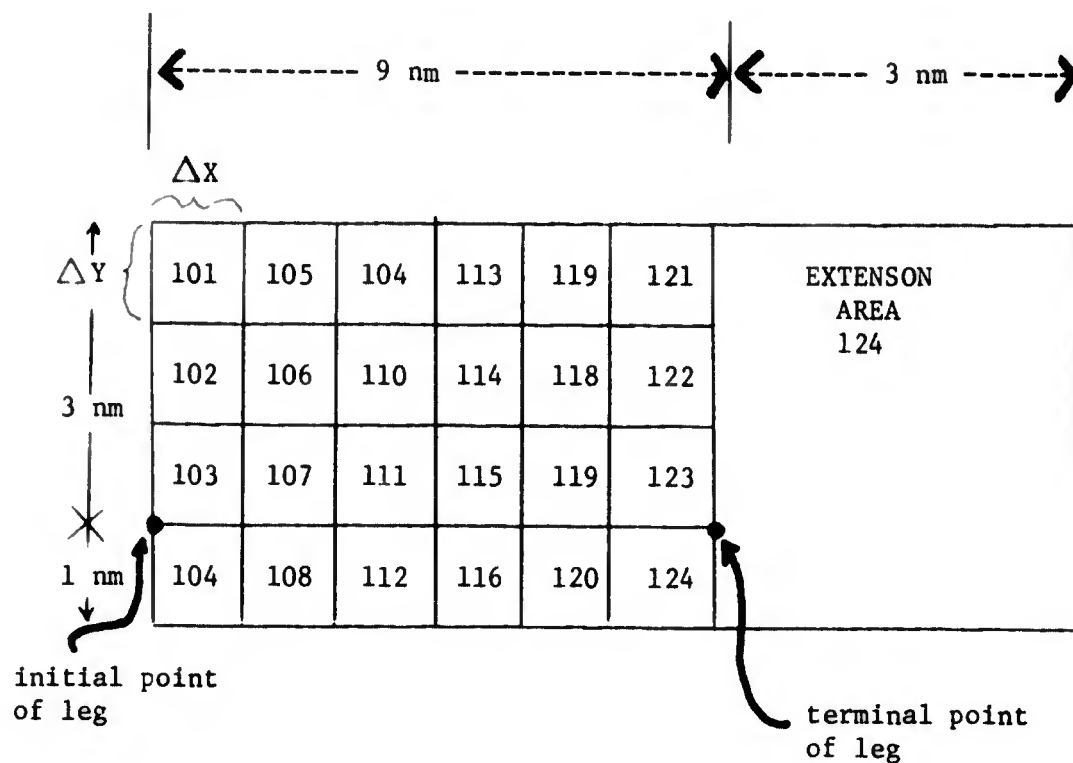
Y is the length of a frame in the Y direction.

FI is the number of frames in the Y direction in the leg rectangle.

The symbol  $[Z]$  represents the largest integer in Z (i.e., if  $Z = 1.37$  then  $[Z] = 1$ ).

These parameters are depicted on the next page.

ENCLOSURE B



Here BF = 101, FI = 4,  $X = \frac{9}{6} = 1.5$  nm, Y = 1 nm

If the time option has been taken, the time from midnight of passing the target in seconds is computed by:

$$\text{TME-AT-TGT} = \text{TIME1} * \text{SEC-PER-MILE};$$

the time in hours by

$$\text{hours} = \left[ \frac{\text{TM-AT-TGT}}{3600} \right];$$

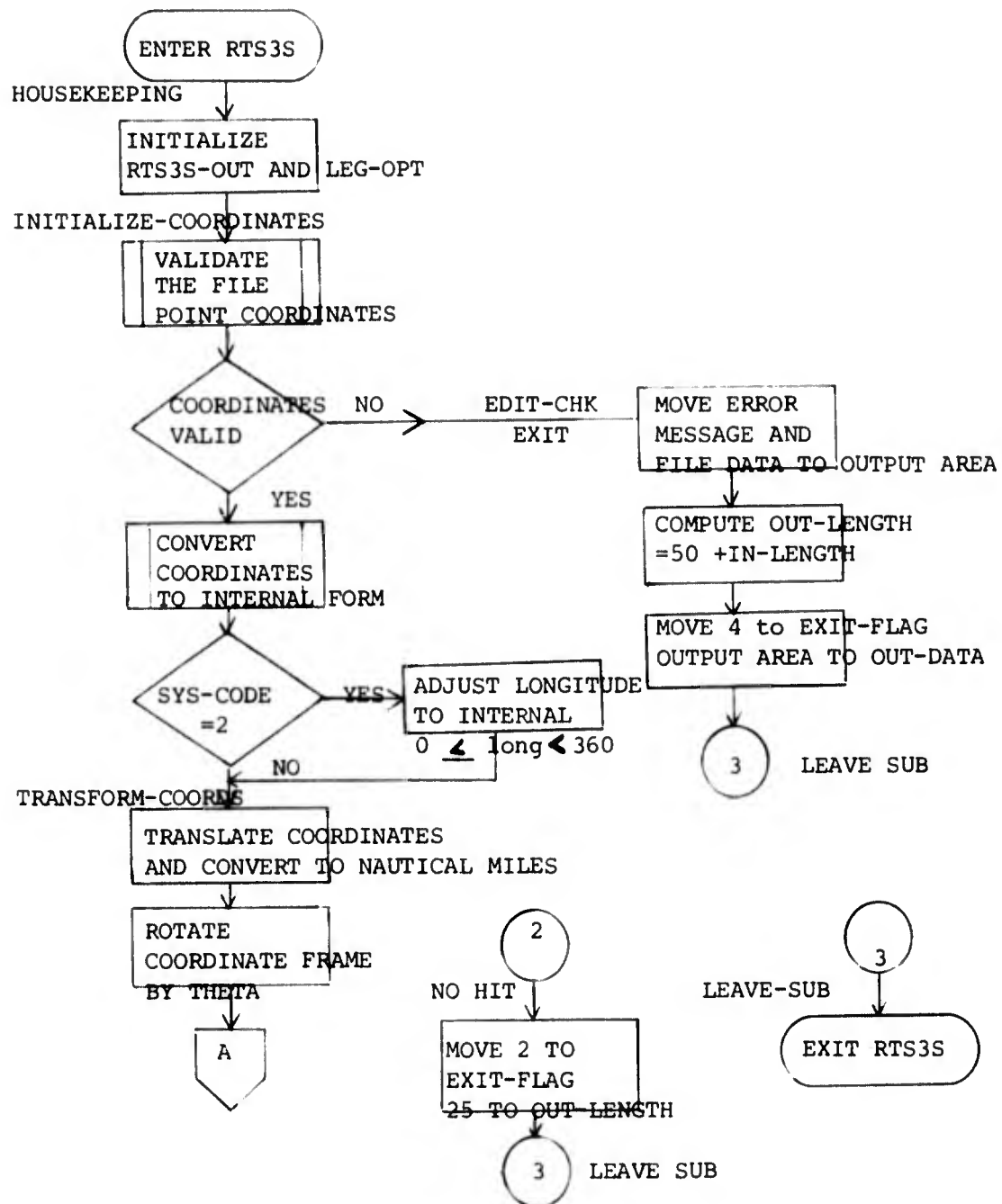
the minutes by

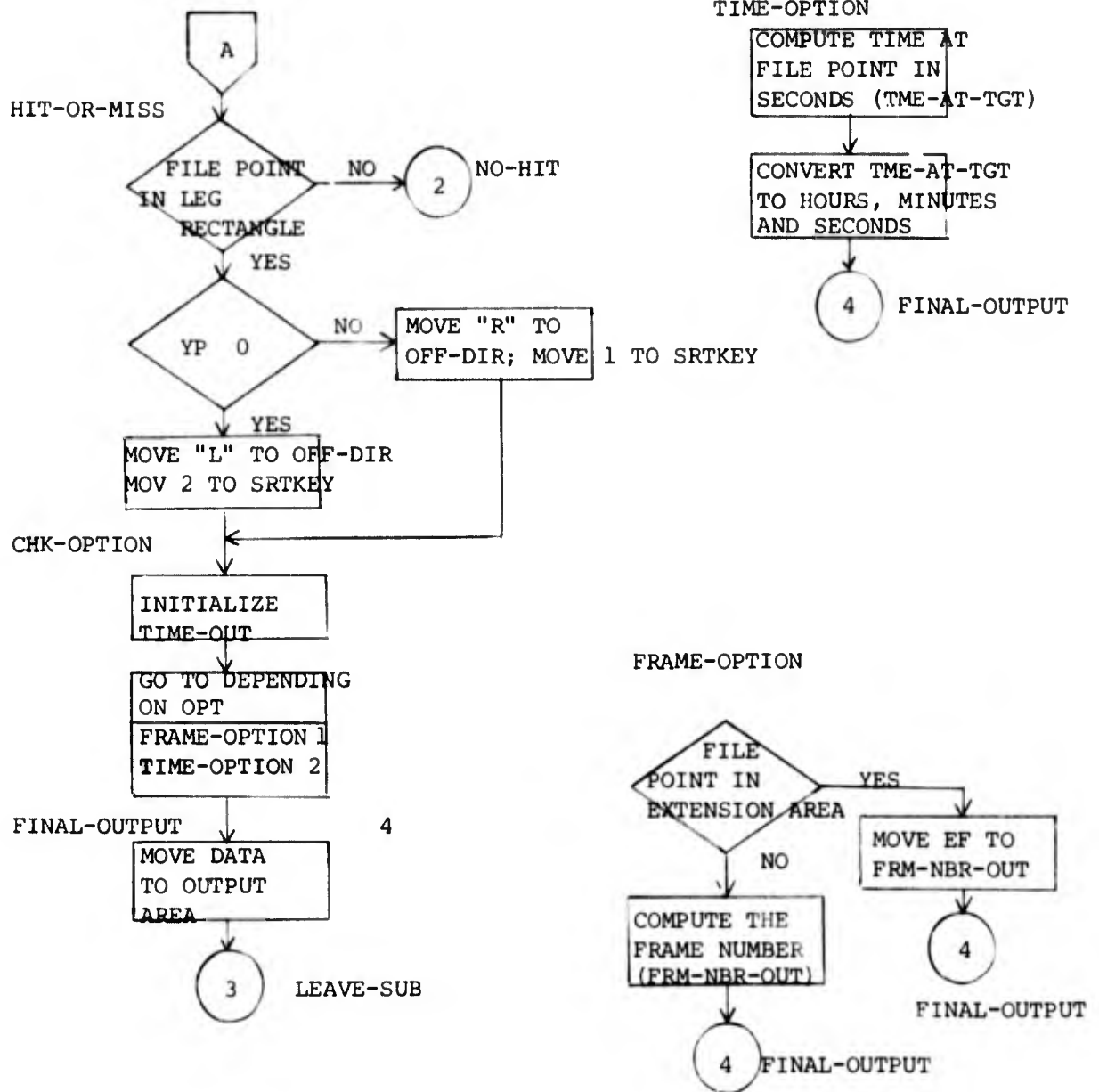
$$\text{minutes} = \left[ \frac{(\text{TME-AT-TGT} - 3600 * \text{Hours})}{60} \right];$$

and the seconds by

$$\text{seconds} = \text{TM-AT-TGT} - 60 * \text{minutes} - 3600 * \text{hours}.$$

In the event that the file point is invalid, EXIT-FLAG is set to 4 and an error message returned.



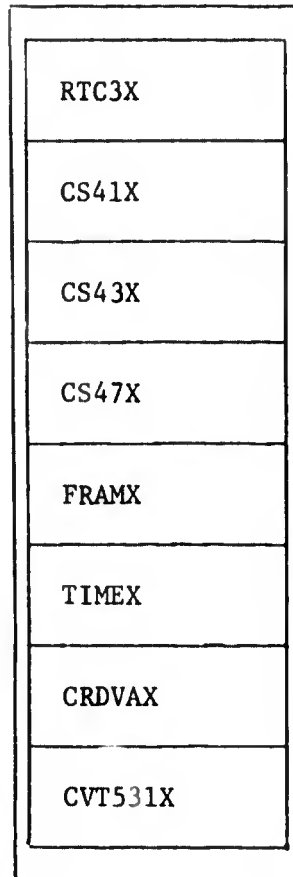


ENCLOSURE B

13. ROUTE SEARCH CONVERSION MODULE (RTC3X).

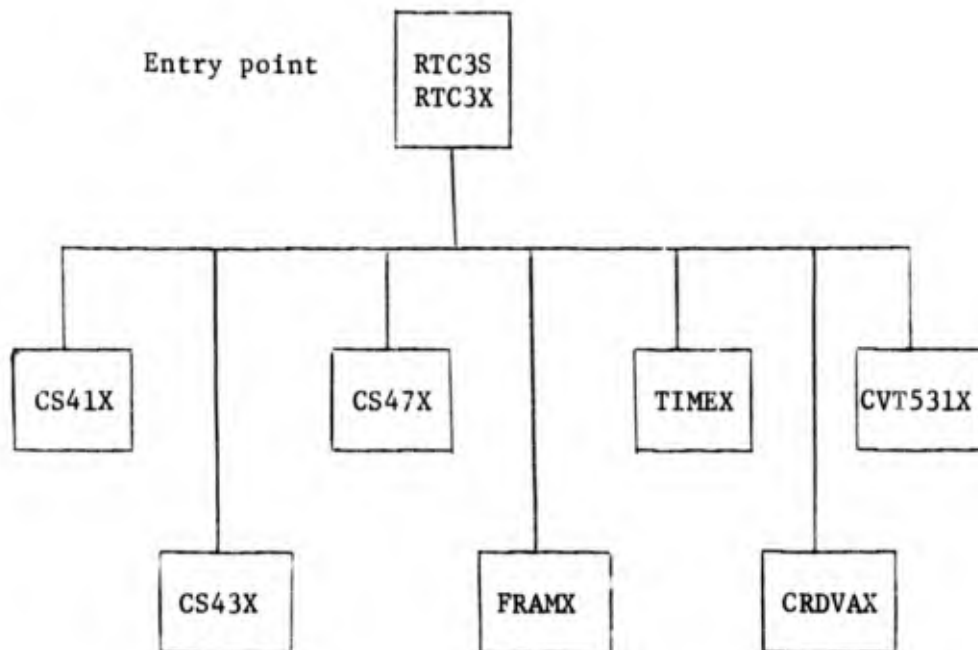
a. Subsystem Structure.

(1) The Route Search Conversion module is depicted below:



The flow of control is shown on the next page.

ENCLOSURE B



The Route Search Conversion module acts as a preprocessor for the Route Search special operator. A description of a leg of the route is input to this module and the parameters to be used by the Route Search special operator during the file search.

b. Identification and Description of Subprograms included in the Subsystem.

(1) RTC3X

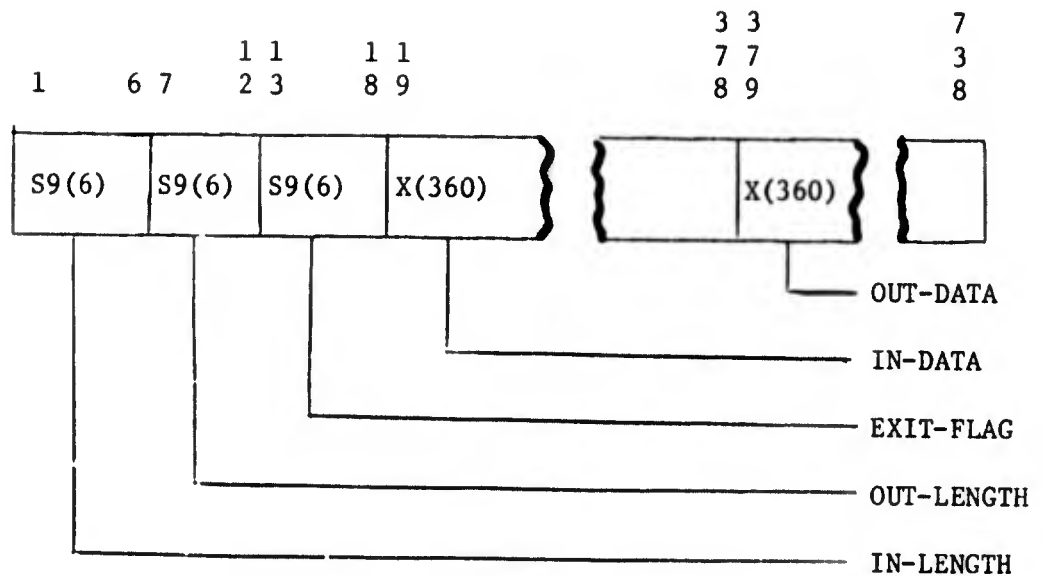
(a) Abstract.

1. Function. This is the executive routine of the Route Search Conversion module. The function of the module is to construct a rectangle about the route leg, convert multiple formats into a standard format, and make error checks on user supplied parameters. If no errors are encountered, data output from this module may be used as input to the Route Search special operator.

2. Calling Sequence.

Entry Point RTC3S using USER-CALLING-SEQUENCE GIVING USER-CALLING-SEQUENCE.

The formats of the fields in USER-CALLING-SEQUENCE are depicted on the next page.



IN-LENGTH is a binary (computational-1) field, which contains the number of valid characters in the IN-DATA field.

OUT-LENGTH is a binary (computational-1) field, which contains the number of characters generated by the program and placed in the OUT-DATA field.

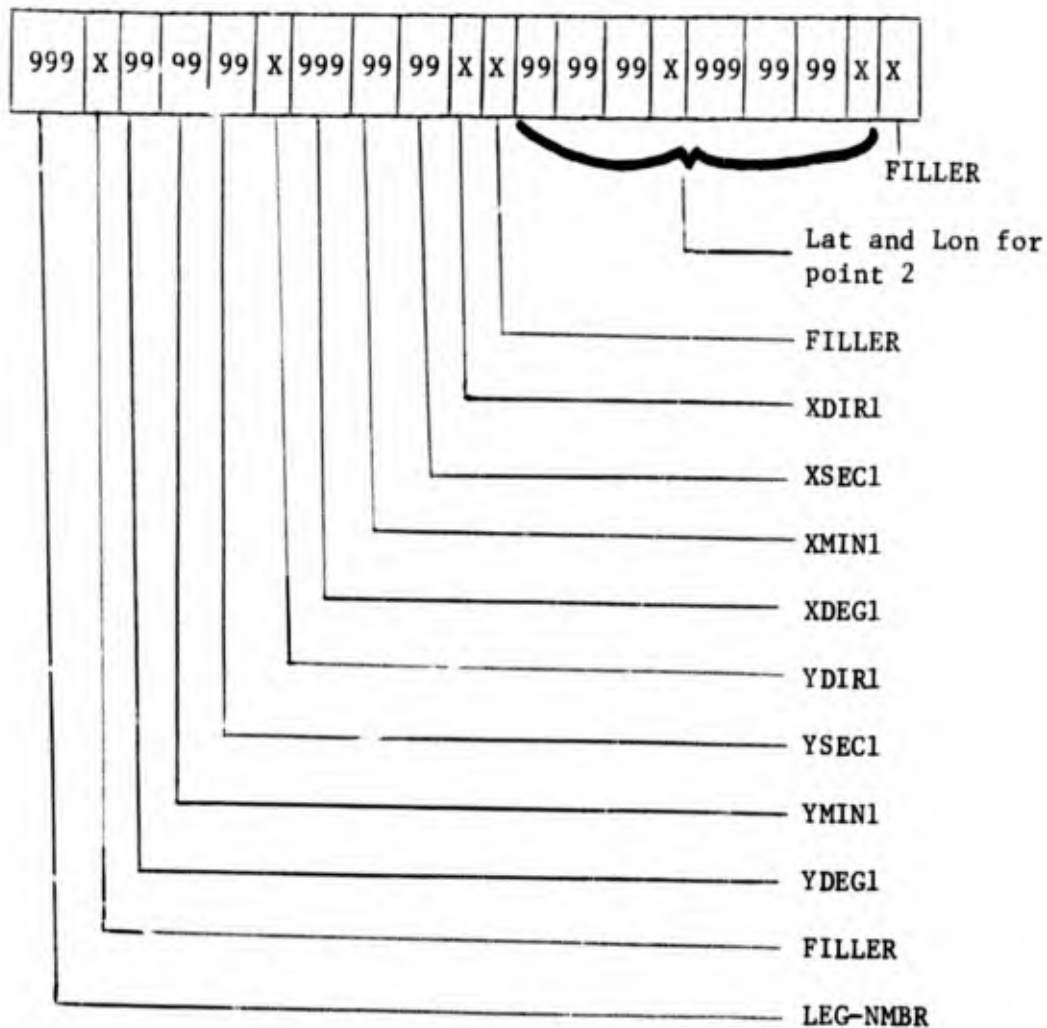
EXIT-FLAG is a binary (computational-1) field which contains a flag which may have a value of "1" or "2". "1" indicates good data and "2" indicates invalid input data and an error message will be found in the OUT-DATA field.

IN-DATA contains the input data left-justified. The first 36 characters are the same for all formats.

ENCLOSURE B

1 1 1 1 11 11 1 2 2  
1 3 4 56 78 90 1 2 4 56 78 9 0 1

3 3  
5 6



LEG-NMBR is the identifying number of the leg (usage is display).

YDEG1, YMIN1, YSEC1 are the degrees, minutes, and seconds of the latitude of the initial point of the leg (usage is display).

YDIR1 is either "N" or "S" indicating North or South (usage is display).

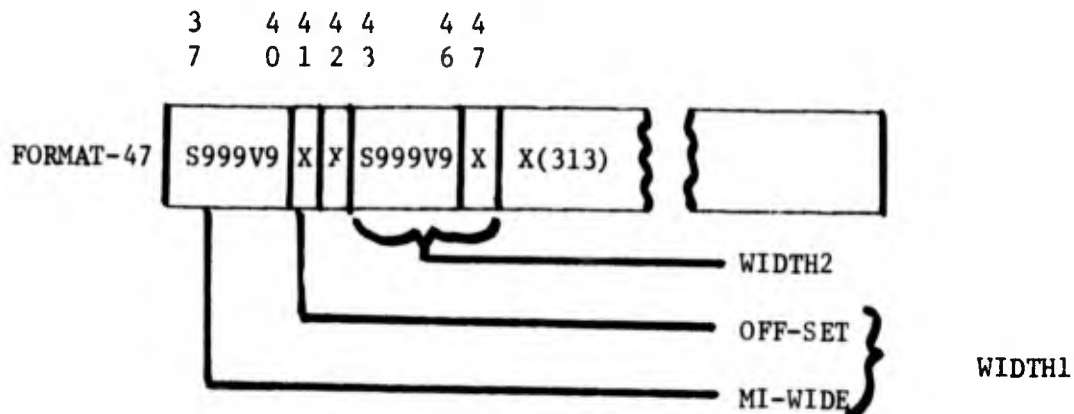
XDEG1, XMIN1, XSEC1 are the degrees, minutes, and seconds of the longitude of the initial point of the leg (usage is display).

XDIR1 is either "E" or "W" indicating East or West (usage is display).

ENCLOSURE B



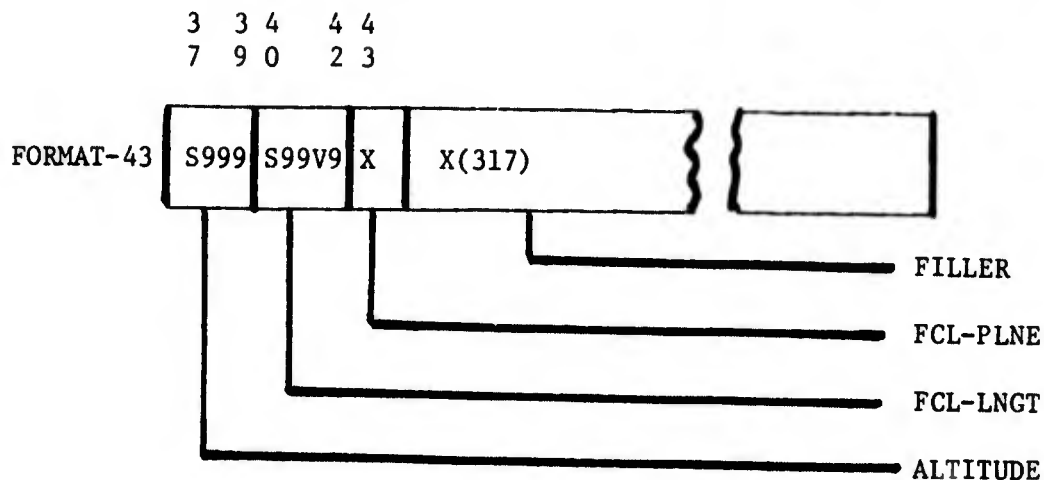
The next portion of the input field contains the left and right distance either or directly on in the altitude, focal length, and focal plane format.



MI-WIDE is the cross-range distance in nautical miles (usage is display).

OFF-SET is "L" for left or "R" for right.

WIDTH2 gives the corresponding data for the other side of the leg.



ALTITUDE is the altitude in thousands of feet (usage is display).

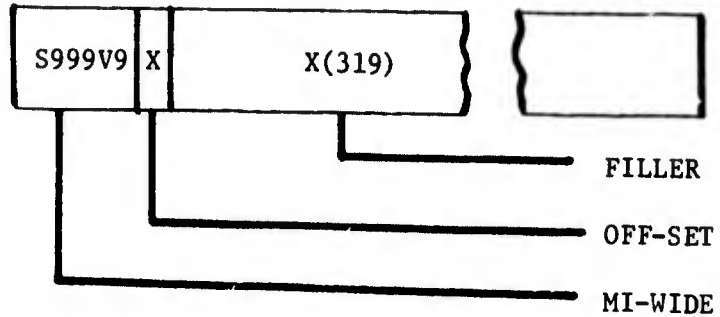
FCL-LNGT is the focal-length in inches.

FCL-PLNE is the focal-point code:

- A 4.5 inches,
- B 9.0 inches.

3 4 4  
7 0 1

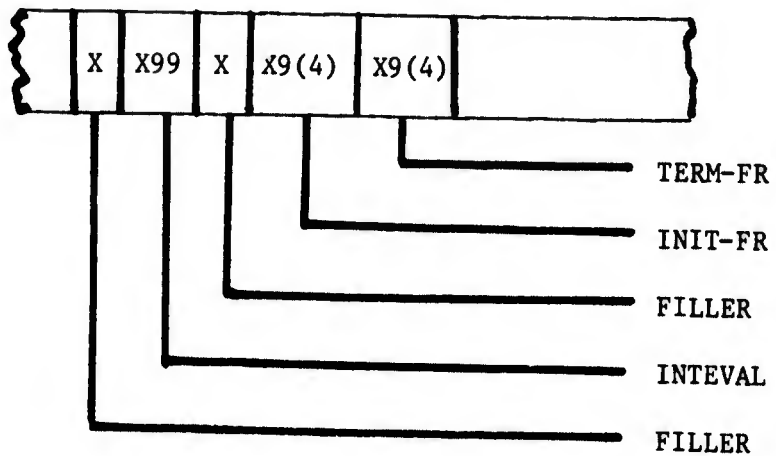
FORMAT-41



Only one off set is given in this format (the other is assumed to be zero).

If the frame option is used we may define the leg rectangle by either format 47, 43, or 41 and continue with the frame data.

FORMAT-59  
-55  
-53

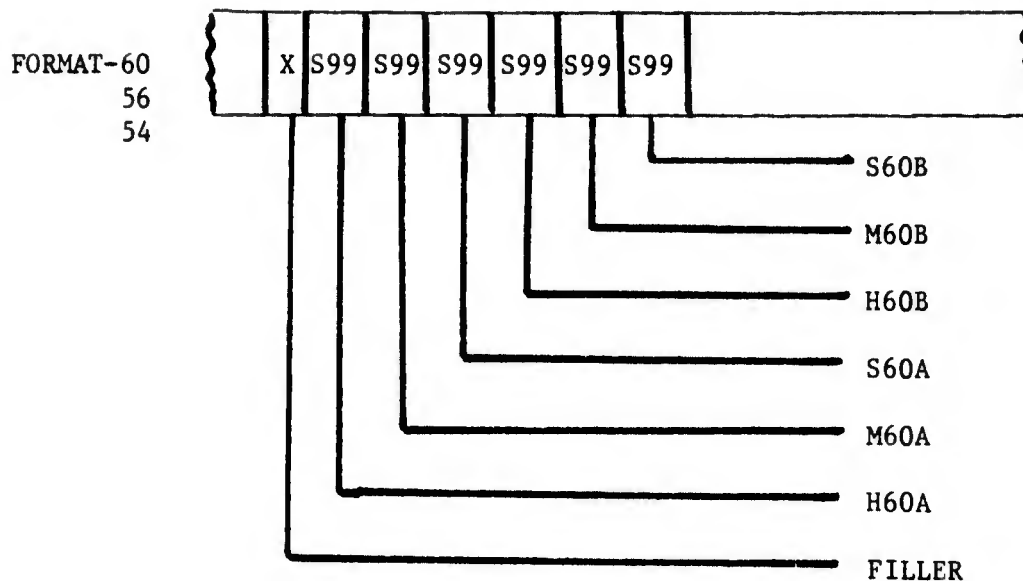


INTEVAL is the number of frames across the leg.

INIT-FR is the initial frame number.

TERM-FR is the terminal frame number.

If the time option is used we may define the leg rectangle by either format 47, 43 or 41 and continue with the time data.



H60A is the hours field.

M60A is the minutes field.

S60A is the seconds field of the time of the initial point of the leg.

H60B, M60B, and S60B are the corresponding fields for the terminal point of the leg.

H56A, ..., S56B are the corresponding fields for FORMAT-56; and  
H54A, ..., S54B are the corresponding fields for FORMAT-54.

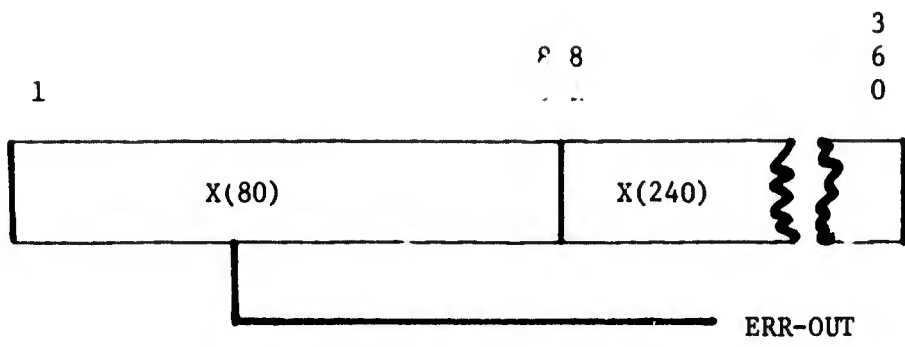
Formats 60, 56, and 54 are formed by adding the time data to formats 47, 43 and 41 respectively.

All entries in IN-DATA are display usage.

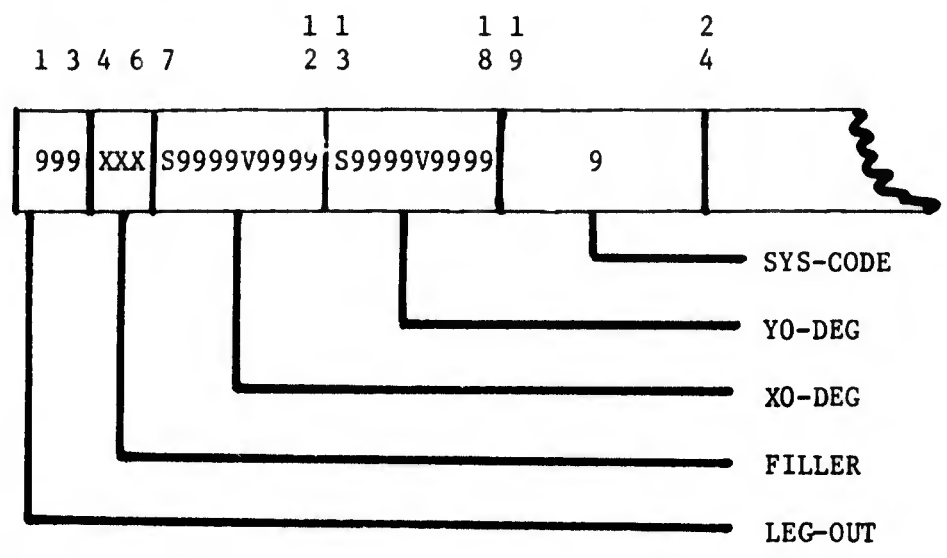
The OUT-DATA field contains the output either an error message or generated data which is to be used as input to the Route Search special operator.

If an error is discovered in the input data, an error message is returned in OUT-DATA as shown on next page.

ENCLOSURE B

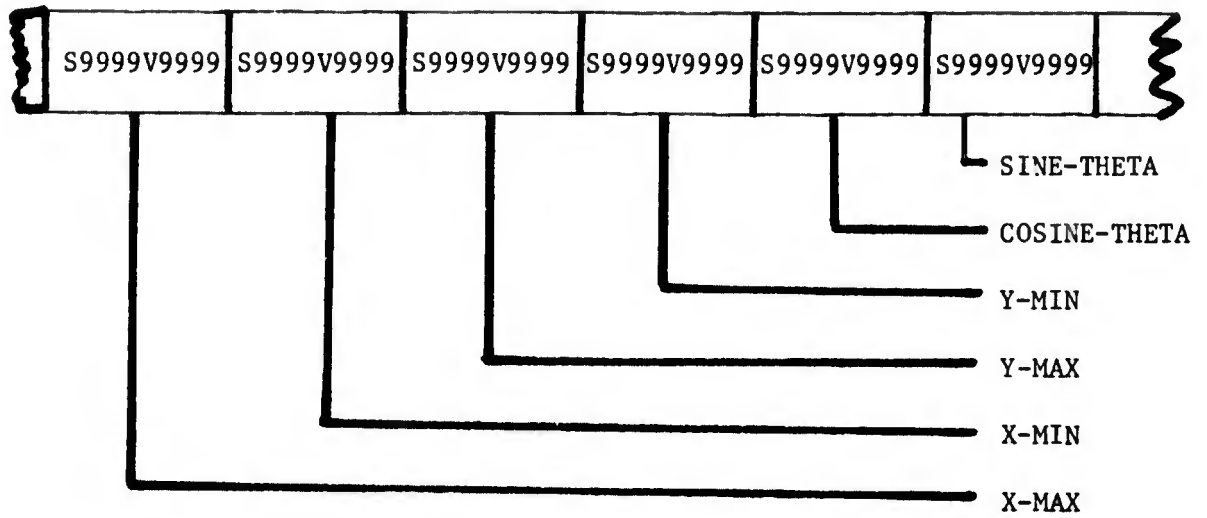


If no error is discovered the OUT-DATA field has the format shown below.

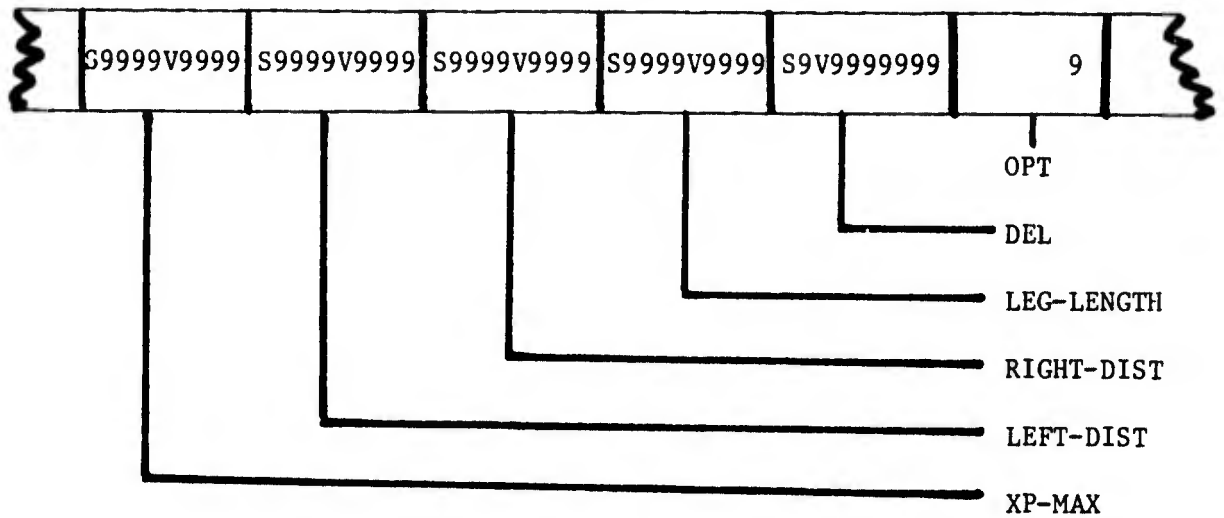


ENCLOSURE B

2	3 3	3 3	4 4	4 4	5 5	6
5	0 1	6 7	2 3	8 9	4 5	0



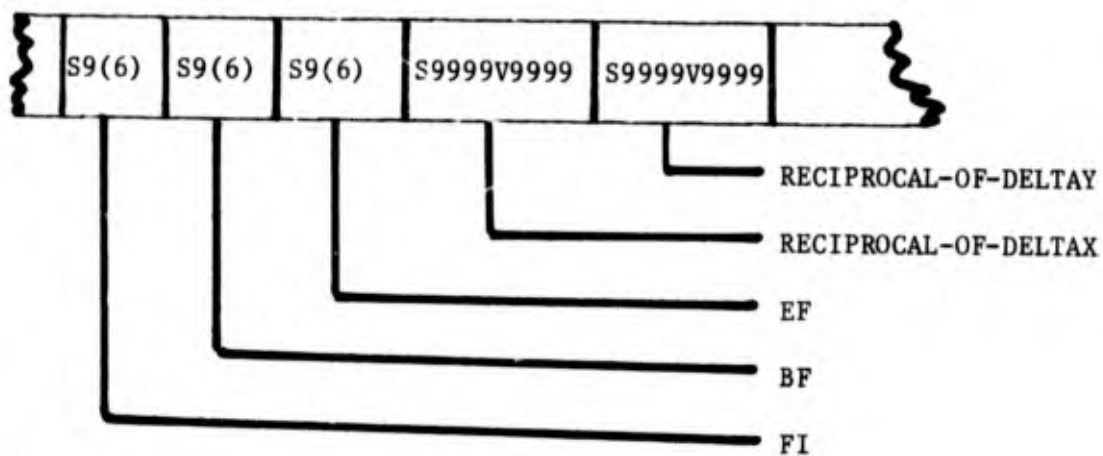
6	6 6	7 7	7 7	8 8	9 9	9
1	6 7	2 3	8 9	4 5	0 1	6



ENCLOSURE B

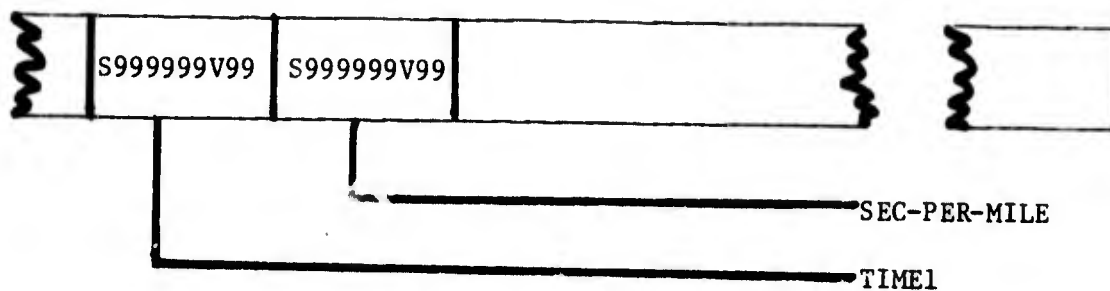
In case of frame option:

	1 1	1 1	1 1	1 1	1
9	0 0	0 0	1 1	2 2	2
7	2 3	8 9	4 5	0 1	6




In case of time option:

	1 1	1	3
9	0 0	0	6
7	2 3	8	0



ENCLOSURE B

Field Name	Usage	Description
LEG-OUT	Display	leg identifying number
X0-DEG	Floating point	Longitude of point 0 in degrees
Y0-DEG	Floating point	Latitude of point 0 in degrees
SYS-CODE	binary	Code indicating coordinate frame used (see diagram below)
X-MAX	Floating point	Maximum X for gross test (see below)
X-MIN	Floating point	Minimum X for gross test (see below)
Y-MAX	Floating point	Maximum Y
Y-MIN	Floating point	Minimum Y
COSINE-THETA	Floating point	Cosine of angle of rotation theta (see below)
SINE-THETA	Floating point	Sine of theta
XP-MAX	Floating point	Length of leg to extension area
LEFT-DIST	Floating point	left displacement of leg rectangle
RIGHT-DIST	Floating point	Right displacement of leg rectangle
LEG-LENGTH	Floating point	Length of leg
DEL	Floating point	Cosine of the average latitude
OPT	binary	Option code = 1 frame 2 time 0 otherwise
FI	binary	 See FRAMX description
BF	binary	
EF	binary	
RECIPROCAL-OF-DELTAX	Floating point	
RECIPROCAL-OF-DELTAY	Floating point	

ENCLOSURE B

Field Name	Usage	Description
TIME1	Floating point	} See TIMEX description
SEC-PER-MILE	Floating point	

The possible coordinate frames are shown below.

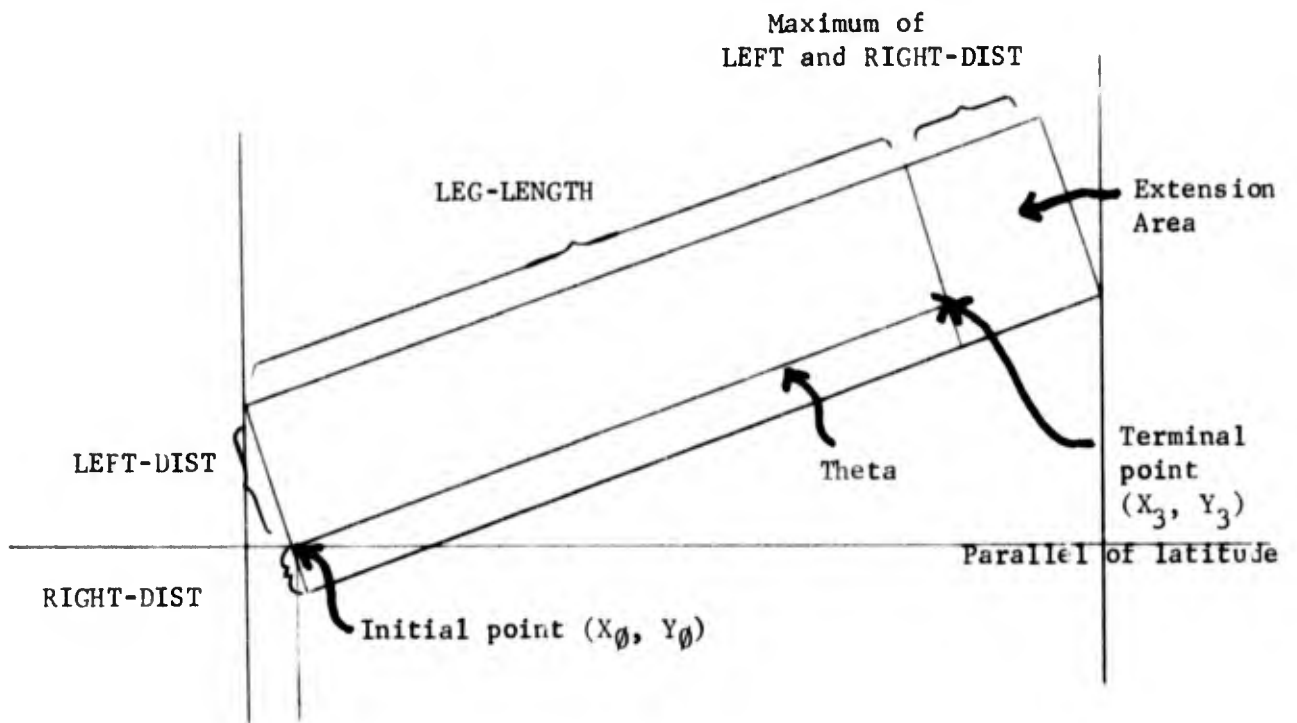
	-180		Greenwich Meridian	
		-90	+90	+180
SYS-CODE=1	<hr/>			
	180°W	90°W	90°E	180°E
	Greenwich Meridian			
		+90	+180	+270
SYS-CODE=2	<hr/>			
	0°	90°E	180°E	90°W
				0°

Example 170°W = -170, SYS-CODE=1;  
190, SYS-CODE=2.

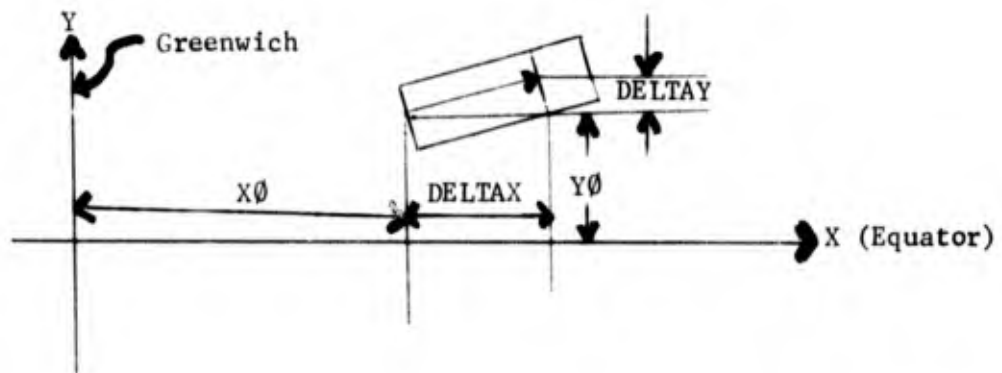
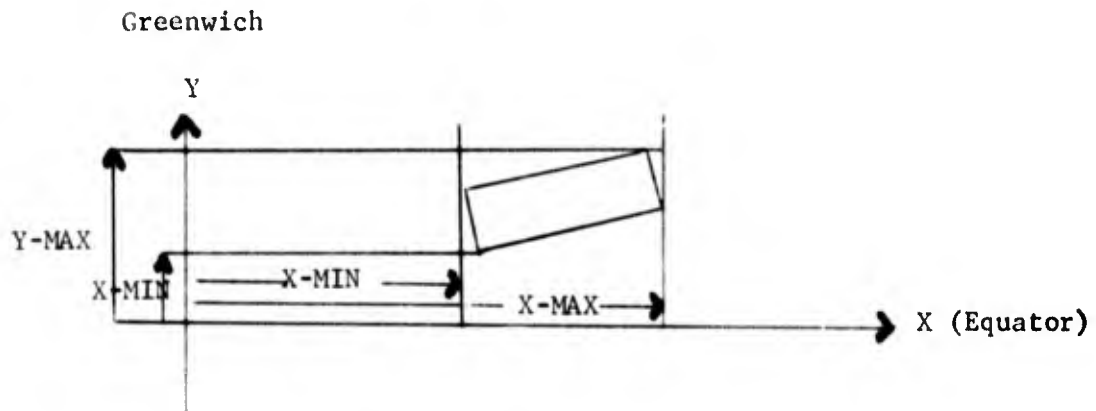
ENCLOSURE B



The rectangle constructed about leg is shown below:



The maximum, minimum values of X and Y are depicted below:



ENCLOSURE B

(b) Description.

(1) A planar model is used throughout this program; however, the convergence of the meridians of longitude is accounted for by multiplying distance along parallels of latitude by the cosine of the average latitude.

(2) The formulas used are presented below:

Given A in radians the truncated series below is used to compute

$$\cosine a = 1 - \frac{a^2}{2!} + \frac{a^4}{4!} - \frac{a^6}{6!} + \frac{a^8}{8!} - \frac{a^{10}}{10!}$$

The average latitude in radians is computed by:

$$AVG = [.0174533 \text{ rad/deg} \times \frac{1}{2} \times (\text{Lat of pt1} + \text{Lat of pt2})]^2$$

The cosine of the average latitude is computed by:

$$DEL = 1 - \frac{1}{2} AVG + \frac{1}{4!} AVG^2 - \frac{1}{6!} AVG^3 + \frac{1}{8!} AVG^4 - \frac{1}{10!} AVG^{10}$$

The X and Y coordinates are computed by means of the following equations:

$$X(\text{nautical miles}) = X(\text{degrees}) \quad 60 \text{ (nm/deg)} \quad \cosine a$$

where: a the average latitude of the leg.

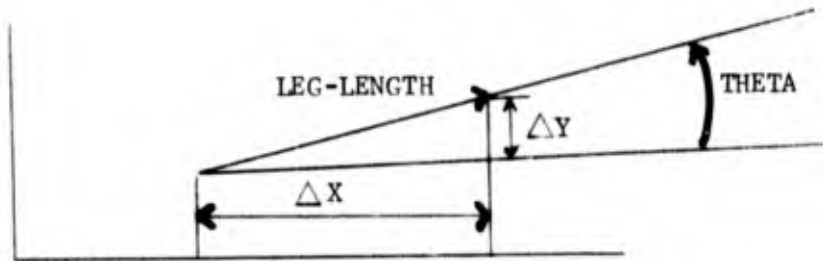
and

$$Y(\text{nautical miles}) = Y(\text{degrees}) \quad 60 \text{ (nm/deg)}$$

ENCLOSURE B

The length of the leg is computed by:

$$\text{LEG-LENGTH} = \sqrt{(\Delta X)^2 + (\Delta Y)^2}$$



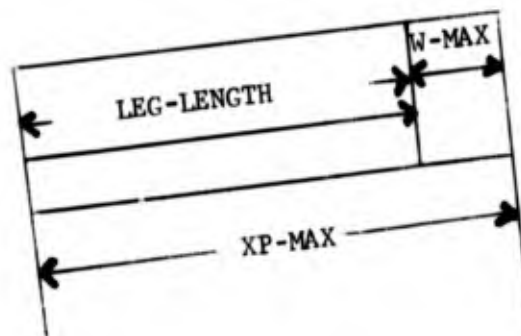
and the functions SINE THETA and COSINE THETA are computed by

$$\begin{aligned}\text{SINE-THETA} &= \Delta Y / \text{LEG-LENGTH} \\ \text{COSINE-THETA} &= \Delta X / \text{LEG-LENGTH}\end{aligned}$$

(NOTE: HIS COBOL did not properly square  $\Delta X$  and  $\Delta Y$  and therefore, GMAP patches were inserted for this purpose.)

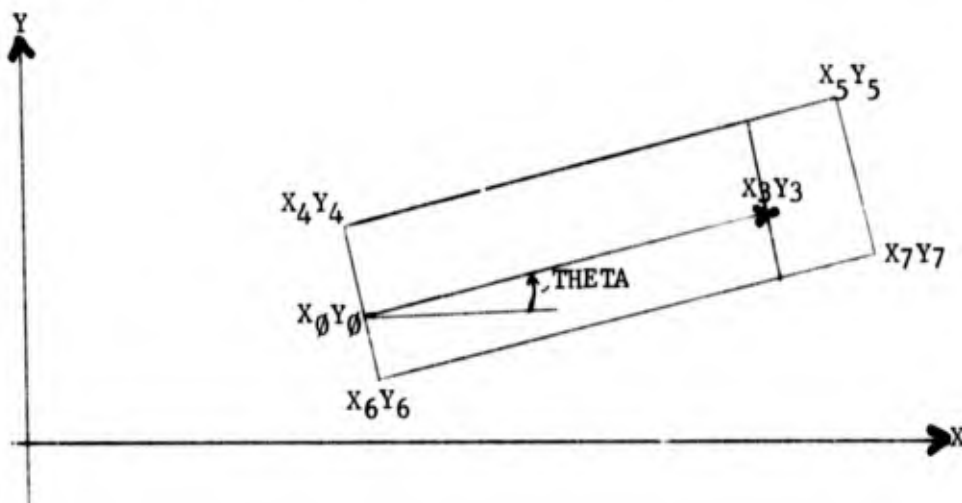
The leg rectangle is extended by the maximum of the left and right distances (LEFT-DIST, RIGHT-DIST). This maximum is called W-MAX.

ENCLOSURE B



In order to test the file point quickly on the Route Search special operator, the limits of latitude and longitude are computed in this program and passed to the Route Search special operator. First, the X,Y coordinates of the corner points are computed. Next, the maximum and minimum values of X and Y are found. Finally, these maximum and minimum X and Y are converted to degrees. The diagram below shows the labelling of the corner points.

#### Corner Points



ENCLOSURE B

(c) Limitations.

(a) All coordinates must be in standard 15 character format.

(b) Accuracy decreases when the leg or its associated data is more than 300 nautical miles.

(c) Leg coordinates cannot extend across the poles.

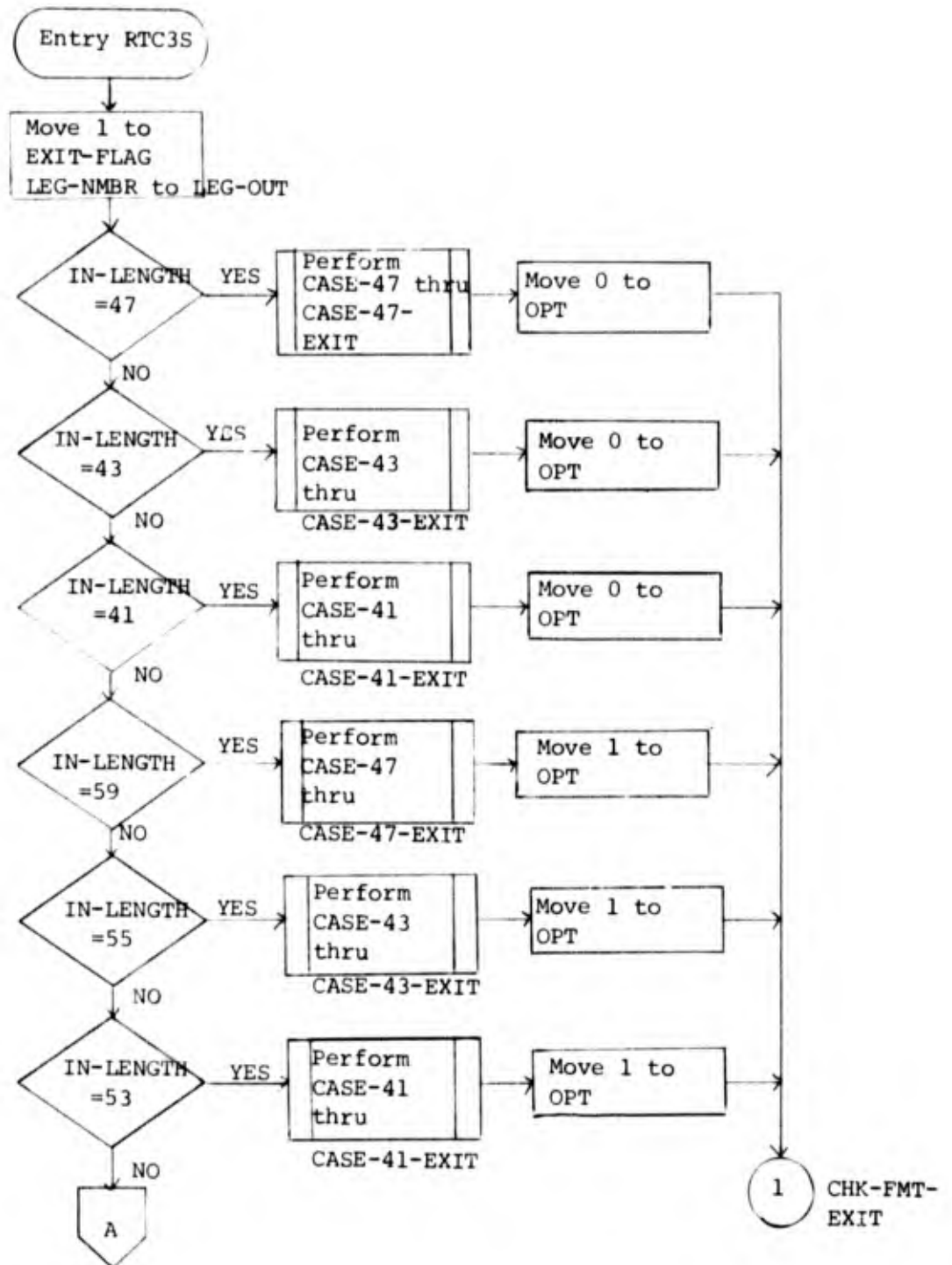
(d) Beginning and ending times are not continuous over the international date line nor through midnight.

(e) Number of leg is limited by the system (main program) to a maximum of 219 (depending on the format).

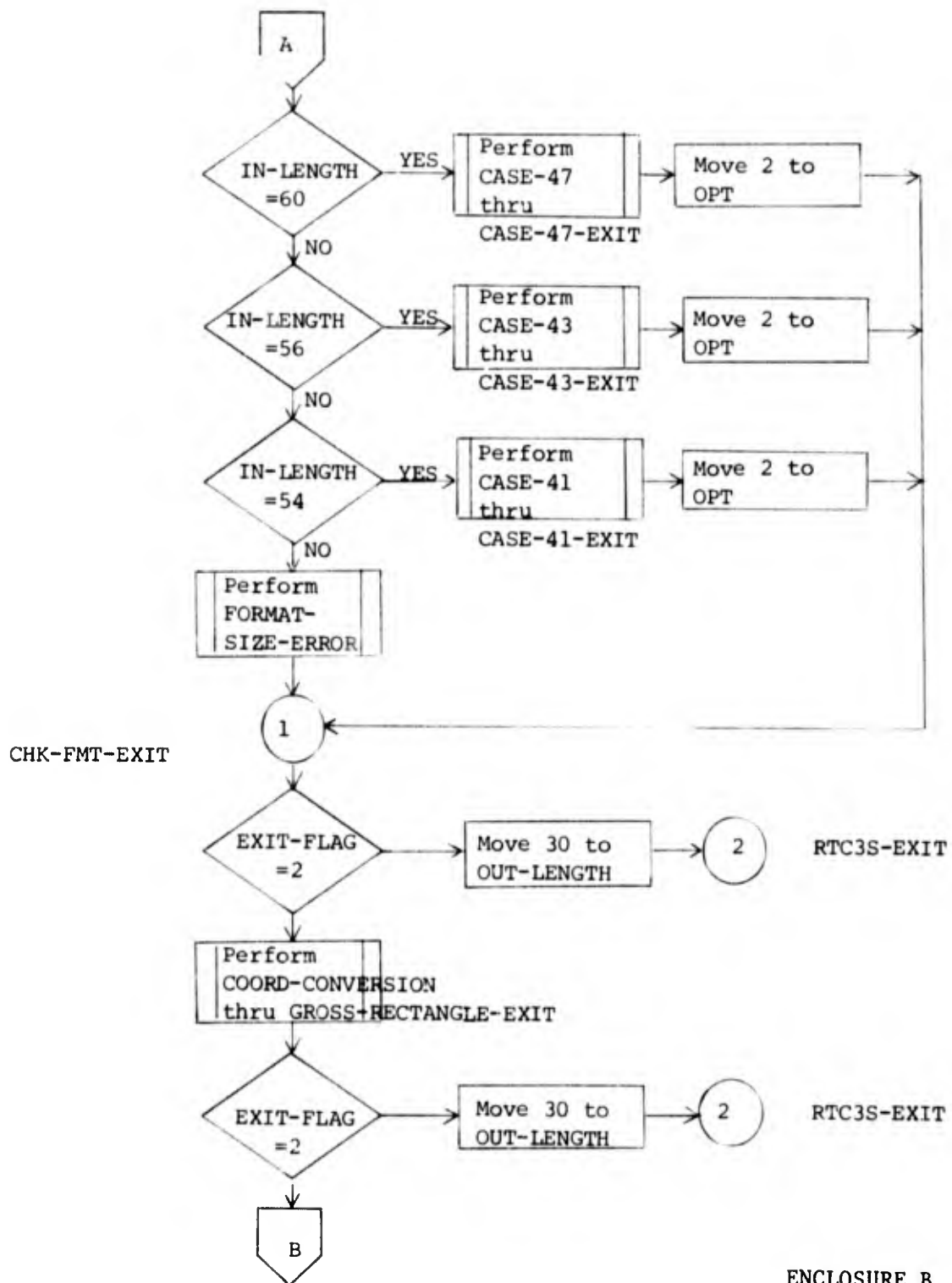
(f) Pitch of camera is assumed to be zero.

ENCLOSURE B

INITIALIZE

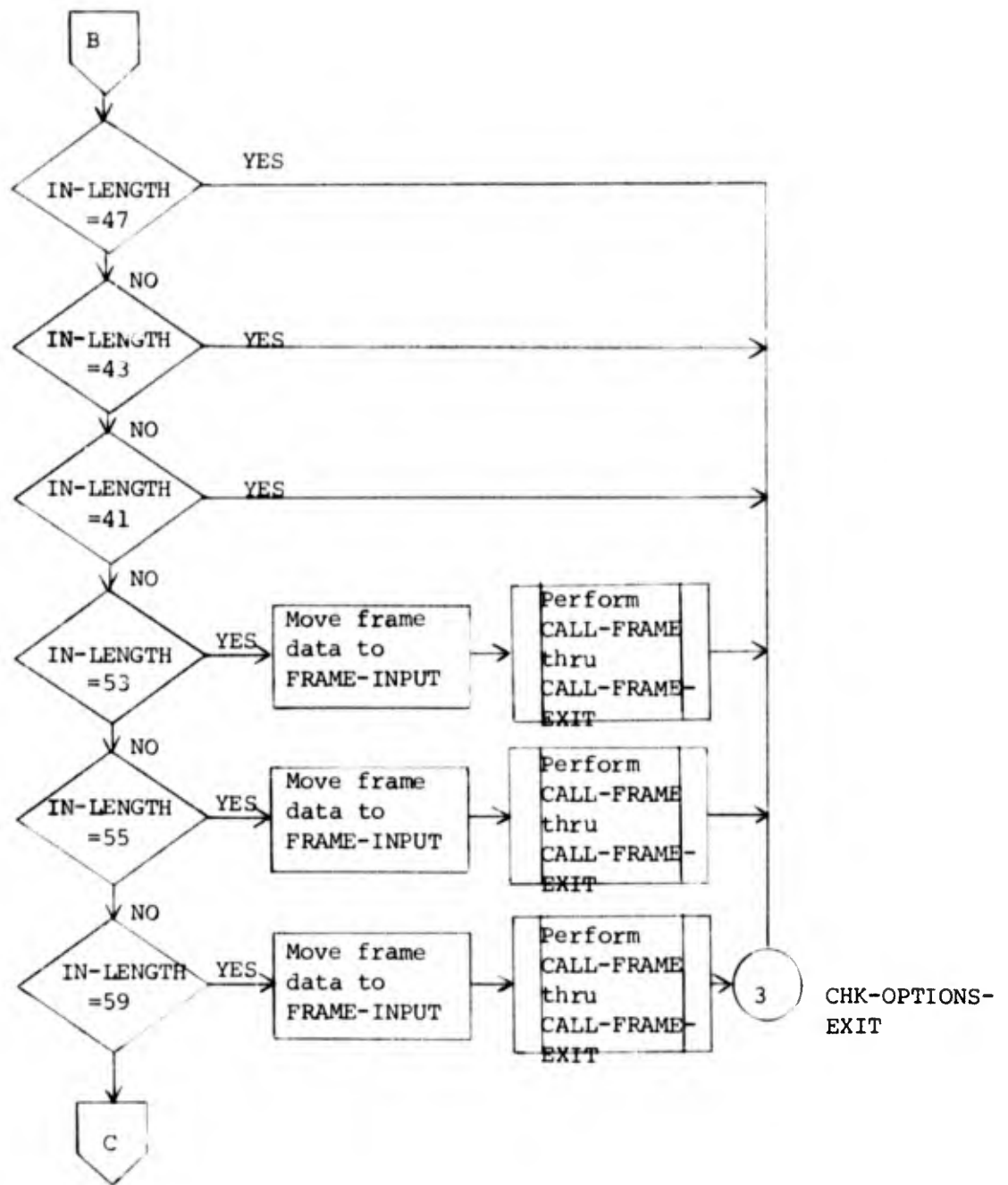


ENCLOSURE B

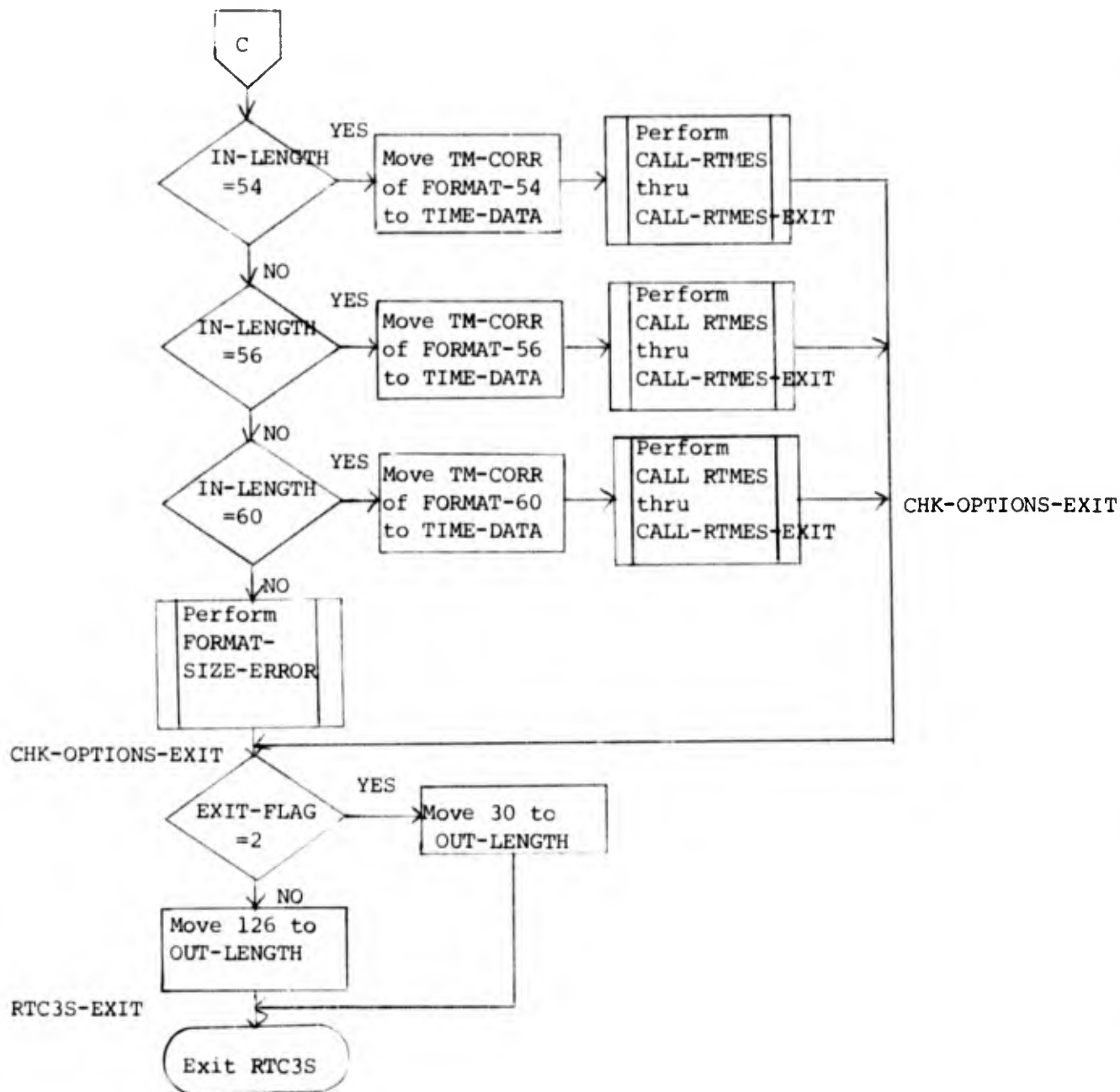


ENCLOSURE B

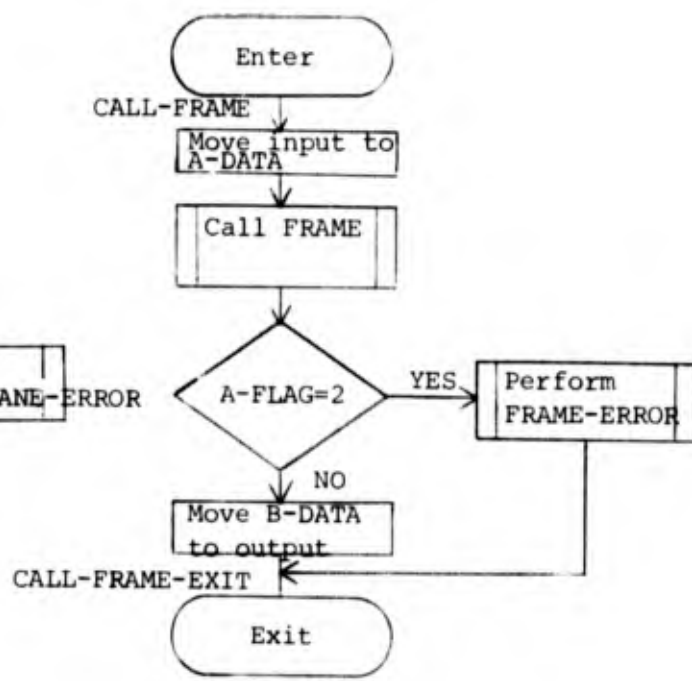
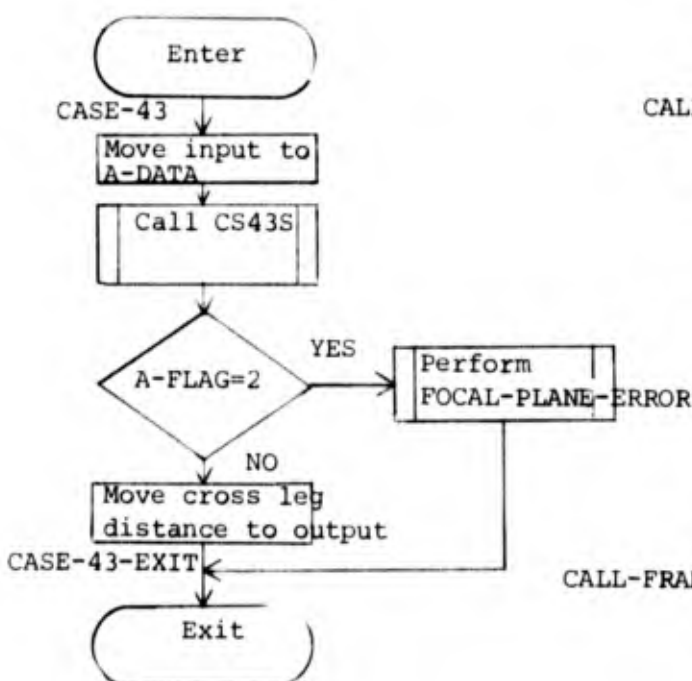
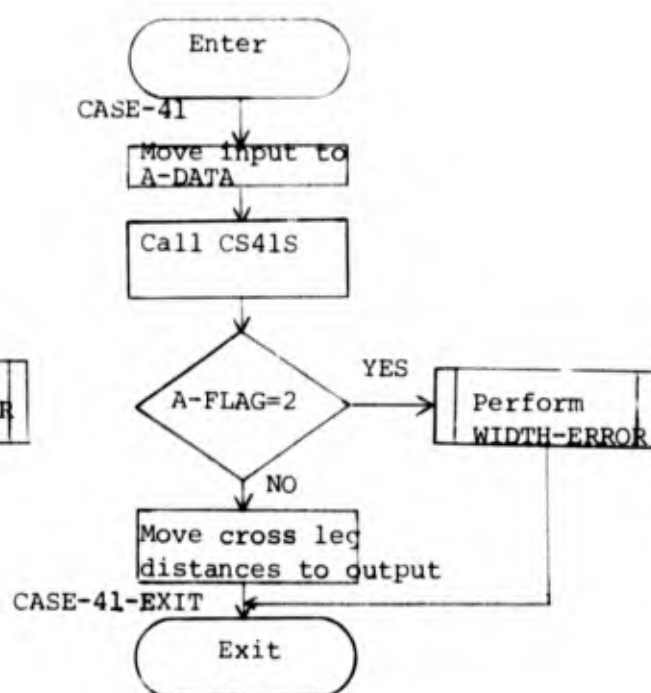
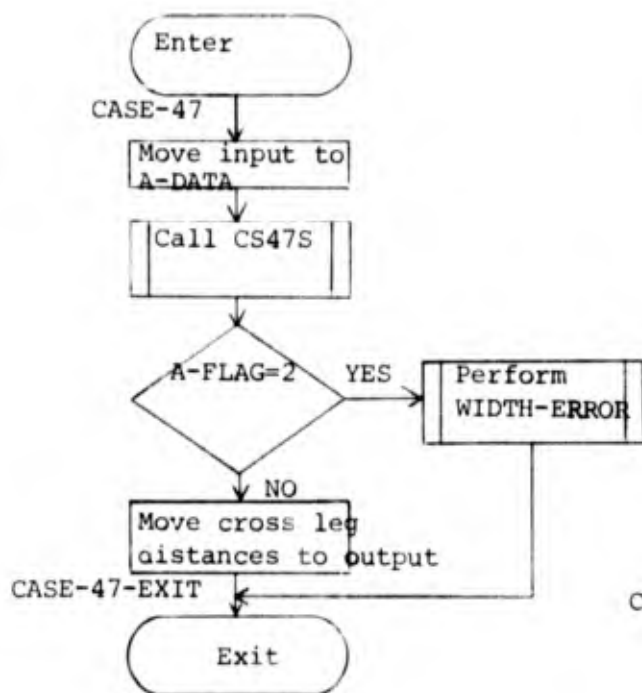


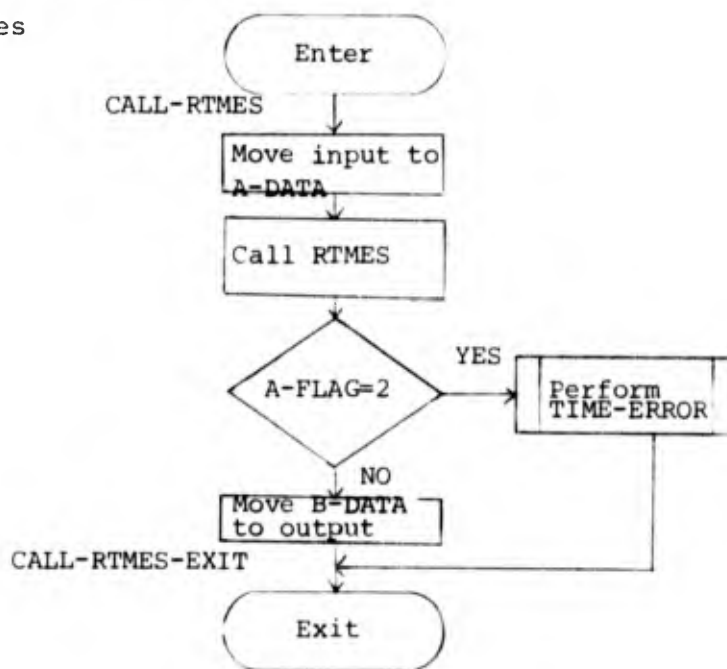
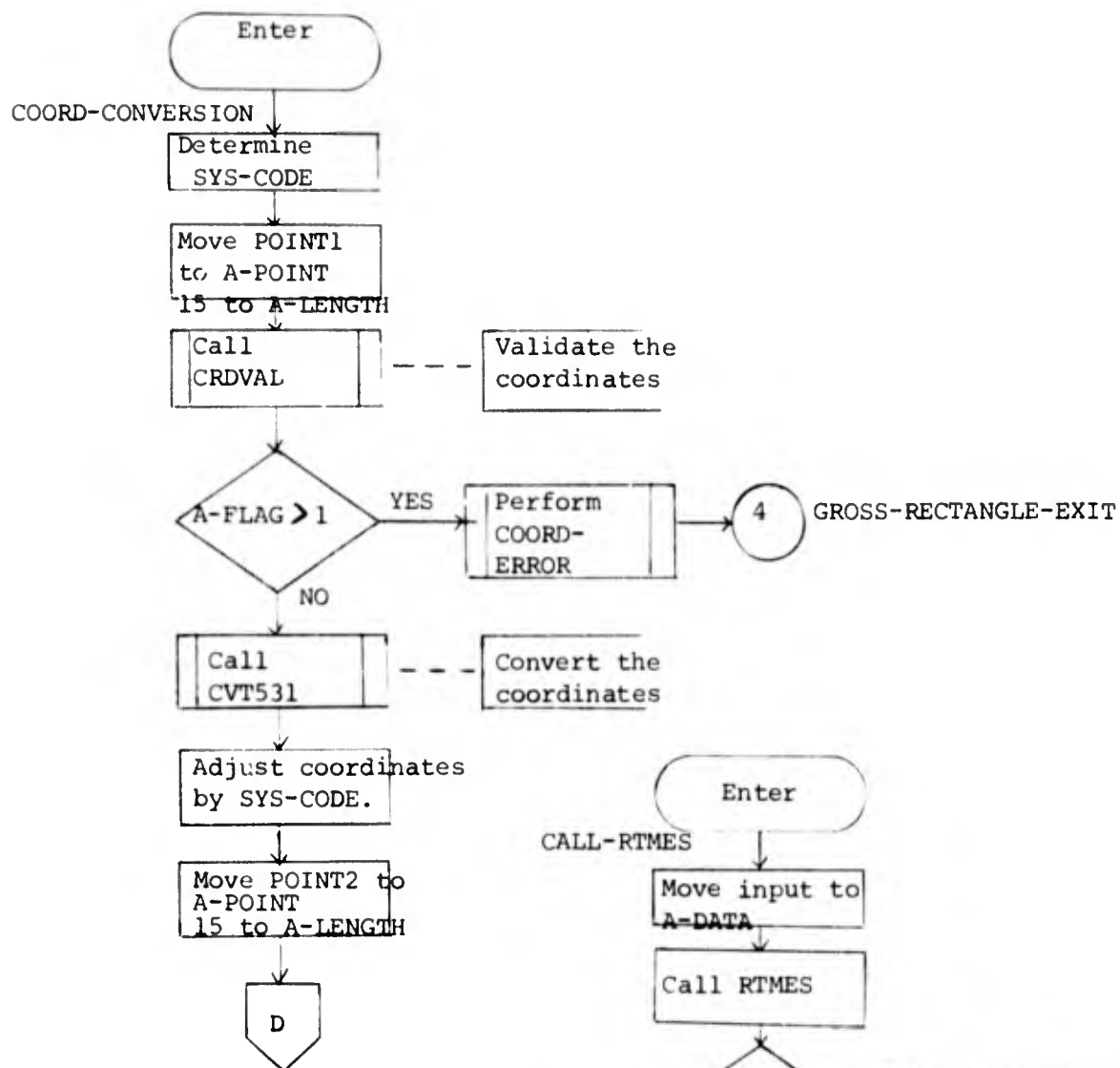


ENCLOSURE B

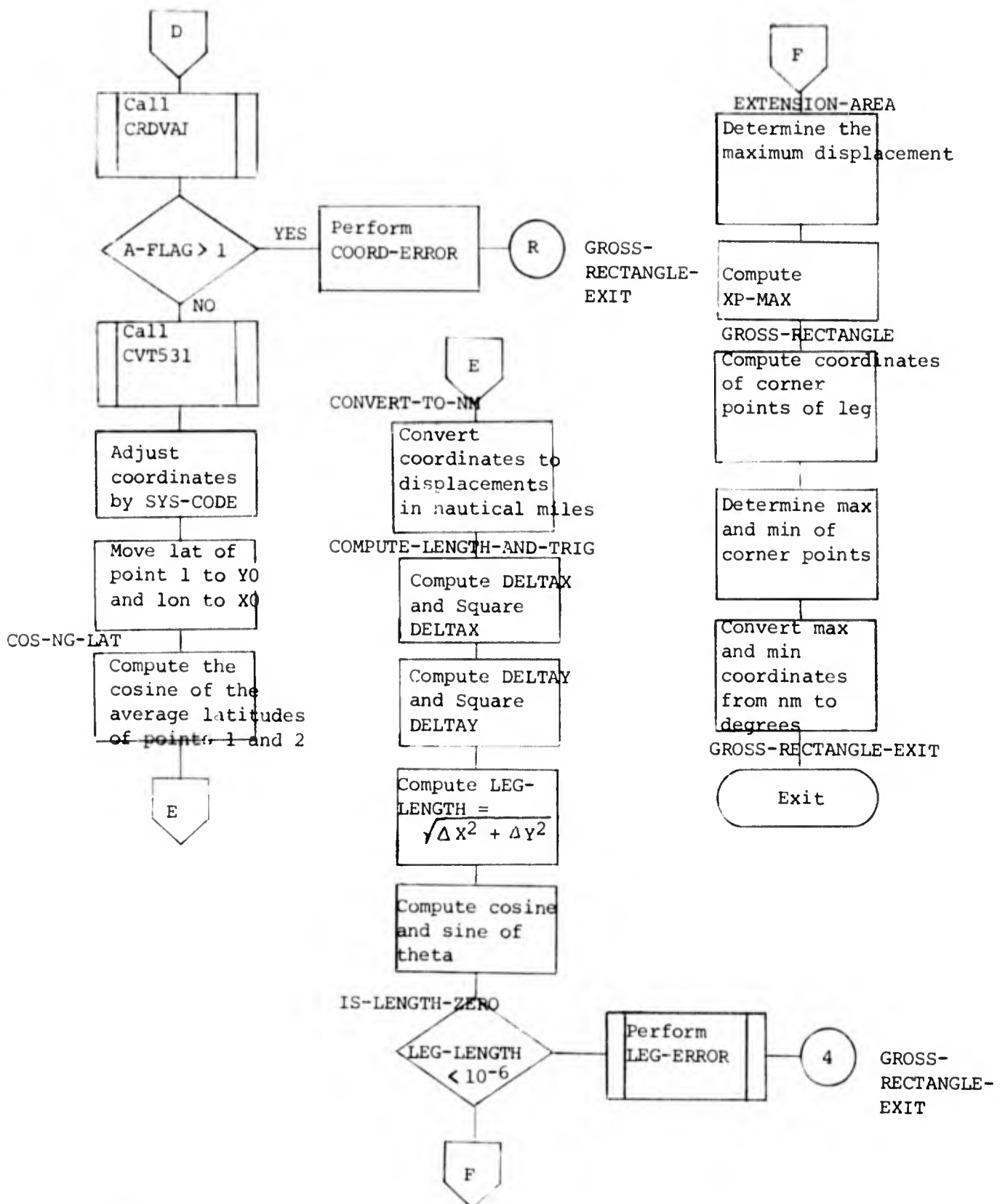


ENCLOSURE B





ENCLOSURE B



(2) CS41X

(a) Abstract.

1. Function.

a. This subroutine handles the case where the Route Search leg being defined has only one displacement left or right.

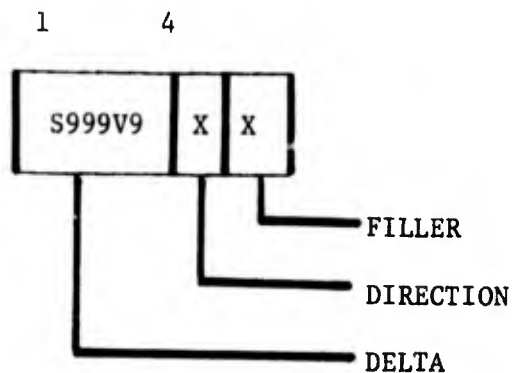
b. Given a distance in nautical miles and a direction, "L" or "R", this program validates the input and converts the distance to floating point format.

2. Calling Sequence.

Entry CS41S USING A-DATA,  
GIVING B-DATA, A-FLAG

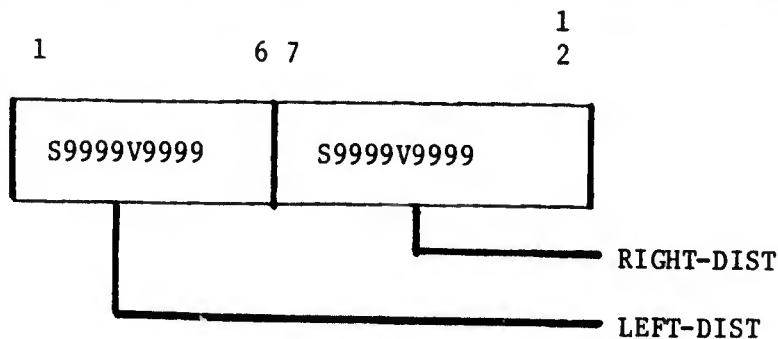
where:

A-DATA contains the input data in the format shown below:



where DELTA is the cross-leg distance and is Display usage.

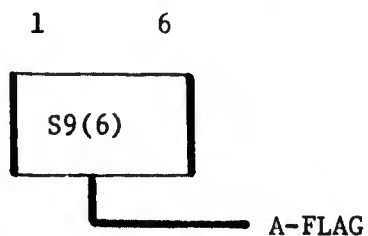
B-DATA contains the output data in the format shown below:



ENCLOSURE B

where RIGHT-DIST and LEFT-DIST are the right and left cross-leg distances and are in computational-2\* (floating point) usage.

A-FLAG contains a flag indicating whether the input data was valid or not. The format is shown below:



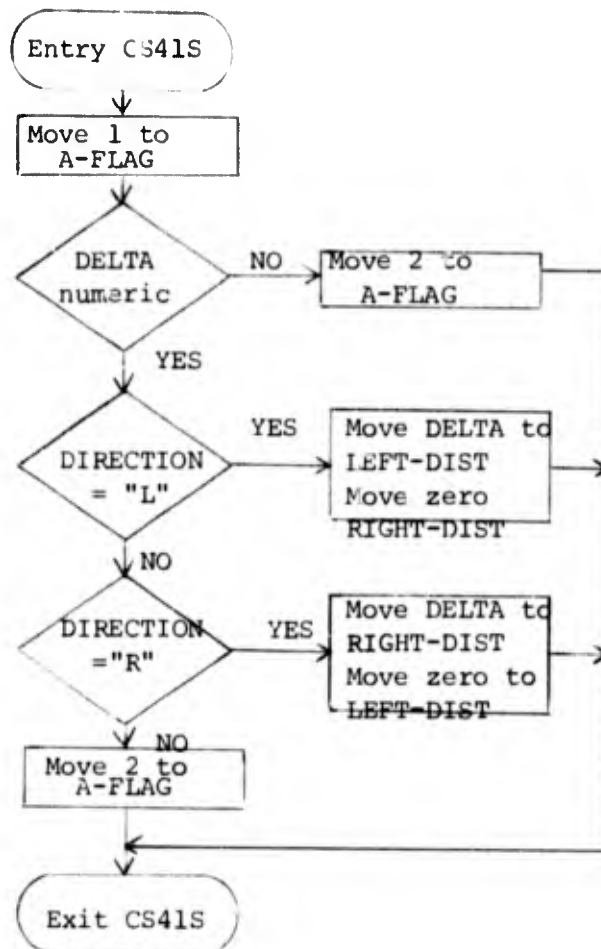
A-FLAG is in computational-1\* (binary) usage and has the values:

A-FLAG =  $\left\{ \begin{array}{l} 1, \text{ successful execution} \\ 2, \text{ invalid data} \end{array} \right.$

(b) Description. DELTA is tested for numeric and if it is numeric, it is moved to RIGHT-DIST, if DIRECTION is "R" or LEFT-DIST if DIRECTION is "L". If direction is neither "R" nor "L" A-FLAG is set to 2.

\* Refers to HIS COBOL.

ENCLOSURE B



ENCLOSURE B



(3) CS43X

(a) Abstract.

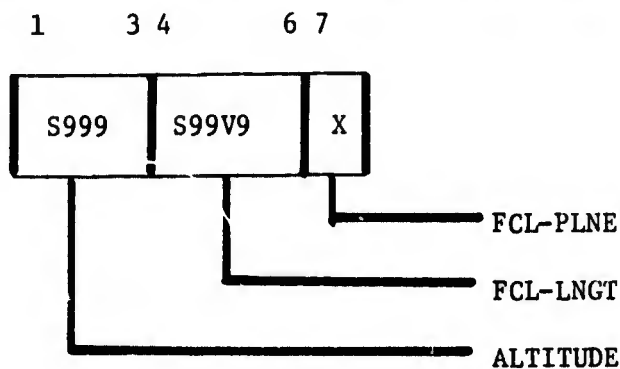
1. Function. This subroutine handles the case where the Route Search leg width is defined by altitude, focal length, and focal plane size. The subroutine validates the input parameters and computes the right and left distances, and returns them in floating point format.

2. Calling Sequence.

ENTRY CS43S USING A-DATA  
GIVING B-DATA, A-FLAG.

where:

A-DATA contains the input data in the format shown below:



where

ALTITUDE is the altitude in thousands of feet (usage is Display).

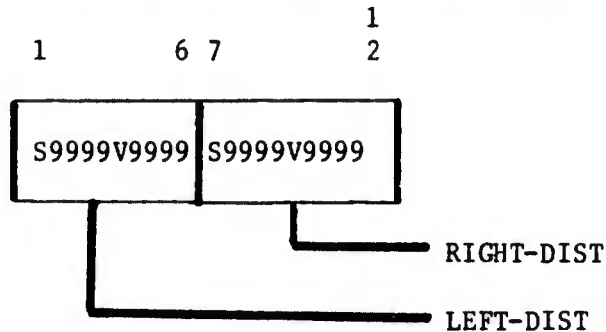
FCL-LNGT is the focal length in inches (usage is Display).

FCL-PLNE is the focal plane code (usage is Display).

$$\text{FCL-PLNE} = \begin{cases} \text{A, 4.5 inches} \\ \text{B, 9.0 inches} \end{cases}$$

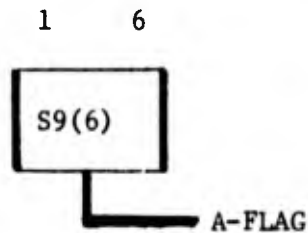
ENCLOSURE B

B-DATA contains the output data in the format shown below:



where RIGHT-DIST and LEFT-DIST are the right and left cross-leg distances and are in computational-2\* (floating point) usage.

A-FLAG contains a flag indicating whether the input data was valid or not. The format is shown below:



A-FLAG is in computational-1\* (binary) usage and has the values:

$$A-FLAG = \begin{cases} 1, & \text{successful execution} \\ 2, & \text{invalid data} \end{cases}$$

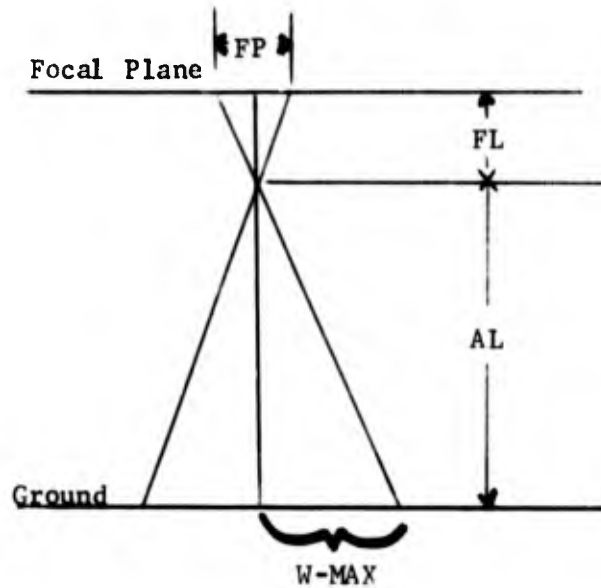
(b) Description. The input data is validated and then: ALTITUDE is moved to AL, FC-LNGT is moved to FL, and 4.5 or 9.0 is moved to FP depending on whether FCL-PLNE equals A or B respectively. The cross-leg distance W-MAX is computed as follows:

$$W-MAX = \frac{FP * AL}{12.1522 * FL}$$

\* Refers to HIS COBOL.

ENCLOSURE B

The formula follows from the law of similar triangles:

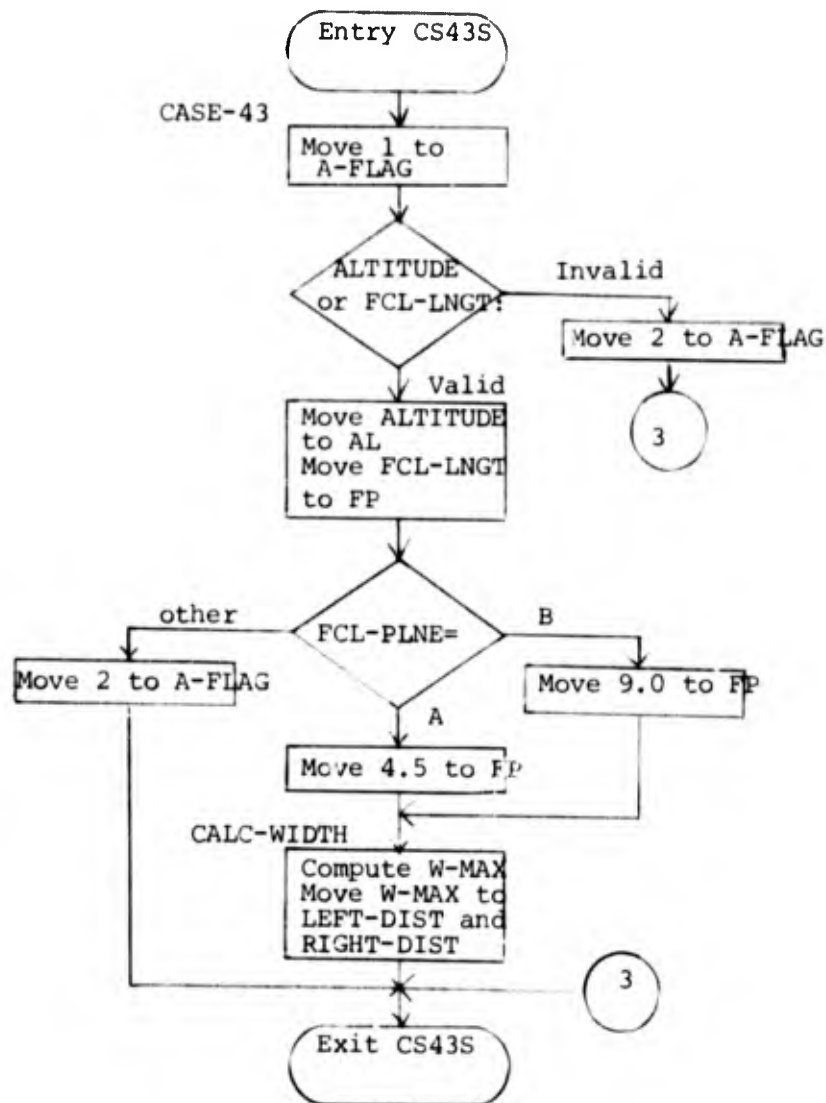


$$W-MAX \text{ nm} = \frac{1}{2} \frac{FP \text{ in} \times AL \text{ thousands of feet}}{FL \text{ in} \times 6.0761 \text{ thousands of feet/nm}}$$

where the units in the above are given by: nm, nautical miles;  
in, inches.

The conversion factor 6076.1 feet/nm is also used.

ENCLOSURE B



ENCLOSURE B

(4) CS47X

(a) Abstract.

## 1. Function.

a. This subroutine handles the case where the Route Search leg being defined has both a right and left displacement (not necessarily equal).

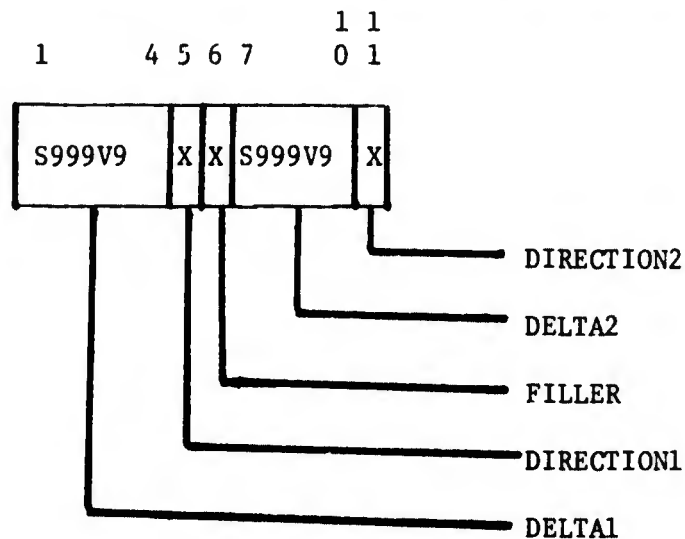
b. Given two distances in nautical miles and directions, "L" and "R", this program validates the input and converts the distance to floating point format.

## 2. Calling Sequence.

ENTRY CS47S USING A-DATA,  
GIVING B-DATA, A-FLAG.

where:

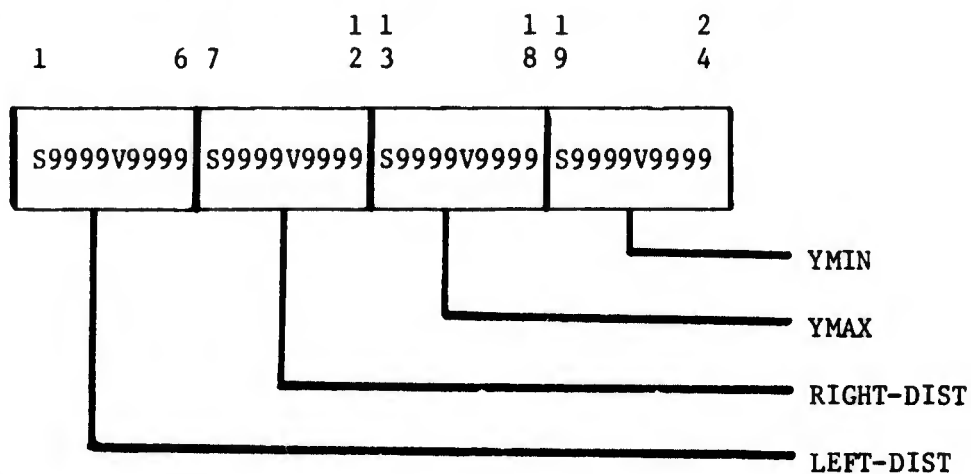
A-DATA contains the input data in the format shown below:



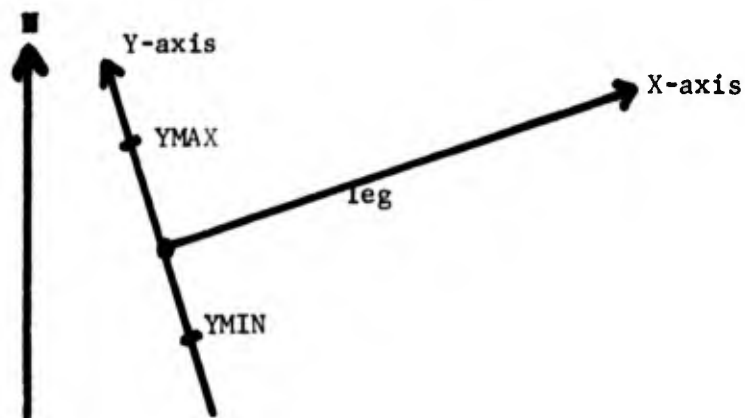
where DELTA1 and DELTA2 are the cross-leg distances and are in DISPLAY usage, and DIRECTION1 and DIRECTION2 give the direction of the corresponding displacements, one must be "L", the other "R".

ENCLOSURE B

B-DATA contains the output data in the format shown below:



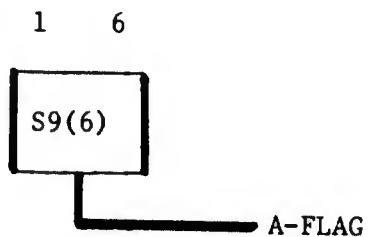
where RIGHT-DIST and LEFT-DIST are the right and left cross-leg distances and are in computational-2 (floating point) usage, YMIN and YMAX are the limits on the Y-axis of a coordinate frame. The coordinate frame is shown below:



YMAX should not be negative and YMIN should not be positive, they are both defined to be computational-2 (floating point).

ENCLOSURE B

A-FLAG contains a flag indicating whether the input data whether the input data was valid or not. The format is shown below:

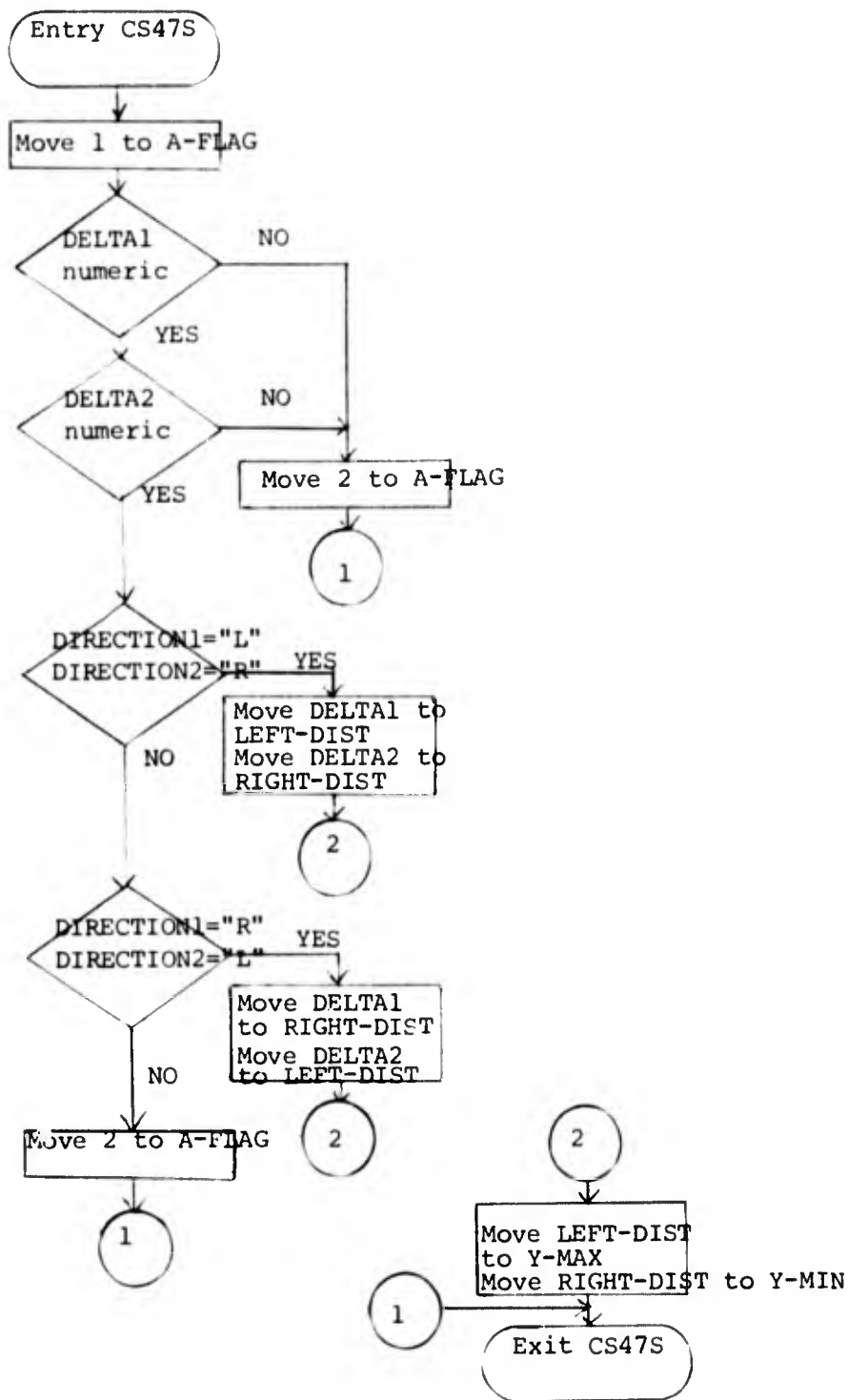


A-FLAG is in computational-1 (binary) usage and has the values:

$$A-FLAG = \begin{cases} 1, & \text{successful execution} \\ 2, & \text{invalid data} \end{cases}$$

(b) Description. DELTA1 and DELTA2 are tested to determine if they contain numeric data. If DELTA1 and DELTA2 are numeric they are moved to LEFT-DIST and RIGHT-DIST depending on DIRECTION1 and DIRECTION2. LEFT-DIST is moved to YMAX and -RIGHT-DIST to YMIN.

ENCLOSURE B



ENCLOSURE B



(5) FRAMX

(a) Abstract.

1. Function.

a. This subroutine is used by the Route Search preprocessor R1C3S, to handle frame data for the frame option.

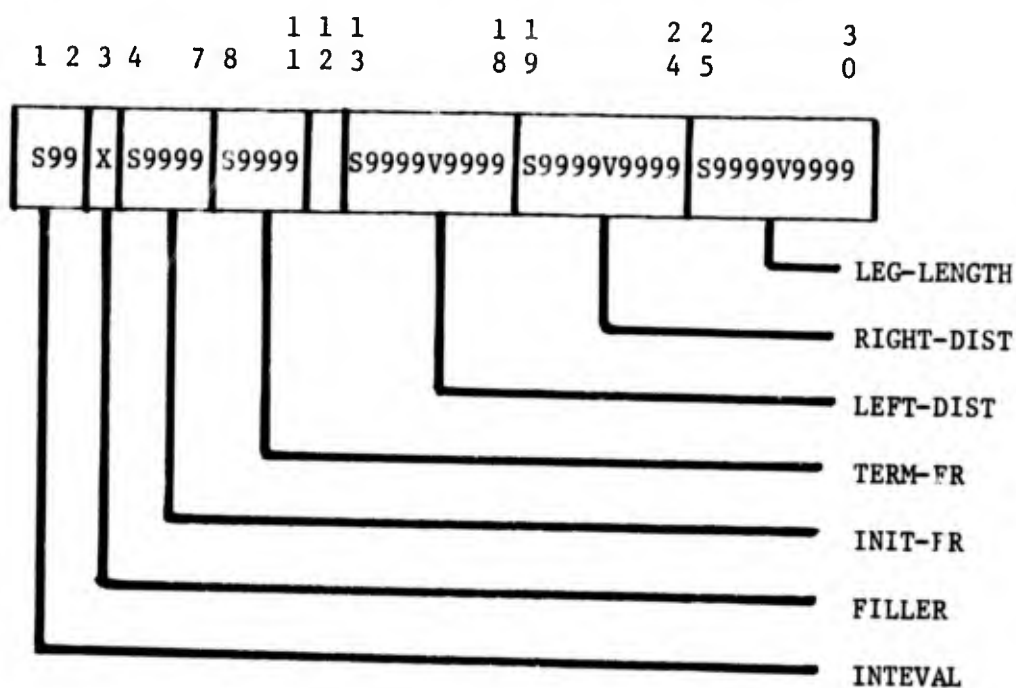
b. Given the frame internal beginning and ending frame numbers and the dimensions of the leg, this program computes the frame parameters to be used by the Route Search special operator.

2. Calling Sequence.

ENTRY POINT FRAME USING A-DATA  
GIVING B-DATA, A-FLAG.

where:

A-DATA contains the input data in the format shown below:



ENCLOSURE B

INTERVAL is the frame interval (usage is Display).

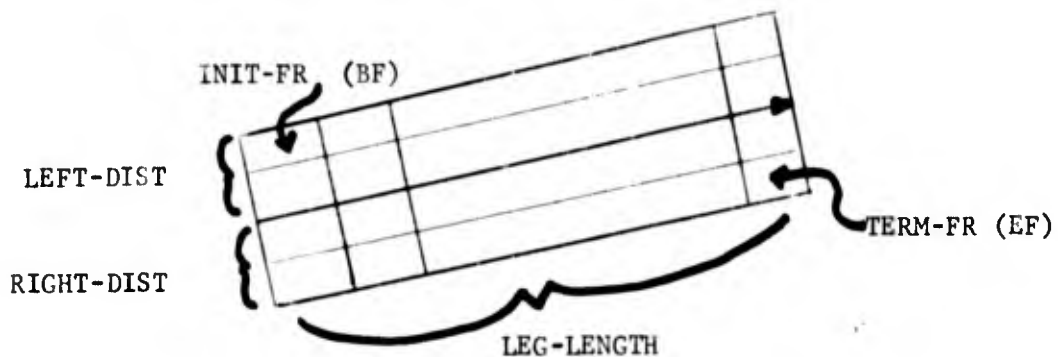
INIT-FR is the initial frame number (usage is Display).

TERM-FR is the terminal frame number (usage is Display).

LEFT-DIST is the left distance in nautical miles of the leg rectangle.  
It is in computational-2 (floating point) usage.

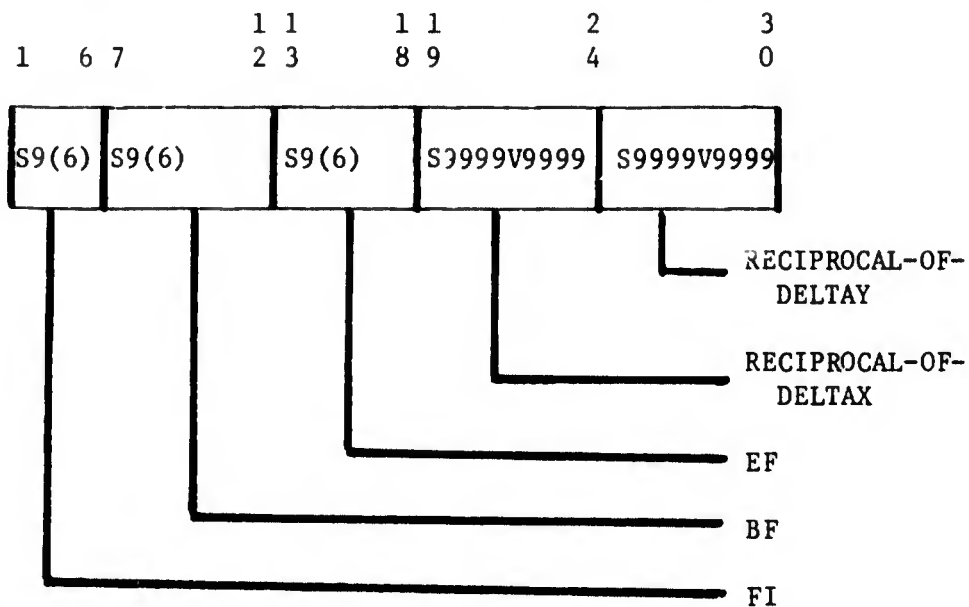
RIGHT-DIST is the right distance in nautical miles of the leg rectangle.  
It is in computational-2 (floating point) usage.

LEG-LENGTH is the length of the leg in nautical miles of the leg rectangle.  
It is in computational-2 (floating point) usage.



ENCLOSURE B

B-DATA contains the output data in the format shown below:



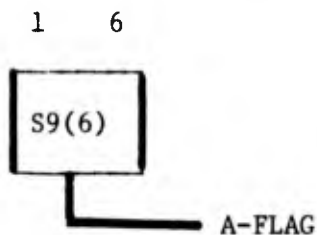
FI is the frame internal converted to binary (computational-1).

BF is the beginning frame number in binary (computational-1).

EF is the ending frame number in binary (computational-1).

RECIPROCAL-OF-DELTAX and RECIPROCAL-OF-DELTAY are special parameters in floating point (computational-2).

A-FLAG contains a flag indicating whether the input data was valid or not. The format is shown below:



A-FLAG is in binary (computational-1) usage, and has the values:

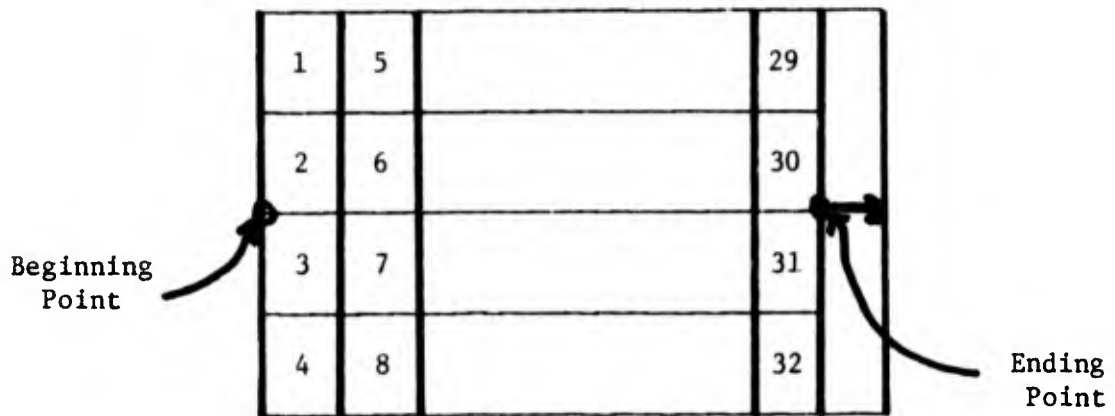
$$A-FLAG = \begin{cases} 1, & \text{successful execution} \\ 2, & \text{invalid data} \end{cases}$$

ENCLOSURE B

(b) Description.

1. INTEVAL, INIT-FR, and TERM-FR are checked for numeric data and if they are numeric they are moved to FI, BF and EF respectively.

2. Since INTEVAL is considered to be the number of frames across the leg INTEVAL should exactly divide the difference between INIT-FR and TERM-FR. This is illustrated below:



In the above example the number of frames across the leg (INTEVAL) is 4. INIT-FR=1 and TERM-FR=32. The total number of frames should be a multiple of INTEVAL. To check this compute

$$\text{INTEGER1} \leftarrow 32 - + 1 = 32$$

(i.e., INTEGER1 is the total number of frames).

$$\begin{aligned} \text{INTEGER2} &\leftarrow \text{INTER1}/\text{FI} \\ &= 32/4 = 8 + \text{no remainder} \end{aligned}$$

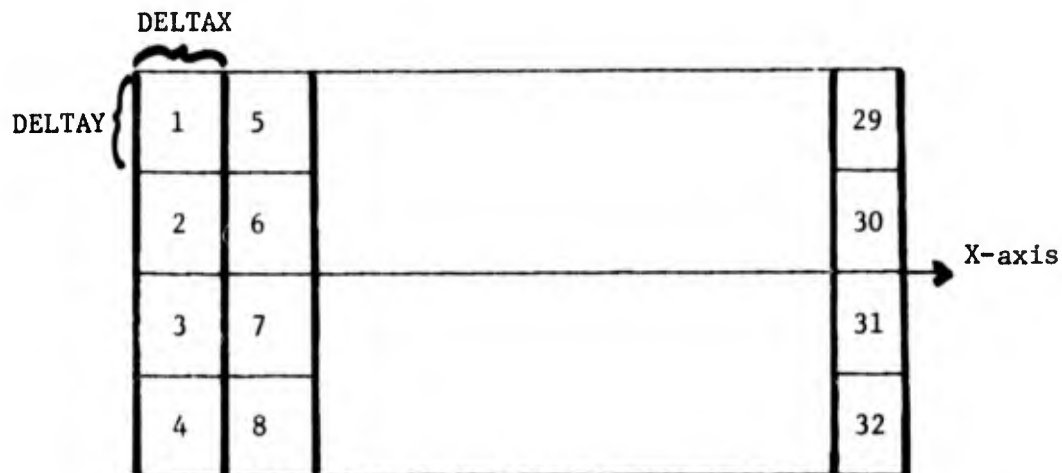
Here the remainder is dropped (i.e., the fraction is terminated).

$$\begin{aligned} \text{INTEGER2} &\leftarrow \text{INTEGER2} \cdot \text{FI} \\ &= 8 \cdot 4 = 32 \end{aligned}$$

Now if INTEGER2 is not equal to INTEGER1, then the total number of frames is not a multiple of the frame internal (INTEVAL, FI), and this is not valid.

ENCLOSURE B

DELTAX and DELTAY are considered to be the dimension of a frame.  
The X-axis is placed along the leg and Y-axis is perpendicular  
to it.

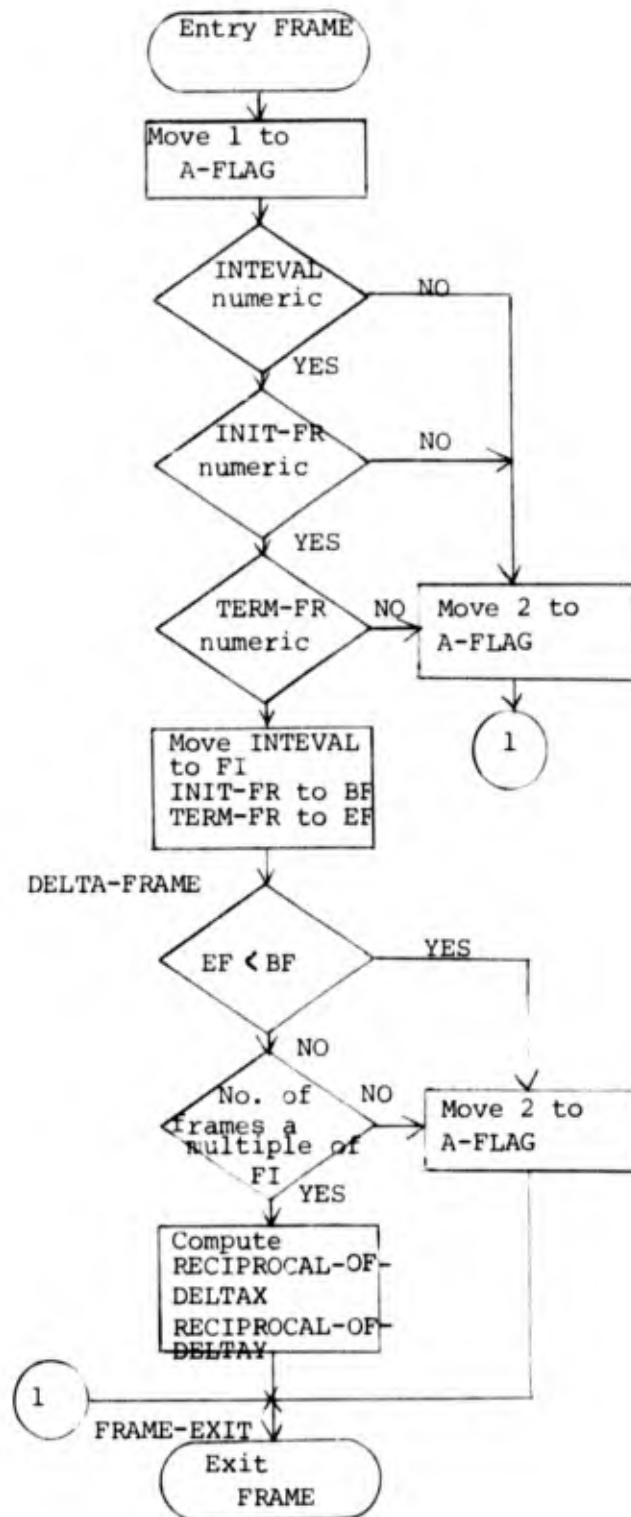


$$\text{RECIPROCAL-OF-DELTAX} = 1/\text{DELTAX} = \text{INTEGER1}/\text{LEG-LENGTH} \quad \text{FI}$$

$$\text{RECIPROCAL-OF-DELTAY} = 1/\text{DELTAY} = \text{FI}/(\text{LEFT-DIST} + \text{RIGHT-DIST})$$

The reciprocals presented above are used in the Route Search special operator.

ENCLOSURE B



ENCLOSURE B

(6) TIMEX

(a) Abstract.

1. Function.

a. This subroutine is used by the Route Search preprocessor RTC3S, to handle time data for the time option.

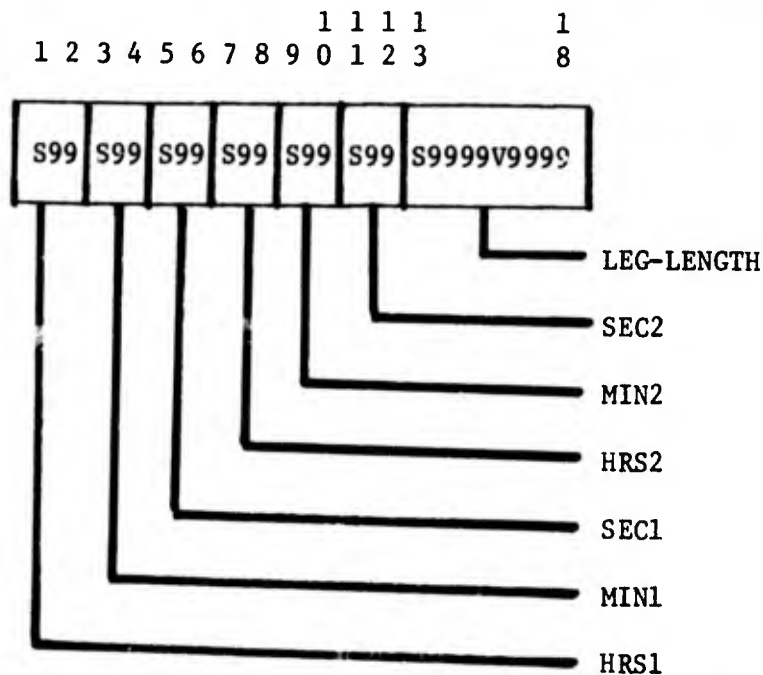
b. The input consists of the beginning and ending time for a leg and the length of the leg. The output is the beginning time as a decimal number in floating point format and the reciprocal of the speed along the leg.

2. Calling Sequence.

ENTRY POINT RTMES USING A-DATA  
GIVING B-DATA, A-FLAG.

where:

A-DATA contains the input data in the format shown below:



HRS1, MIN1, SEC1 are the hours, minutes, and seconds of the initial point of the leg.

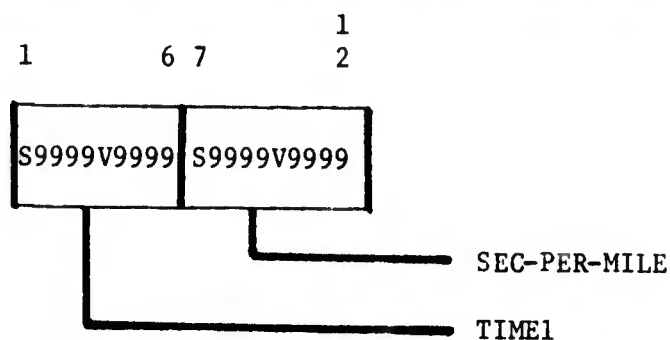
ENCLOSURE B

HRS2, MIN2, SEC2 are the hours, minutes and seconds at the terminal point of the leg.

These values are all in Display usage.

LEG-LENGTH is the length of the leg in nautical miles, floating point (computational-2) usage.

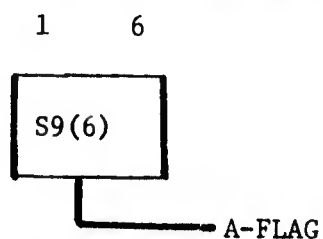
B-DATA contains the output data on the format shown below:



TIME1 is the initial time in seconds from midnight in floating point (computational-2) usage.

SEC-PER-MILE is the reciprocal of the speed (nautical miles per sec) in floating point (computational-2) usage.

A-FLAG contains a flag indicating whether the input data was valid or not. The format of A-FLAG is shown below.



A-FLAG is in binary (computational-1) usage, and has the values:

$$A-FLAG = \begin{cases} 1, & \text{successful execution} \\ 2, & \text{invalid data} \end{cases}$$

ENCLOSURE B



(b) Description. The hours, minutes, and seconds fields are validated by verifying that each is in its appropriate range of values.

$$0 \leq \text{hours} \leq 24$$

$$0 \leq \text{minutes} \leq 59$$

$$0 \leq \text{seconds} \leq 59$$

The times in seconds are then computed and checked to insure that TIME2 is greater than TIME1.

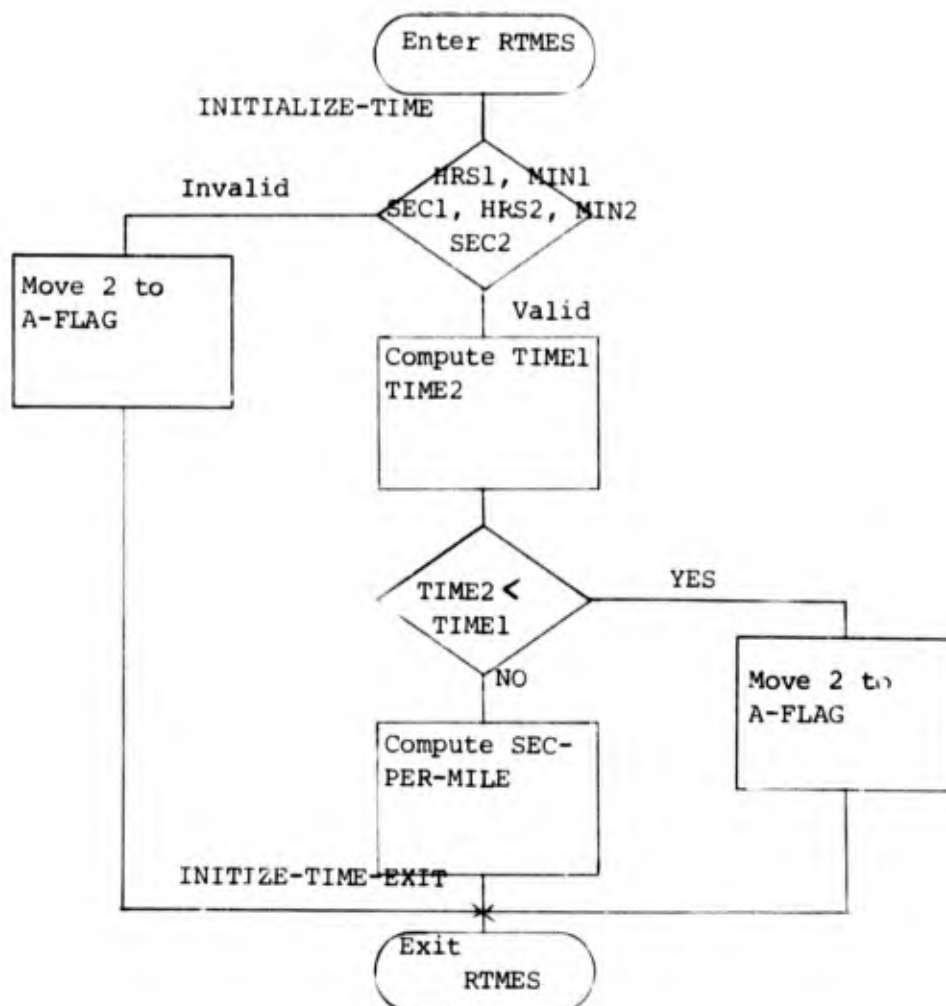
Finally, the reciprocal of the velocity is computed.

The equations used are:

$$\begin{aligned} \text{Time (in seconds)} &= 3600 \text{ (hours)} + 60 \text{ (minutes)} + \text{seconds} \\ \text{SEC-PER-MILE} &= \frac{\text{Total Time (seconds)}}{\text{length of leg (nautical miles)}} \\ &= (\text{TIME2} - \text{TIME1}) / \text{LEG-LENGTH} \end{aligned}$$

The time is kept in seconds since this keeps the value a whole number and avoids roundoff errors.

ENCLOSURE B



ENCLOSURE B

(7) CRDVAX

(a) Abstract.

1. Function.

a. This subroutine validates geographic coordinates input in either 11, 13 or 15 character format. It is used by the MIDMS HIS 600/6000 geographic search routines.

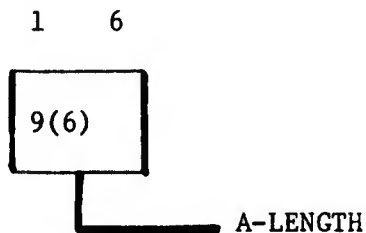
b. Given a length and an input set of coordinates CRDVAX returns a flag indicating the validity of the data.

2. Calling Sequence.

ENTRY CRDVAL USING A-LENGTH,  
A-DATA GIVING A-FLAG.

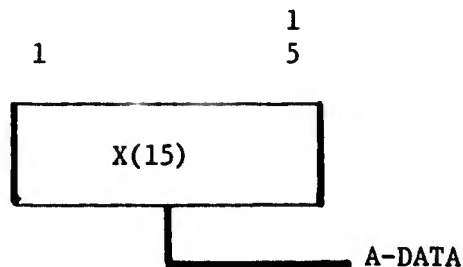
where:

A-LENGTH contains the number of valid characters, left-justified, in the A-DATA field. The format of A-LENGTH is shown below:



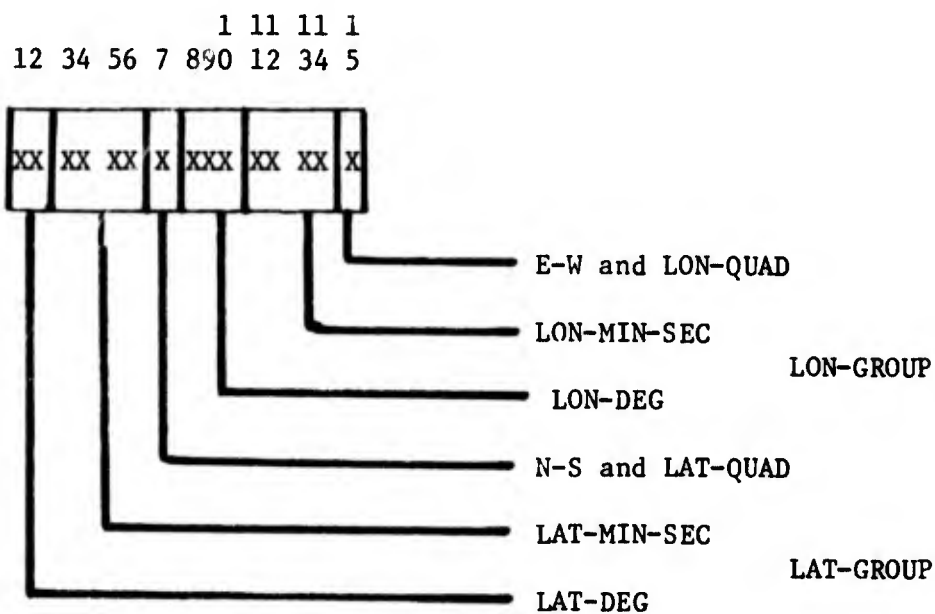
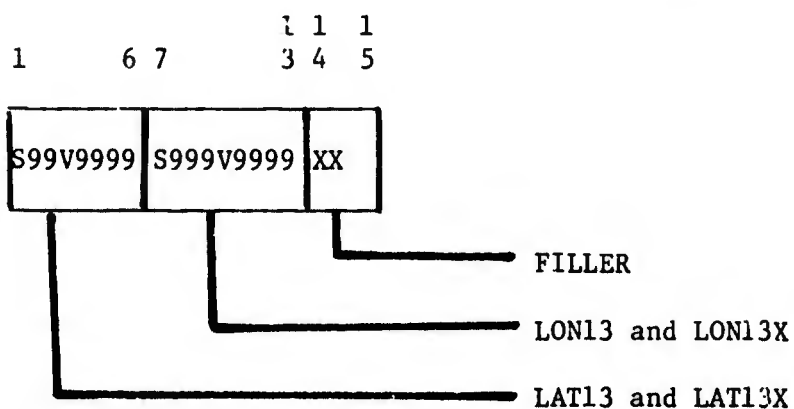
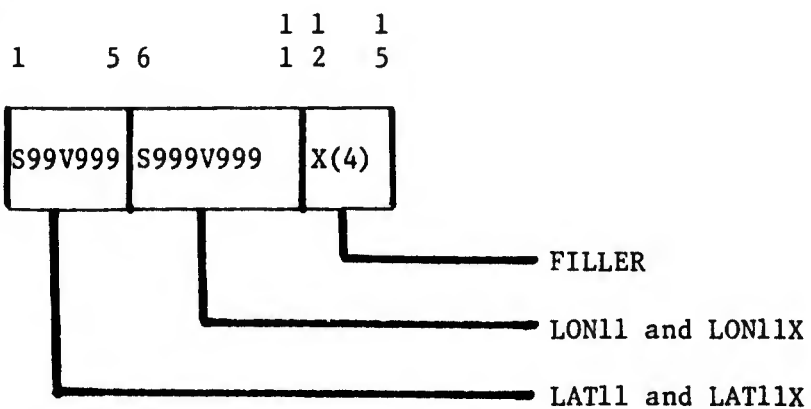
A-LENGTH is computational-1 (binary) usage.

A-DATA contains the geographic coordinates left-justified. The format of A-DATA is shown below:



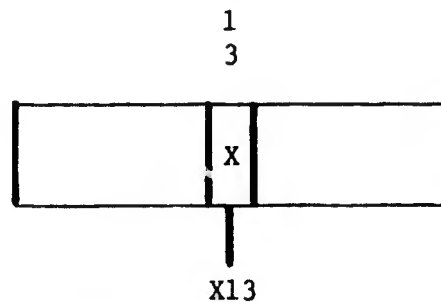
ENCLOSURE B

The A-DATA is redefined for 11, 13 and 15 character format as shown below:



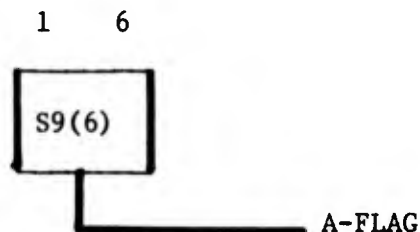
Each character in A-DATA may be individually addressed by Xi.

e.g.



The usage of all the A-DATA formats is display.

A-FLAG contains a flag indicating whether the input was valid or not. The format is shown below:



A-FLAG is computational-1 (binary) usage and has the possible values

- A-FLAG = 1, good data  
2, not numeric  
3, latitude exceeds 90°  
4, longitude exceeds 180°  
5, Deg min sec invalid  
6, latitude not North or South  
7, longitude not East or West  
8, incorrect length  
9, Degrees minutes seconds field in  
15-character format is not numeric.

(b) Description. The three cases of possible inputs are handled separately, but the logic is basically the same for 11 and 13 character cases.

ENCLOSURE B

The conversion of MIDMS files from IBM to HIS 600/6000 caused a problem in the geographic coordinates. Since the files are converted character by character without regard to the meaning of a field, signed numeric characters may be converted incorrectly.

The table below shows the IBM signed numeric characters and what they will be converted to on the HIS 600/6000.

IBM			HIS 600/6000		
<u>Value</u>	<u>Print</u>	<u>Her</u>	<u>Value</u>	<u>Print</u>	<u>Octal</u>
+0	0	C0	0	0	00
+1	A	C1	+17	A	21
+2	B	C2	+18	B	22
+3	C	C3	+19	C	23
+4	D	C4	+20	D	24
+5	E	C5	+21	E	25
+6	F	C6	+22	F	26
+7	G	C7	+23	G	27
+8	H	C8	+24	H	30
+9	I	C9	+25	I	31
-0	0	D0	0	↑	40
-1	J	D1	-1	J	41
-2	K	D2	-2	K	42
-3	L	D3	-3	L	43
-4	M	D4	-4	M	44
-5	N	D5	-5	N	45
-6	O	D6	-6	O	46
-7	P	D7	-7	P	47
-8	Q	D8	-8	Q	50
-9	R	D9	-9	R	51

As can be seen in the above the value of the positive numbers is changed. This may be corrected by turning off the number 1 bit of the character by using a GMAP "AND" operation as shown below:

$$\begin{array}{rcccccc}
 & b_0 & b_1 & b_2 & b_3 & b_4 & b_5 \\
 \text{AND} & 1 & 0 & 1 & 1 & 1 & 1 \\
 \hline
 & b_0 & 0 & b_2 & b_3 & b_4 & b_5
 \end{array}
 \quad \leftarrow \text{Mask}$$

This operation must be performed for both the 11 and 13 character inputs.

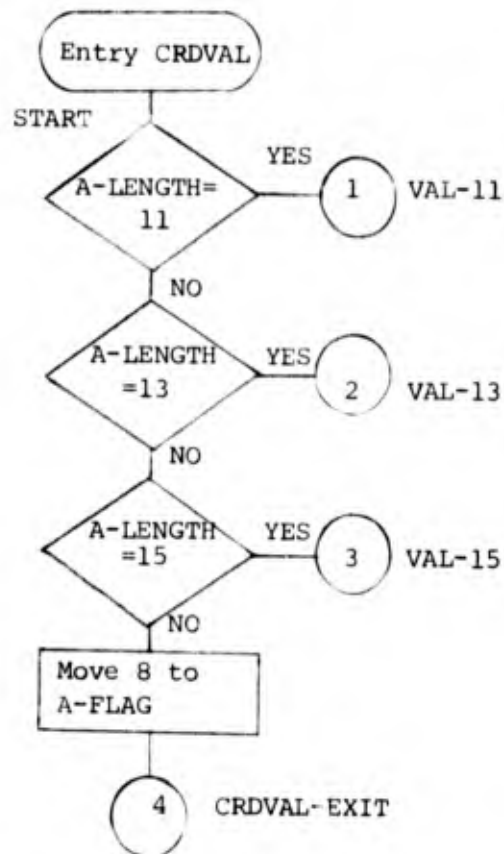
ENCLOSURE B

Since the HIS COBOL conditional test IF ... NUMERIC ... compares each character to the limits 0 to 9 (octal: 00 to 11) a negative digit would be considered not numeric. Thus, it was necessary to turn off bit zero ( $b_0$ ) as well as bit one ( $b_1$ ) before testing either the 11 or 13 character fields.

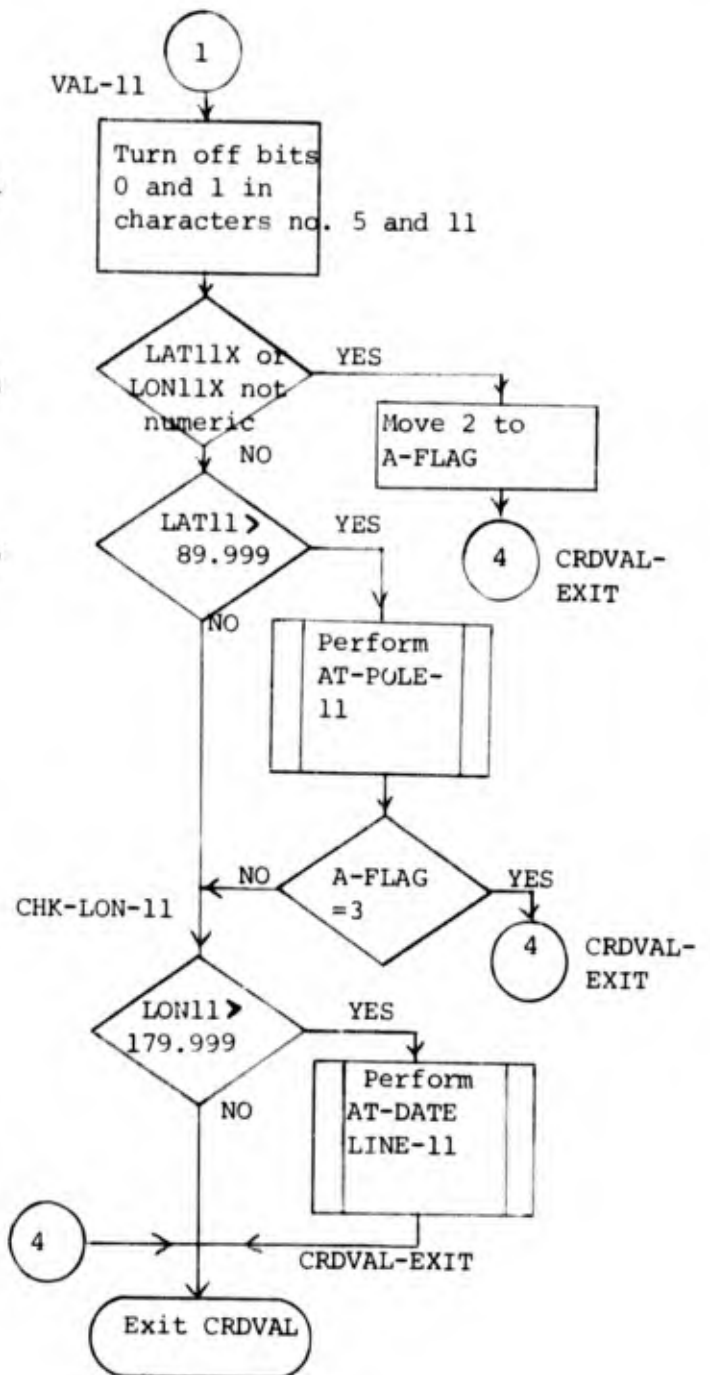
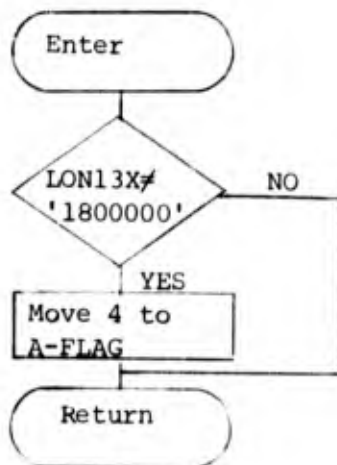
The following table presents the output value (A-FLAG) for various input.

Input Type	Input Data	A-FLAG
11 or 13	$0 \leq \text{Lat} \leq 90.000$ }	1
	$0 \leq \text{Lon} \leq 180.0000$ }	
	Lat or Lon not numeric	2
	Lat 90.0000	3
	Lon 180.0000	4
15	Lat = 900000 N (or S) }	1
	Lon = 1800000 E (or W) }	
	Lat or Lon not numeric	9
	Lat > 90°	3
	Lat minutes > 59 or sec > 59	5
	Lat Quadrant not "N" or "S"	6
	Lon > 180°	4
	Lon minutes or second > 59	5
	Lon Quadrant not "E" or "W"	7

ENCLOSURE B

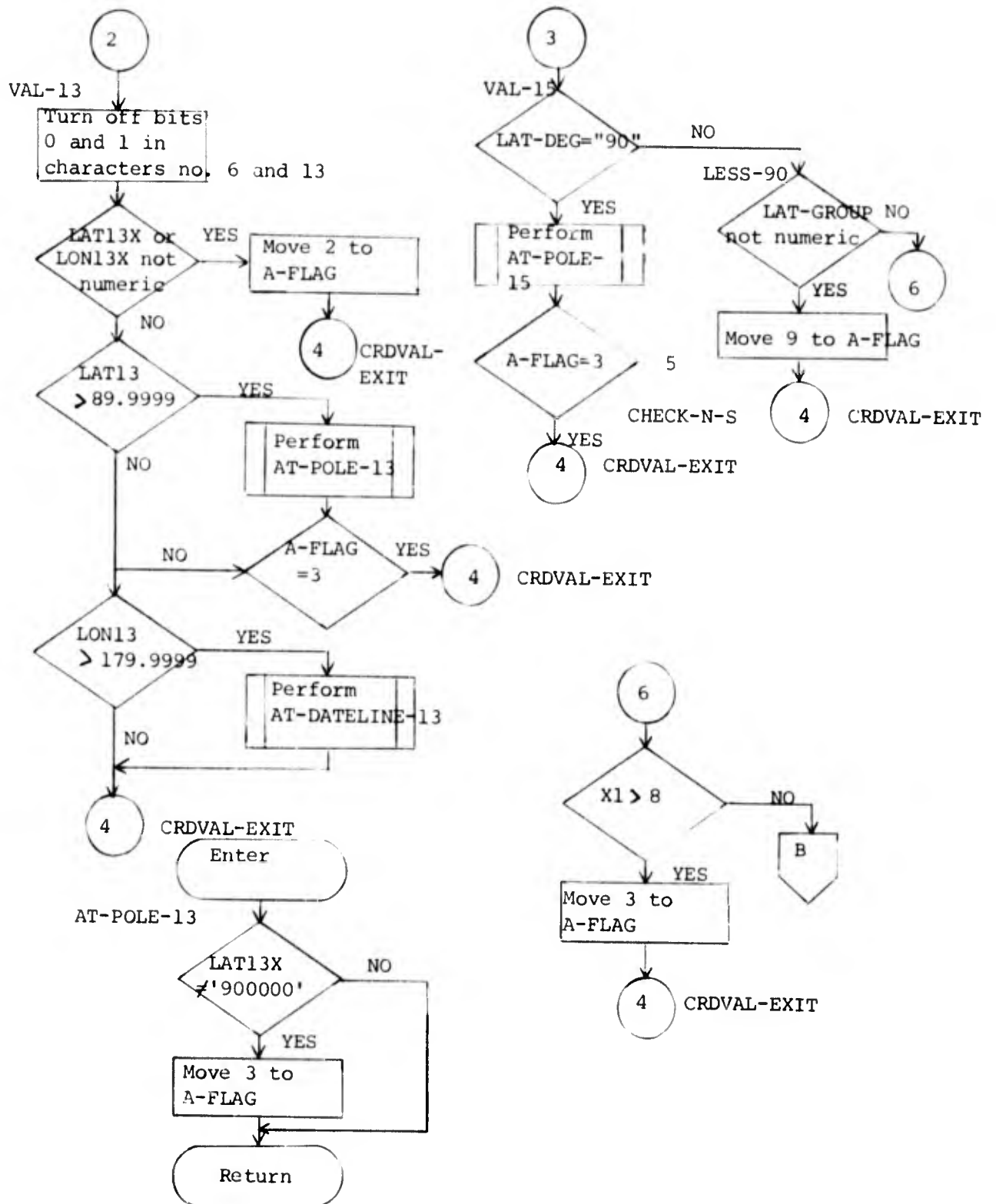


AT-DATELINE-13

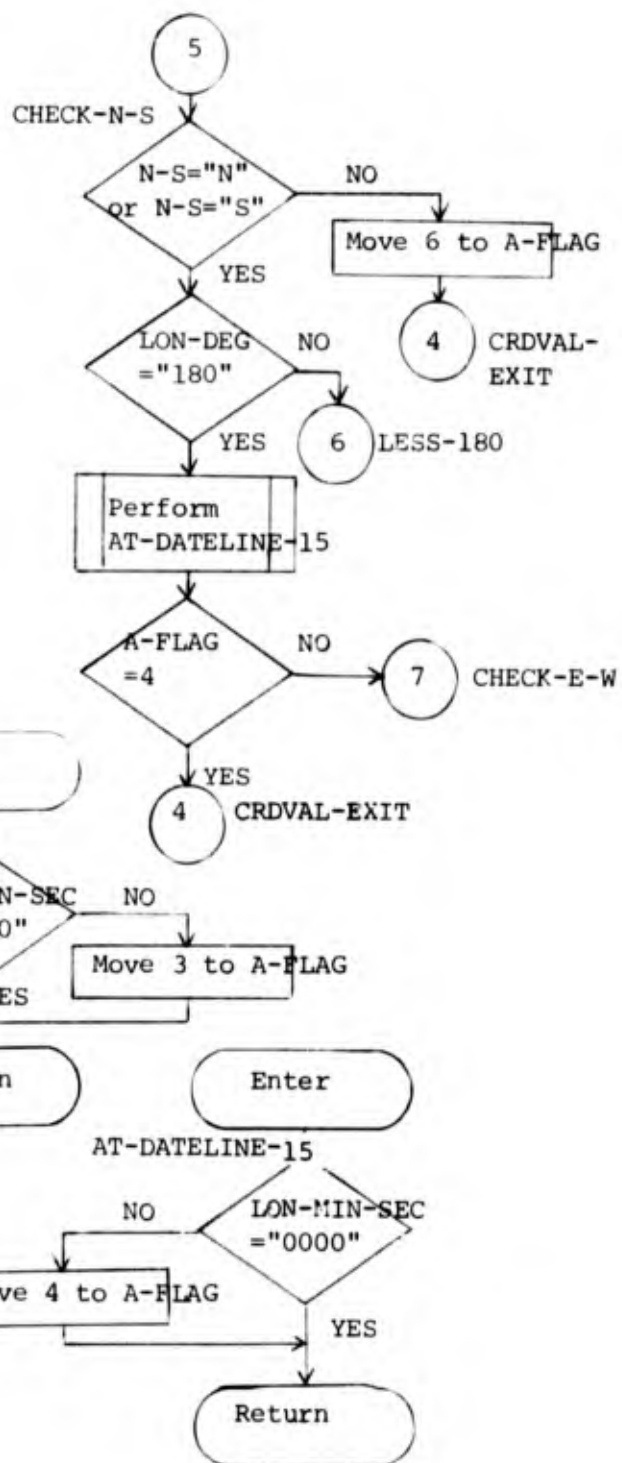
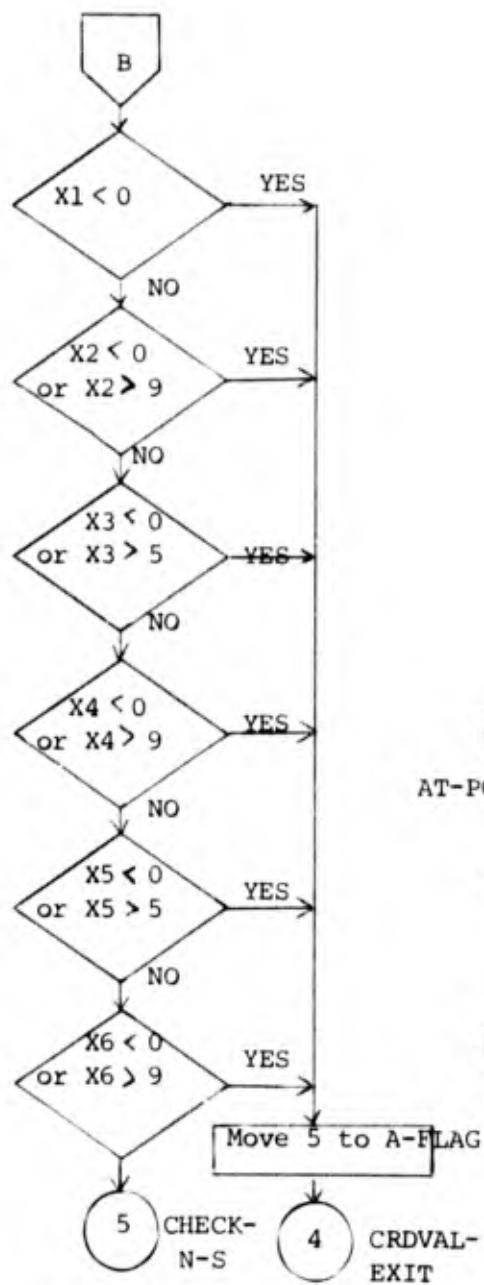


ENCLOSURE B

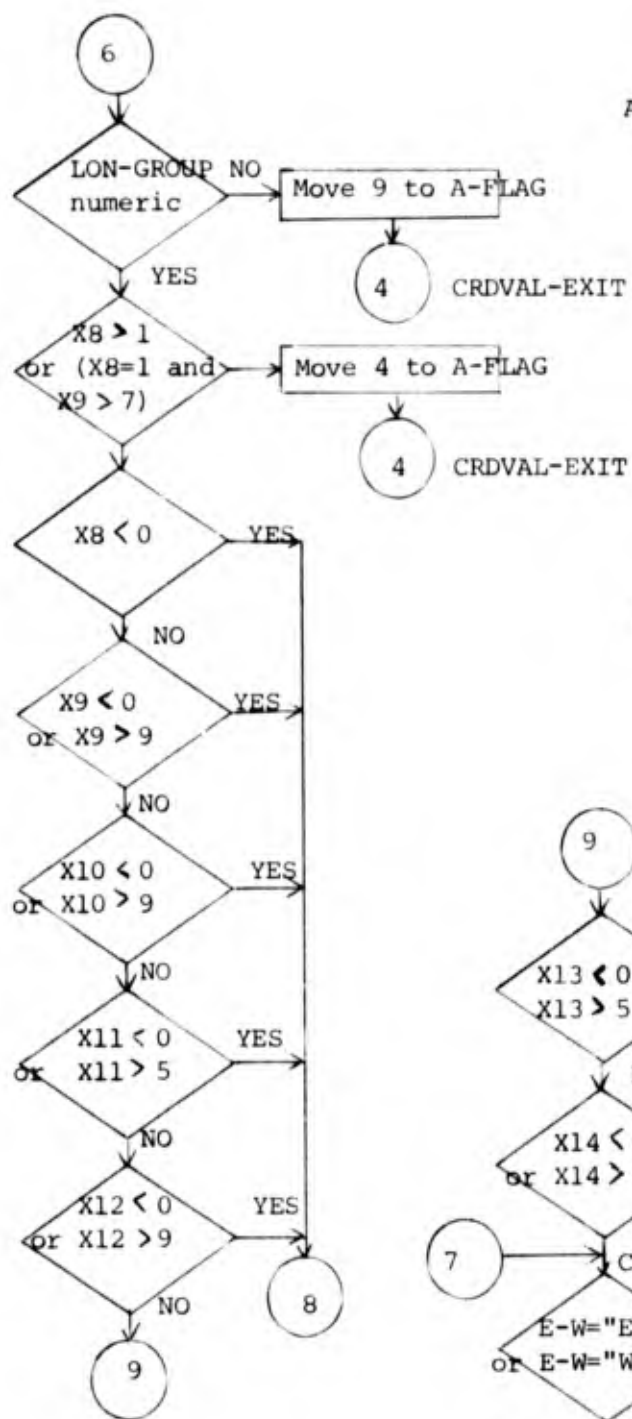




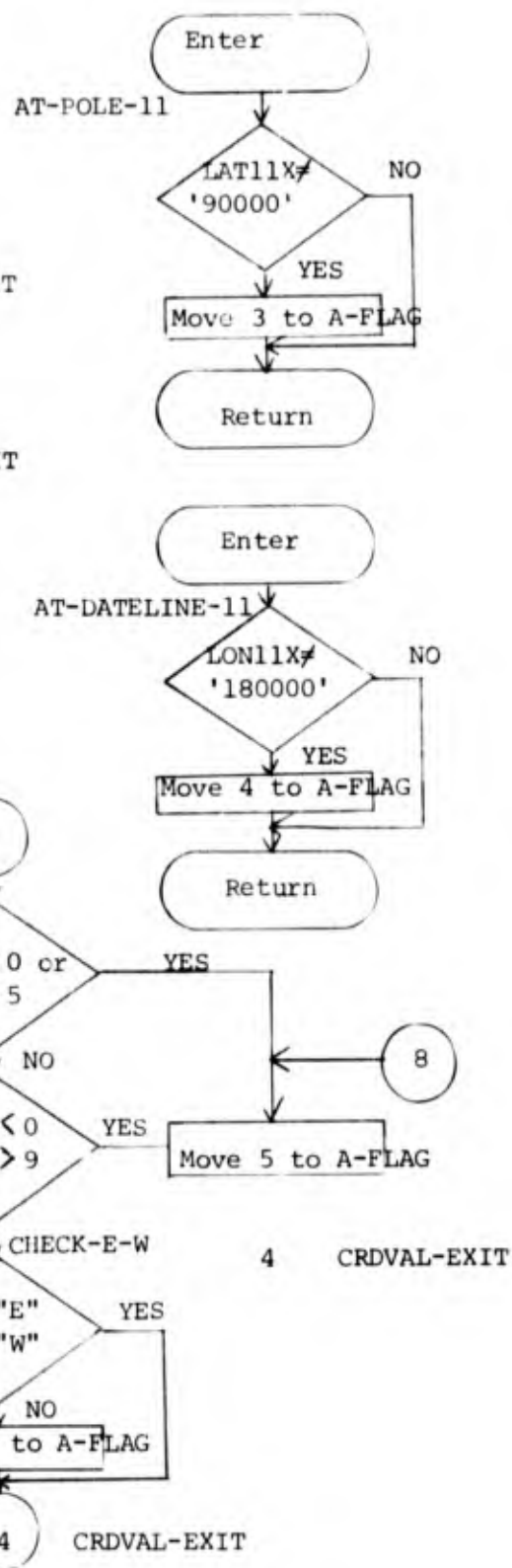
ENCLOSURE B



ENCLOSURE B



ENCLOSURE B



c. Error Messages.

(1) ERROR IN FORMAT SIZE.

(a) In-length is not the same value as required by the conversion program.

(b) Check for improperly imbedded blanks, missing commas, extra fields, and missing characters in the parameter field (field enclosed with quotations) of the query card.

(2) ERROR IN EVENT POINT COORDINATES.

(a) Coordinates are incorrect.

(b) Check longitudes for over  $180^\circ$ , latitudes for over  $90^\circ$ ; insure characters for direction are correct and in their appropriate positions; check for alphabetic characters in numeric part of the field.

(3) ERROR WIDTH SPECIFICATIONS.

(a) Widths (explicit) are incorrect.

(b) Check format to insure that "L" and "R" are used appropriately; that preceding digits are numeric and both widths are present.

(4) ERROR IN TIME DATA.

(a) Either the hours, minutes, or seconds fields of the beginning and ending times are invalid or ending time is less than beginning time.

(b) Check data to insure that hours  $\leq 24$ , minutes and seconds  $\leq 59$ .

(5) ERROR IN FRAME DATA.

(a) Data for the frame option are not valid.

(b) Check that frame internal and initial and terminal frame numbers are numeric, and that total number of frames is a multiple of the frame internal (i.e., terminal frame number - initial frame number + 1 = M x frame internal for some integer M).

ENCLOSURE B

(6) LENGTH OF LEG CLOSE TO ZERO.

(a) The leg is less than  $10^{-6}$  nm long, and is rejected.

(b) Check the coordinates for incorrect entries.

(7) ERROR IN ALTITUDE, FOCAL LENGTH, FOCAL PLANE.

(a) The altitude, focal length or focal plane entry is invalid.

(b) Check that altitude and focal length are numeric, that focal length  $\neq 0$ , and that focal plane code is either "A" or "B".

## Ancillary System Routines

### Section 1 - IBM

#### 1. ABGET

a. Function. To move data, whose location is specified as an absolute address, into a symbolically-named field.

b. Calling Sequence. In COBOL one may write:

```
CALL 'ABGET' USING ABSOLUTE-LOC  
RECEIVING-FLD  
FLD-LEN.
```

where:

(1) ABSOLUTE-LOC is a fullword binary field containing the absolute address of the leftmost character of the wanted data.

(2) RECEIVING-FLD names the field to receive the data.

(3) FLD-LEN is a halfword binary field containing the number of characters to move.

c. Limitation. FLD-LEN must fall within the range 1-256 (inclusive); any other value is effectively reduced modulo 256 (but if the remainder on division by 256 is 0, the effective value of FLD-LEN becomes 256).

```
MEMBER NAME  ABGET  
ABGET      START 0                      GET FM ABSOLUTE LOC  
* ROUTINE TO MAKE CONTENTS OF FIELD AVAILABLE ON THE COBOL-LEVEL  
* WHEN ONLY THE ABSOLUTE CORE-ADDRESS OF FIELD IS KNOWN. TO USE  
*      CALL 'ABGET' USING ABLOC, FLDNAME, FLDLEN  
* WHERE ABLOC IS A FULLWORD BINARY HOLDING THE HOP-ADDRESS TO BE  
* MOVED, FLDNAME IS THE COBOL DATANAME INTO WHICH CONTENTS OF FIELD  
* WILL BE MOVED, AND FLDLEN IS A HALFWORD BINARY CONTAINING THE  
* FIELD LENGTH -- N. B. MAX FLDLEN IS 256.  
      USING *,15  
      SAVE (2,5)  
      LM   2,4,0(1)  
      L    2,0(2)  
      LH   4,0(4)  
      BCTR 4,0  
      EX   4,MV  
      RETURN (2,5),T  
MV      MVC 0(0,3),0(2)  
      END
```

ENCLOSURE C

## 2. CALLIO

a. Function. To interface between File Maintenance routines and the FMIOX subroutine.

b. Calling Sequence. The subroutine is called by File Maintenance routines through one of three COBOL calling sequences:

```
CALL 'FMSET' USING FMIO.  
CALL 'FMCRD' USING FM-INPUT END-SW.  
CALL 'FMPRT' USING PRT-LINE CC.  
CALL 'FMPUN' USING CARD-IMAGE.
```

where:

(1) FMSET is the entry point to initialize the CALLIO subroutine by setting the entry address for the FMIOX subroutine.

(2) FMIO is a computational (Picture 9(5)) field containing the entry address of the FMIOX subroutine.

(3) FMCRD is the entry point causing CALLIO to branch to FMIO with the equivalent of the following calling sequence:

```
CALL 'FMIO' USING FMIO-SW FM-INPUT END-SW  
(FMIO-SW being set to 1).
```

(4) FM-INPUT is an area of 80 characters in File Maintenance into which FMIOX will read a card.

(5) END-SW is a one character switch in File Maintenance into which FMIOX will specify end of file, ordinary card, or type of FM control card.

(6) FMPRT is the entry point causing CALLIO to branch to FMIO with the equivalent of the following calling sequence:

```
CALL 'FMIO' USING FMIO-SW PRT-LINE CC  
(FMIO-SW being set to 2).
```

(7) PRT-LINE is an area of 132 characters in File Maintenance from which FMIOX will print a line.

(8) CC is a one character field in File Maintenance containing a code from which FMIOX will determine printer carriage control.

ENCLOSURE C

(9) FMPUN is used by Single File Logical Maintenance as the entry point causing CALLIO to branch to FMIO with the equivalent of the following calling sequence:

CALL 'FMIO' USING FMIO-SW CARD-IMAGE  
(FMIO-SW being set to 3).

(10) CARD-IMAGE is an 80-character area from which FMIOX will punch a card.

c. Reference. See program documentation for the FMIOX subroutine.

MEMBER	NAME	CALL IO
1	...	...
2	...	...
3	...	...
4	...	...
5	...	...
6	...	...
7	...	...
8	...	...
9	...	...
10	...	...
11	...	...
12	...	...
13	...	...
14	...	...
15	...	...
16	...	...
17	...	...
18	...	...
19	...	...
20	...	...
21	...	...
22	...	...
23	...	...
24	...	...
25	...	...
26	...	...
27	...	...
28	...	...
29	...	...
30	...	...
31	...	...
32	...	...
33	...	...
34	...	...
35	...	...
36	...	...
37	...	...
38	...	...
39	...	...
40	...	...
41	...	...
42	...	...
43	...	...
44	...	...
45	...	...
46	...	...
47	...	...
48	...	...
49	...	...
50	...	...
51	...	...
52	...	...
53	...	...
54	...	...
55	...	...
56	...	...
57	...	...
58	...	...
59	...	...
60	...	...
61	...	...
62	...	...
63	...	...
64	...	...
65	...	...
66	...	...
67	...	...
68	...	...
69	...	...
70	...	...
71	...	...
72	...	...
73	...	...
74	...	...
75	...	...
76	...	...
77	...	...
78	...	...
79	...	...
80	...	...
81	...	...
82	...	...
83	...	...
84	...	...
85	...	...
86	...	...
87	...	...
88	...	...
89	...	...
90	...	...
91	...	...
92	...	...
93	...	...
94	...	...
95	...	...
96	...	...
97	...	...
98	...	...
99	...	...
100	...	...

CALL ID START

```
* THE PURPOSE OF THIS SUBROUTINE IS TO BRANCH TO THE FMIO SUBROUTINE
* LOAD MODULE AFTER SETTING UP A SWITCH TELLING FMIO TO READ A CARD (1)
* OR PRINT A LINE (2) OR PUNCH A LINE (3).
```

NAME	ENTRY	FMCRD	CARD READ ENTRY
FMCRD	LA	0,1	SET RO TO 1
	B	20(15)	GO TO SAVE MACRO
FMPRT	ENTRY	FMPRT	PRINT ENTRY
	LA	0,2	SET RO TO 2
	B	12(15)	GO TO SAVE MACRO
FMPUN	ENTRY	FMPUN	PUNCH ENTRY
	LA	0,3	SET RO TO 3
	SAVE	(14,2)	SAVE REGISTERS, ALL ENTRY POINTS
	BALR	2,0	SET UP
	USING	*,2	BASE REGISTER
	STH	0, SWITCH	SAVE SWITCH VALUE
	ST	13, SAVE+4	SAVE OLD SAVE ADDR
	LA	13, SAVE	LOAD NEW SAVE ADDR
	MVC	LIST+4(8), 0(1)	MCVE OLD PARAMETER LIST
	LA	1, LIST	LOAD NEW PARM LIST ADDR
	L	15, =A(FMIC)	OBTAIN FMIO HOLD AREA ADDR
	L	15, 0(15)	LOAD FMIO ENTRY POINT ADDR
	BALR	14, 15	CALL FMIO
	L	13, SAVE+4	LOAD OLD SAVE ADDR
	RETURN	(14, 2), T	RETURN
SAVE	DS	18F	SAVE AREA
LIST	DC	A(SWITCH)	SWITCH ADDRESS
	DS	2F	OLD PARAMETER LIST
SWITCH	DS	H	FMIO SWITCH

ENTRY FMSET INITIALIZE ENTRY

- \* FMSET INITIALIZES FIELD FMIO TO THE ENTRY POINT OF LOAD MODULE FMIO
- \* PROVIDED IN THE INPUT PARAMETER LIST.

```

* PROVIDE IN THE ENTRY POINT
FMSET USING *,15 SET UP BASE REGISTER
L 1,0(1) LOAD ADDR OF FMIO ENTRY POINT ADDR
MVC FMIO,0(1) MOVE FMIO ENTRY POINT ADDR
BR 14 RETURN
FMIO DS F FMIO ENTRY POINT ADDR
END

```



### 3. COMABSY.

a. Function. To compare data whose location is specified as an absolute address with data in a symbolically named field.

b. Calling Sequence. In COBOL one may write:

```
CALL 'COMABSY' USING ABSOLUTE-LOC
                        COMPARE-FLD
                        FLD-LEN, RESULT.
```

where ABSOLUTE-LOC is a fullword binary field containing the absolute address of the leftmost character of a field to be compared with the contents of COMPARE-FLD. The number of characters to be compared is in the halfword binary FLD-LEN. At return time, RESULT (a halfword binary) contains 1, 2 or 3 according as the data addressed by ABSOLUTE-LOC is less, equal or greater than the contents of COMPARE-FLD.

c. Limitation. FLD-LEN must fall within the range 1-256 (inclusive); any other value is effectively reduced modulo 256 (but if the remainder on division by 256 is 0, the effective value of FLD-LEN becomes 256).

```
MEMBER NAME COMABSY
COMABSY START 0 COMPARE ABSOLUTE-SYMBOLIC
* ROUTINE TO COMPARE 2 FIELDS ON COBOL LEVEL WHEN ONE FIELD IS KNOWN
* TO COBOL-PGM ONLY IN TERMS OF ABSOLUTE CORE ADDRESS. TO USE
* CALL 'COMABSY' USING ABLOC, FLDNAME, FLDLEN, RESULT.
* WHERE ABLOC IS A FULLWORD BINARY HOLDING THE HOP-ADDRESS OF ONE
* COMPARAND, FLDNAME IS THE COBOL DATANAME OF THE OTHER COMPARAND,
* FLDLEN IS A HALFWORD BINARY CONTAINING THE LENGTH OF THE COMPARAND
* FIELDS, AND RESULT CONTAINS 1, 2 OR 3 AT RETURN TIME, ACCORDING AS
* THE FIRST COMPARAND IS LESS, EQUAL OR GREATER THAN THE SECOND.
* N. B. MAX FLDLEN IS 256.
      USING *,15
      SAVE (2,6)
      LM 2,5,0(1)
      L 2,0(2)
      LH 4,0(4)
      BCTR 4,0
      EX 4,CP
      BH R3
      BL R1
      MVC 0(2,5),=H'2'
      B RN
R1 MVC 0(2,5),=H'1'
      B RN
R3 MVC 0(2,5),=H'3'
RN RETURN (2,6),T
CP CLC 0(0,2),0(3)
      END
```

ENCLOSURE C

4. COMALL.

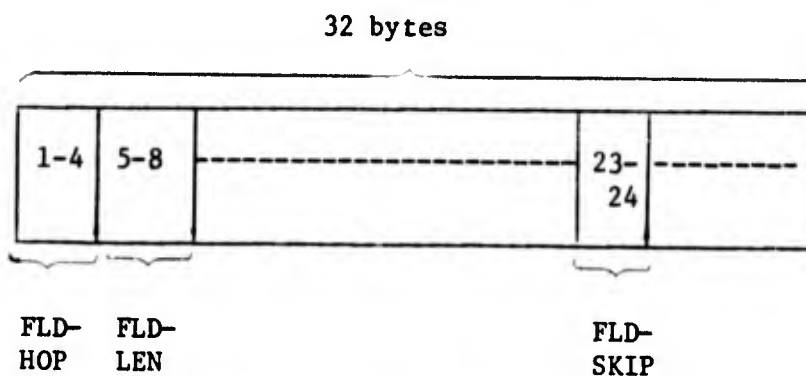
a. Function. To determine whether a source input field contains data or blanks, and set an appropriate switch in FLD-LIST.

b. Calling Sequence. This routine is called by FMIP as follows:

CALL 'COMALL' USING INPUT-RECORD,  
FLD-LIST,  
FIELD-LIMIT.

where FLD-LIST is a list, provided to FMIP. The number of entries in it is given by the binary fullword FIELD-LIMIT. Each field in the source data record (INPUT-RECORD) is described by an entry of FLD-LIST.

c. Capabilities. COMALL scans over the entries of FLD-LIST. The layout of each entry is as follows:



FLD-HOP locates a field in INPUT-RECORD relative to 1 (i.e., a field starting with the first bytes of INPUT-RECORD will have FLD-HOP=1, not 0). The length of the field is given in FLD-LEN. If the field, so defined, contains all blanks, then FLD-SKIP is set negative; otherwise, it is set positive. FLD-HOP and FLD-LEN are binary fullwords; FLD-SKIP is a binary halfword.

ENCLOSURE C

MEMBER NAME COMALL

CCMALL START 0

\* THE PURPOSE OF THIS ROUTINE IS TO SCAN THE INPUT AREA AND SET  
\* FLD-SKIP (I) FIELD TO NEGATIVE OR POSITIVE DEPENDING CN THE PRE  
\* OR ABSENCE (-) OF DATA.

\* THE CALLING SEQUENCE IS

\* CALL 'COMALL' USING INPUT-RECORD

\* FLD-LIST

\* \* FLD-HOP PICTURE 9(5) COMP 04

\* \* FLD-LEN PICTURE 9(5) COMP 08

\* FLD-PSS-HOP PICTURE 9(5) COMP 12

\* FLD-PSS-LEN PICTURE 9(5) COMP 16

\* FLD-LR2-POINTER PICTURE 999 COMP 18

\* FLD-VAL-CT PICTURE 999 COMP 20

\* FLD-VAL-POINT PICTURE 9999 COMP 22

\* \*FLD-SKIP PICTURE 9 COMP 24

\* FLD-NAME X 30

\* FIELD-LIMIT

\* NOTE THE FIELDS MARKED WITH \* ARE THE ONES USED.

USING \*,15

SAVE (2,7)

LM 2,4,0(1) INPLT-RECORD

FLD-ITEM FLD-LIMIT

\* XR2

XR3 XR4

\*

\*

\*

\*

LOOP LH 4,0(4) FLC-LIMIT

LH 7,22(3) FLD-SKIP

L 5,0(3) FLD HCP

AR 5,2 INPLT-RECORD + FLD-HOP

BCTR 5,0 ADJUST ADDRESS

L 6,4(3) FLD-LEN IN XR6

CLI 0(5),C' ' FIRST CHAR EQUAL BLANK

BNE POS

BCT 6,CONT GO TO CCNT IF 6 NOT EQUAL ZERO

B NEG IS ZERO

CONT BCTR 6,C

EX 6,COMPARE

BE NEG

POS LPR 7,7

STH 7,22(3)

BCT 4,MORE

B RETURN

COMPARE CLC 0(0,5),1(5)

MORE LA 3,32(0,3)

B LOOP

NEG LNR 7,7

STH 7,22(3)

BCT 4,MORE

RETURN RETURN (2,7),T

END

ENCLOSURE C

C-1-6

## 5. COMARAYS

a. Function. To compare two character-strings of equal length, each starting at a distinct position within a named field. The string-length and starting position are determined at execution time.

b. Calling Sequence. The subroutine can be called from a COBOL routine as follows:

```
CALL 'COMARAYS' USING ARRAYA INDEXA  
                     ARRAYB INDEXB LENGTHX INDICATOR
```

where:

(1) ARRAYA is the name of an array. It is meant to be the known or constant character stream (from input).

(2) INDEXA is a computational (fullword) field representing the character position within ARRAYA.

(3) ARRAYB is the name of an array to compare ARRAYA to.

(4) INDEXB is a computational (fullword) field representing the character position within ARRAYB.

(5) LENGTHX is the length of the fields to be compared (fullword computational).

(6) INDICATOR is a computational (halfword) area to store an indicator of the result where:

```
1:ARRAYA string collates lower than ARRAYB string.  
2:ARRAYA string collates equal to ARRAYB string.  
3:ARRAYA string collates higher than ARRAYB string.
```

For example, suppose ARRAYA, ARRAYB have the contents:

```
ARRAYA:31415926534  
ARRAYB:QWERTY15927
```

It is desired to compare 5 characters of ARRAYA and ARRAYB. The field in ARRAYA starts at position 4; the field in ARRAYB starts in position 7. One may then code:

ENCLOSURE C

```

MOVE 4 TO INDEXA.
MOVE 7 TO INDEXB.
MOVE 5 TO LENGTHX.
CALL 'COMARAYS' USING ARRAYA
INDEXA ARRAYB INDEXB LENGTHX
INDICATOR.

```

This will cause the string "15926" in ARRAYA to be compared with "15927" in ARRAYB, so that ARRAYA will test low and COMARAYS will return with INDICATOR=1.

c. Limitations. In the system-360 implementation, LENGTHX is limited to 256, and any larger value will be effectively reduced to the remainder on division by 256 (or to 256 if the remainder is zero).

```

MEMBER NAME COMARAYS
COMARAYS START 0
* THE CALLING SEQUENCE OF COMARAYS IS
*      CALL 'COMARAYS' USING ARRAYA      INDEXA
*                                ARRAYB    INDEXB
*                                LENGTHX   INDICATOR
* INDICATOR IS A HALFWORD.
      USING *,15
      SAVE (2,7)
      LM 2,7,0(1)
      L 6,0(6)
      BCTR 6,C
      A 2,0(3)
      BCTR 2,0
      A 4,0(5)
      BCTR 4,C
      EX 6,COMEX
      BL MOVE1
      BH MOVE3
      MVC C(2,7),=H'2'
      B COMRET
MOVE1 MVC 0(2,7),=H'1'
      B COMRET
MOVE3 MVC 0(2,7),=H'3'
      B COMRET
COMEX CLC 0(0,2),0(4)
CCMRET RETURN (2,7),T
      END

```

ENCLOSURE C

## 6. COMLIST

a. Function. To compare the contents of a field with the entries of a list.

b. Calling Sequence. To call from a COBOL program, code as follows:

```
CALL 'COMLIST' USING TEST-DATA XA
                        FIELD-LIST XB
                        FLD-CNT FLD-LEN
                        INDICATOR.
```

where:

(1) TEST-DATA and FIELD-LIST are character-strings. All other parameters are fullword binary fields except INDICATOR, which is a halfword binary field.

(2) The field whose HOP is at position XA of TEST-DATA is compared to the array of fields which start at position XB of FIELD-LIST. The length of each field is given by FLD-LEN and the number of entries in the list by FLD-CNT. If an entry of the array is found to match with the TEST-DATA field, the subroutine returns with INDICATOR=2; if not, INDICATOR=0.

(3) For example, suppose that TEST-DATA contains an 80-character card image. Starting in position 8 of the card is a 3-character field, and we need to know whether the field contains "GEN," "DEL," "UPD," "REP," or none of these. We may code:

```
MOVE 8 TO XA.
MOVE 1 TO XB.
MOVE 3 TO FLD-LEN.
MOVE 4 TO FLD-CNT.
MOVE 'GENDELUPDREP' TO FIELD-LIST.
CALL 'COMLIST' USING TEST-DATA XA
                        FIELD-LIST XB
                        FLD-CNT FLD-LEN
                        INDICATOR.
```

If the field contains one of the words "GEN," "DEL," ..., INDICATOR will contain 2; if none of these, INDICATOR=0.

```

MEMBER NAME CCMLIST
CCMLIST START 0
        USING *,15
*       THE PURPOSE OF THIS ROUTINE IS TO COMPARE A LIST OF FIELD
*       VALUES
*       THE FORMAT OF THE VAL-LIT-AREA IS 7-NNNN-LIST1--LIST2.....
*       WHERE LIST1,LIST2,...ARE THE ITEM
*       NNNN    NUMBER OF ELEMENTS
*       THE CALLING SEQUENCE IS
*       CALL 'CCMLIST' USING
*       INPUT-RECORD  INDEXA
*       VAL-LIT-AREA  INDEXB
*       NO-OF-ITEMS   LENGTH
*       INDICATOR .
        SAVE (2,8)
        LM      2,8,0(1)      GET ADDRESSES INPUT-RECORD INDEXA
*       VAL-LIT-AREA
*
*                                     XR 2      XR 3      XR 4
*                                     INDEXB NO-OF-ITEMS LENGTH
*                                     XR 5      XR 6      XR 7
*                                     INDICATOR
*                                     XR 8
*
        L      3,0(3)
        L      5,0(5)
        AR      2,3      INPUT-RECORD - INDEXA
        AR      4,5      VAL-LIT-AREA - INDEXB
        L      6,0(6)
        L      7,0(7)
        BCTR    2,0
        BCTR    4,0
        BCTR    7,0
EXECUTE EX      7,COMPARE      CCMPARE RESULT
        BE      HIT      FOUND
        BCT     6,MORE      RECYCLE
        STH     6,C(8)      NOT FOUND ZERO STORED
        B       RETURN
MORE     LA      4,1(4,7)      SUBTRACT LENGTH OF ELEMENT + 1
        B       EXECUTE
CCMPARE CLC     0(C,2),0(4)      THE CCMPARE
HIT      MVC    0(2,8),=H'2'
RETURN   RETURN (2,8),T
        END

```

## 7. COMNUMS

a. Function. To compare two numbers. The numbers may be signed or unsigned and 1 to 18 digits long.

b. Calling Sequence. The calling sequence is :

CALL 'COMNUMS' USING FIELDA, INDEXA, FIELDDB, INDEXB,  
LENGTHX, INDICATOR.

where:

(1) FIELDA is the location of the field or array from which the number is to be compared with FIELDDB.

(2) INDEXA is the subscript of the high order character of FIELDA. INDEXA is a fullword computational.

(3) INDEXB is the same as INDEXA except the subscript of the high order character of the FIELDDB.

(4) LENGTHX is the length of the fields to compare. LENGTHX is a fullword computational.

(5) INDICATOR is a halfword computational with the following returning values:

- 1 = FIELDA IS LOW
- 2 = FIELDS ARE EQUAL
- 3 = FIELDA IS HIGH
- 4 = DATA ERROR

MEMBER NAME	COMNUMS
COMNUMS	START 0
*	THE CALLING SEQUENCE IS
*	CALL 'COMNUMS' USING
*	ARRAYA INDEXA ARRAYB INDEXB
*	LENGTHX INDICATOR.
*	ALL ARGUMENTS ARE FULLWORD EXCEPT INDICATOR.
*	INDICATOR IS HALFWORD.
	USING *,15
	SAVE (2,9)
LM	2,7,0(1)
L	6,0(6) LENGTH
PCTR	6,0 LENGTH -1.
A	2,0(3) ARRAYA + INDEXA



	RCTR	2,0	-1
	A	4,0(5)	
	RCTR	4,0	
	LR	8,2	INPUTA + INDEX
	AR	8,6	LENGTH ADDED
	LR	9,4	INPUTB + INDEX
	AR	9,6	LENGTH ADDED
	CLI	0(8),X'F0'	FIELDA F0 THRU F9
	RL	LOWB	
	CLI	0(8),X'FA'	
	BNL	ERROR	
GOOD1	CLI	0(9),X'F0'	FIELDB F0 THRU F9
	RL	GOOD2	
	CLI	0(9),X'FA'	
	RL	GOOD	
	B	ERROR	
LOWB	CLI	0(8),X'CO'	FIELDB C0 THRU C9
	RL	ERROR	
	CLI	0(8),X'CA'	
	RL	GOOD1	
	CLI	0(8),X'D0'	FIELDB D0 THRU D9
	RL	ERROR	
	CLI	0(8),X'DA'	
	RL	GOOD1	
	B	ERROR	
GOOD2	CLI	0(9),X'CO'	FIELDB C0 THRU C9
	RL	ERROR	
	CLI	0(9),X'CA'	
	RL	GOOD	
GOOD3	CLI	0(9),X'D0'	FIELDB D0 THRU D9
	RL	ERROR	
	CLI	0(9),X'DA'	
	RL	GOOD	
ERROR	MVC	0(2,7),=H'4'	
	B	COMRET	
GOOD	EX	6,PACK1	
	EX	6,PACK2	
	CP	FIELDA,FIELDB	
	RL	MOVE1	
	BH	MOVE3	
	MVC	0(2,7),=H'2'	
COMRET	RETURN	(2,9),T	
MOVE1	MVC	0(2,7),=H'1'	
	B	COMRET	
MOVE3	MVC	0(2,7),=H'3'	
	B	COMRET	
PACK1	PACK	FIELDA,0(0,2)	
PACK2	PACK	FIELDB,0(0,4)	
FIELDA	DS	CL10	
FIELDB	DS	CL10	
	END		

ENCLOSURE C

## 8. COMREC

a. Function. To enable a COBOL programmer to test a character string of an arbitrary starting point and of a variable length for blank status.

b. Calling Sequence. The COBOL calling sequence is as follows:

```
CALL 'COMREC' USING INPUT-RECORD
                    INPUT-FLD-HOP
                    INPUT-FLD-LENGTH
                    HIT-TABLE-1.
```

where:

(1) INPUT-RECORD is the name of an area (data record) in which the character string (data field) may be found.

(2) INPUT-FLD-HOP is the arbitrary starting point of the character string (high order position of the data field) which is used as an index to locate a particular data field in the data record.

(3) INPUT-FLD-LENGTH defines the length of the character string (data field).

(4) HIT-TABLE-1 is a halfword computational field. At return time, it will hold a zero if the field is all blanks; otherwise, it will hold a one.

MEMBER NAME	COMREC
COMREC	START 0
* THE CALLING SEQUENCE OF COMREC IS	
* CALL 'COMREC' USING INPUT-RECORD INPUT-FLD-HOP	
* INPUT-FLD-LENGTH HIT-TABLE-1	
* HIT-TABLE-1 IS A HALFWORD.	
	USING *,15
SAVE	(2,5)
LM	2,5,0(1)
L	3,0(3)
AR	2,3
BCTR	2,0
L	4,0(4)
CLI	0(2),C' '
BNE	MOVE1
BCT	4,*+8
B	MOVE0

ENCLOSURE C

	BCTR	4,0
	EX	4,COMPAR
	BE	MOVED
MOVE 1	MVC	0(2,5),=H'1'
	B	LEAVE
MOVED	MVC	0(2,5),=H'0'
LEAVE	RETURN	(2,5),T
COMPAR	CLC	0(0,2),1(2)
	END	

## 9. DATESUB

a. Function. To take the five character Julian date from the 360 Operating System.

b. Calling Sequence. The subroutine can be called from a COBOL routine as follows:

CALL 'DATESUB' USING DATE-FLD.

where DATE-FLD is a five digit numeric field.

DATESUB START

\* THE FUNCTION OF THIS SUBROUTINE IS TO READ THE OS SYSTEM DATE  
 \* THE FORMAT OF THE DATE IS FIVE CHARACTER JULIAN  
 \*

	SAVE	(14,12)
	BALR	10,0
	USING	*,10
	ST	13,SAVE+4
	LR	9,13
	LA	13,SAVE
	ST	13,8(9)
	L	5,0(1)
	TIME	DEC
	ST	1,LIST
	UNPK	0(5,5),LIST+1(3)
	LR	13,9
	RETURN	(14,12),T
SAVE	DS	18F
LIST	DS	F
	END	

## 10. EXPNSP

a. Function. To expand every MIDMS Automated Installation Intelligence File (MIDMS AIF) record from compressed form into fixed form.

ENCLOSURE C

b. Calling Sequence. The subroutine can be called from a COBOL program as follows:

CALL 'EXPNSP' USING RXF RX.

where:

(1) RXF is the high order position address of the compressed record.

(2) RX is the high order position address of the expanded record.

c. Capabilities. The subroutine initializes the RX output area to 6856 blank characters. The fixed portion of the record is moved from RXF to RX. The expansion control words for the compressed portion of the record are validated, decoded, and the compressed data are moved to the appropriate locations in the expanded record area RX.

d. Limitations.

(1) This subroutine is used only by the AIF PROCESSOR. The user should not attempt to call this subroutine from either RT or OP since a call is issued automatically by the AIF PROCESSOR.

(2) An invalid expansion control word in the compressed record causes expansion of the record to cease. Normal return is made to the invoking program. No warning message is issued by EXPNSP.

## 11. LINK

a. Function. To provide dynamic loading and execution of MIDMS system load modules, user subroutines, and logic packages.

b. Calling Sequence. The LINK subroutine can be called from a COBOL routine as follows:

CALL 'LINK' USING NAME Subroutine-Parameters.

where:

(1) NAME is the load module member or alias name (usually the COBOL PROGRAM-ID or ENTRY name).

ENCLOSURE C

(2) Subroutine-Parameters are the parameters being passed to the subroutine being linked (zero or more parameters are permitted).

```
MEMBER NAME LINK
LINK      START 0
          SAVE (14,12)
          BALR 10,C
          USING *,10
          LR   12,13
          LA   13,SAVE
          ST   13,8(12)
          ST   12,4(13)
          L    2,0(1)
          LA   1,4(1)
          LINK EPLOC=(2)
          LR   13,12
          RETURN (14,12),T
SAVE      DS    18F
          END
```

## 12. LMLOOK

a. Function. This routine is called via entry-point LMLKUP when a table lookup is executed in Logical Maintenance or Retrieval. It loads the table into core, if not already loaded, and carries out a binary search for a table-argument matching the argument in the calling-sequence.

b. Calling Sequence. LMLOOK uses three subroutines written in machine-code (ALC for 360):

(1) LOADTAB executes the LOAD macro of the operating system.

(2) ABGET obtains contents of a field, given the absolute address.

(3) COMABSY compares contents of two fields, of which one is available as a COBOL dataname, while the other is known only in terms of absolute location.

c. Capabilities.

(1) Upon entry, the program scans the list of tables loaded (paragraph TAD-SCAN). If the table is not yet loaded, its name is added to the list, and control goes to LODIT, where the new table is loaded (CALL 'LOADTAB'), and the field TRIPLET (argument-length, function-length, item-count) picked up from the table (CALL 'ABGET').

ENCLOSURE C

If the table has already been loaded, control goes to FINDIT, where TRIPLET is picked up from the list.

```
MEMBER NAME  LML00K
C01000 IDENTIFICATION DIVISION.
C02000 PROGRAM-ID. 'LML00K'.
C03000 ENVIRONMENT DIVISION.
C04000 CONFIGURATION SECTION.
C05000 SOURCE-COMPUTER.
C06000      IBM-360.
C07000 OBJECT-COMPUTER.
C08000      IBM-360.
C09000 INPUT-OUTPUT SECTION.
C10000 DATA DIVISION.
C11000 FILE SECTION.
C12000 WORKING-STORAGE SECTION.
C13000 77 TANDEX      PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED.
C14000*      RUNNING INDEX OVER TAD-ENTRIES.
C15000 77 TADCNT      PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED
C16000      VALUE 0.
C17000*      COUNT OF ITEMS CURRENTLY IN TAD.
C18000*****      MUST RESIDE IN FM SUPERVISOR *****
C19000 77 TADMAX      PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED
C20000      VALUE 5.
C21000*      MAX ALLOWED ENTRIES IN TAD.
C22000 77 LOADAD      PICTURE 9(08) COMPUTATIONAL SYNCHRONIZED.
C23000*      CORE-ADDRESS WHERE TABLE NAMED 'TABNAME' STARTS.
C24000 77 HWB-6      PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED
C25000      VALUE 6.
C26000*      HALFWORD BINARY 6.
C27000 77 LCMARK      PICTURE 9(08) COMPUTATIONAL SYNCHRONIZED.
C28000 77 HIMARK      PICTURE 9(08) COMPUTATIONAL SYNCHRONIZED.
C29000 77 MIDMARK      PICTURE 9(08) COMPUTATIONAL SYNCHRONIZED.
C30000*      HIGH, LOW, AND SPLIT-THE DIFFERENCE CORE-ADDRESSES
C31000* OF TABLE ENTRIES FOR BINARY SEARCH.
C32000 77 LODX      PICTURE 9(04) COMPUTATIONAL SYNCHRONIZED.
C33000 77 HIDX      PICTURE 9(04) COMPUTATIONAL SYNCHRONIZED.
C34000 77 MIDX      PICTURE 9(04) COMPUTATIONAL SYNCHRONIZED.
```

ENCLOSURE C

MEMBER NAME LMLOCK  
 035000\* HIGH, LOW, AND SPLIT-THE-DIFFERENCE SEQUENTIAL POSITIONS  
 036000\* OF TABLE ENTRIES FOR BINARY SEARCH.  
 037000 77 HILODEX PICTURE 9(04) COMPUTATIONAL SYNCHRONIZED.  
 038000\* TO HOLD HLODEX + LODEX.  
 039000 77 ITMLEN PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED.  
 040000\* = ARGLEN & FUNLEN = LENGTH OF A TABLE-ENTRY.  
 041000 77 ITMHAF PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED.  
 042000\* = ITMLEN/2, DISCARDING REMAINDER.  
 043000 77 RESULT PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED.  
 044000\* I E RESULT OF COMPARE.  
 045000 77 REM PICTURE 9(02) COMPUTATIONAL SYNCHRONIZED.  
 046000\* REMAINDER ON DIVISION BY 2.  
 047000 01 TAD.  
 048000\* T A D -- TABLE-DIRECTORY.  
 049000 02 TADENTRY OCCURS 5 TIMES.  
 050000 03 TADNAME PICTURE X(8).  
 051000\* NAME OF A TABLE (5-CHAR MNEMONIC)  
 052000 03 LODAX PIC 9(7) COMP.  
 053000\* CORE-ADDRESS WHERE TABLE NAMED 'TADNAME' STARTS.  
 054000 03 TRIPLEX.  
 055000\* 3 ITEMS OCCUPY FIRST 6 BYTES OF TABLE.  
 056000 04 ARGLENX PIC 99 COMP.  
 057000\* LENGTH OF EACH ARGUMENT IN THIS TABLE.  
 058000 04 FUNLENX PIC 99 COMP.  
 059000\* LENGTH OF EACH FUNCTION IN THIS TABLE.  
 060000 04 ITMCNTX PIC 9999 COMP.  
 061000\* COUNT OF ITEMS (ARG-FUNC PAIRS) IN THIS TABLE.  
 062000 01 TRIPLET.  
 063000\* SIMILAR TO 'TRIPLEX' BUT NOT SUBSCRIPTED.  
 064000 02 ARGLEN PIC 99 COMP.  
 065000\* ARGUMENT LENGTH, NOT SUBSCRIPTED.  
 066000 02 FUNLEN PIC 99 COMP.  
 067000\* FUNCTION LENGTH, NOT SUBSCRIPTED.  
 068000 02 ITMCT PIC 9999 COMP.  
 069000\* ITEM-COUNT, NOT SUBSCRIPTED.  
 070000 01 ERR-MSG.  
 071000 02 FILLER PICTURE X(41) VALUE  
 072000 '\*\*\* LM -- TABLE LOOKUP -- TOO MANY TABLES'.  
 073000 02 FILLER PICTURE X(92) VALUE SPACE.  
 074000 01 CRG-CTL PICTURE X VALUE SPACE.  
 075000 LINKAGE SECTION.  
 076000 01 USER-CALL-SEQ.  
 077000 02 IN-LENGTH PIC S9(6) COMP.  
 078000 02 OUT-LENGTH PIC S9(6) COMP.  
 079000 02 EXIT-FLAG PIC S9(6) COMP.  
 080000 02 IN-DATA PIC X(360).  
 081000 02 SEARCH-ARG REDEFINES IN-DATA PICTURE X(360).  
 082000 02 OUT-DATA PIC X(360).

```

MEMBER NAME  LMLOOK
C83000  C2  FIELD-NAME      PIC X(8).
C84000  C2  TABNAME        FEDEFINES FIELD-NAME PIC  X(8).
C85000  C2  FILE-NAME      PIC X(8).
C86000  PROCEDURE DIVISION.
C87000      ENTRY 'LMLKUP' USING USER-CALL-SEQ.
088000      MOVE 1 TO TANDEX.
C89000  TAD-SCAN.
090000      IF TADNAME (TANDEX) EQUAL TO TABNAME
C91000      GO TO FINDIT.
C92000      IF TANDEX LESS THAN TADCNT
C93000      ADD 1 TO TANDEX
C94000      GO TO TAD-SCAN.
C95000      IF TADCNT NOT LESS THAN TADMAX
C96000      GO TO PRINTIT.
097000      ADD 1 TO TADCNT.
C98000      MOVE TADCNT TO TANDEX.
099000      MOVE TABNAME TO TADNAME (TANDEX).
100000      GO TO LODIT.
101000  FINDIT.
102000      MOVE LODAX      (TANDEX) TO LODAD.
103000      MOVE TRIPLEX (TANDEX) TO TRIPLET.
104000      GO TO BIN.
105000  LODIT.
106000      CALL 'LOADTAB' USING TADNAME (TANDEX)
107000      LODAD.
108000      MOVE LODAD TO LCCAX (TANDEX).
109000      CALL 'ABGET' USING LODAD
110000      TRIPLET
111000      HWB-6.
112000      MOVE TRIPLET TO TRIPLEX (TANDEX).
113000  BIN.
114000      ADD 6, LODAD GIVING LOMARK.
115000      ADD ARGLEN FULLEN GIVING ITMLLEN.
116000      DIVIDE 2 INTO ITMLLEN GIVING ITMHAF.
117000      MULTIPLY ITMLLEN BY ITMCT GIVING HIMARK.
118000      ADD LOMARK TO HIMARK.
119000      SUBTRACT ITMLLEN FROM HIMARK.
120000      CALL 'COMABSY' USING LOMARK
121000      SEARCH-ARG
122000      ARGLEN
123000      RESULT.
124000      IF RESULT = 3
125000      GO TO FAIL.
126000      IF RESULT = 2
127000      MOVE LOMARK TO MIDMARK
128000      GO TO FOUND.
129000      CALL 'COMABSY' USING HIMARK
130000      SEARCH-ARG

```

ENCLOSURE C



MEMBER NAME	LMLOOK	
131000		ARGLEN
132000		RESULT.
133000	IF RESULT = 1	
134000	GO TO FAIL.	
135000	IF RESULT = 2	
136000	MOVE HIMARK TO MIDMARK	
137000	GO TO FOUND.	
138000	MOVE ITMCT TO HINDEX.	
139000	MOVE 1 TO LODEX.	
140000	BINI.	
141000	ADD HIMARK LCMARK GIVING MIDMARK.	
142000	DIVIDE 2 INTO MIDMARK.	
143000	ADD HINDEX, LODEX GIVING HILODEX.	
144000	DIVIDE HILODEX BY 2 GIVING MIDEX, REMAINDER REM.	
145000	IF REM = 1	
146000	SUBTRACT ITMHALF FROM MIDMARK.	
147000	IF LCMARK = MIDMARK	
148000	GO TO FAIL.	
149000	CALL 'COMABSY' USING MIDMARK	
150000		SEARCH-ARG
151000		ARGLEN
152000		RESULT.
153000	IF RESULT = 1	
154000	MOVE MIDMARK TO LCMARK	
155000	MOVE MIDEX TO LODEX	
156000	GO TO BINI.	
157000	IF RESULT = 3	
158000	MOVE MIDMARK TO HIMARK	
159000	MOVE MIDEX TO HINDEX	
160000	GO TO BINI.	
161000	FOUND.	
162000	MOVE 1 TO EXIT-FLAG.	
163000	ADD ARGLEN TO MIDMARK.	
164000	CALL 'ABGET' USING MIDMARK	
165000		OUT-DATA
166000		FUNLEN.
167000	GOBACK.	
168000	PRINTIT.	
169000	ENTER LINKAGE.	
170000	CALL 'FMPRT' USING ERP-MSG	
171000		CRG-CTL.
172000	ENTER COBOL.	
173000	FAIL.	
174000	MOVE 2 TO EXIT-FLAG.	
175000	GOBACK.	

(2) If the table is not yet loaded and there is no more room in the list, an error-message is inserted into OUT-DATA and control returns to the calling program.

(3) With TRIPLET given the proper contents, control goes to BIN, where the binary search starts. Searching continues in BIN1. If a hit is made during the search, control goes to FOUND; otherwise, it goes to FAIL.

(4) A few words are in order concerning the subroutines LOADTAB, ABGET, COMABSY. When LOADTAB executes the LOAD macro for the table, it returns to LMLOOK with the address of the table. This address cannot be used directly in a COBOL program as the operand of a MOVE or IF statement. Instead, when we need to move data from a field whose address (in a table) we know, to a field which is defined within a COBOL program, we use ABGET. Likewise, to compare a field within a table, known only by its address, to a field defined as a COBOL dataname, we use COMABSY ("compare absolute address and symbolic label").

d. Limitations. No more than 5 tables can be handled in one job step. This is due to the fact that the program must keep a list of what tables have been loaded, and there must be a limit to the length of this list. This limitation is an aggregate for all logic packages that may be operating in one job step.

### 13. LMTABGEN

a. Function. LMTABGEN is a free-standing program, which may be used to generate tables for use by the program LMLOOK. After a table is produced by LMTABGEN on dataset OBJDECK, it must be stored as a load module (in the IBM-360 implementation) by means of the Linkage Editor.

b. Calling Sequence. LMTABGEN uses the subroutine MUVE. Inputs and outputs are discussed in the user documentation. The table format, as stored on OBJDECK, is discussed below:

(1) The first 3 fields of the table contain, in order, the argument-length, function-length, and item-count for the table. The first two fields have PICTURE 99, the last has PICTURE 9999; all three have USAGE COMPUTATIONAL. The argument- and function-lengths give the number of characters for the argument and function fields in the table, while the item-count gives the number of argument-function pairs in the table.

ENCLOSURE C

(2) After these three fields, the arguments and functions follow in order: arg-1, func-1, arg-2, func-2,.... These fields are close-packed; that is, after the number of characters allotted to arg-1 by the argument-length field, the next character begins func-1, and so on. The argument-function pairs are arranged by ascending sequence of argument, to facilitate binary search.

c. Capabilities.

(1) Paragraph STARTALL makes general initialization for the program. READ1 processes the first card (carrying the table-name, argument-length, function-length) and checks for errors. READ2 processes all succeeding cards and stores the arguments and functions into UNSORTED-TABLE. Paragraph SORTIT starts a bubble-sort of this table, which is carried on by SORT1 and SORT2. Then comes SORTX, which formats and outputs the ESD-card and first TXT-card. Following TXT-cards are processed by TXT-LOOP, TXT-LOOP-1, which finally produces the END-card. At this point we have a complete object module on OBJDECK. However, Linkage Editor requires a NAME-card, consisting of the words:

NAME xxxxL(R)

where xxxxL is the module-name. This is built up by paragraphs REP-1, REP-2, and the job is done.

(2) Paragraph PRINTIT handles all printer output.

d. Limitations. This program is strongly machine-dependent, because of the requirement of storing the tables as load modules. However, its shortness will lighten the task of conversion to other machines. The JCL to run this program and process the output through the Linkage Editor is discussed in the MIDMS Procedures for System/360 OS Manual.

14. LOAD

a. Function. To dynamically load a subroutine or table load module without execution and return the address of the subroutine or table.

b. Calling Sequence. The subroutine can be called from a COBOL routine as follows:

CALL 'LOAD' USING ENTRY-ADDRESS.

ENCLOSURE C

where ENTRY-ADDRESS is a computational (PICTURE 9(5)) field into which the subroutine will place the initial execute address of the loaded subroutine or the load point address of a table.

MEMBER NAME LOAD

LOAD START

\* THE PURPOSE OF THIS SUBROUTINE IS TO LOAD A LOAD MODULE AND RETURN  
 \* ITS ENTRY POINT ADDRESS OR THE HOP OF A TABLE IN CORE.  
 \* THE SUBROUTINE'S CALLING SEQUENCE IN COBOL IS...

\* ENTER LINKAGE.

\* CALL 'LOAD' USING NAME ORIGIN.

\* ENTER COBOL.

\* NAME AND ORIGIN ARE DEFINED AS...

\* 01 NAME PICTURE X(8).

\* 01 ORIGIN PICTURE 9(5) COMPUTATIONAL.

SAVE (14,4)

SAVE REGISTERS

BALR 2,0

SET UP

USING \*,2

BASE REGISTER

ST 13,SAVE+4

SAVE OLD SAVE ADDR

LR 3,13

COPY OLD SAVE ADDR

LA 13,SAVE

LOAD NEW SAVE ADDR

ST 13,8(3)

PLACE NEW SAVE ADDR IN OLD SAVE AREA

LM 3,4,0(1)

LOAD PARAMETER LIST

LOAD EPLOC=(3)

LOAD THE LOAD MODULE

ST 0,0(4)

PLACE ENTRY POINT ADDR IN ORIGIN

L 13,SAVE+4

LOAD OLD SAVE ADDR

RETURN (14,4),T

RETURN

SAVE DS 18F

SAVE AREA

END

## 15. LOADTAB.

a. Function. Used by LMLKUP to read a table from the load-module library into available core.

b. Calling Sequence. From COBOL:

CALL 'LOADTAB' USING TABLENAM, TABLOC

where TABLENAM is an 8-character field containing the table name (left-adjusted with trailing blanks) and TABLOC is a fullword binary which will contain the beginning core address (HOP) into which the OS Supervisor loads the table.

ENCLOSURE C

c. Capabilities. The subroutine executes a LOAD macro for the module named by the user in TABLENAME. At return, TABLOC contains the core address of the module.

MEMBER NAME	LOADTAB	LOAD TABLE
LOADTAB	START 0	
* ROUTINE TO FETCH TABLE FROM LINK OR JOB LIBRARY VIA 'LOAD' MACRO.		
* CALL LOADTAB USING TABLENAME, TABLOC.		
* WHERE TABLENAME IS THE 8-CHAR NAME OF THE TABLE (LEFT-ADJUSTED,		
* W/ BLANKS) AND TABLOC IS A FULLWORD BINARY TO HOLD THE HOP OF THE		
* TABLE AFTER IT ENTERS CORE.		
	USING #,15	
	SAVE (14,12)	
	BALR 10,0	
	USING #,10	
	LR 12,13	
	LA 13,SV	
	ST 13,8(12)	
	ST 12,4(13)	
	LM 2,3,0(1)	
	LOAD EPLOC=(2)	
	ST 0,0(3)	
	LR 13,12	
	RETURN (14,12),T	
SV	DS 18F	
	END	

## 16. MOCHA

a. Function. To move a character string from one place to another, where the initial and final positions of the string are variable, as well as its length, and where the region finally occupied by the string may overlap some part of the initial region.

b. Calling Sequence. The subroutine may be called from COBOL as follows:

```
CALL 'MOCHA' USING SENDING-ARRAY
                   RECEIVING-ARRAY
                   SENDING-OFFSET
                   RECEIVING-OFFSET
                   LENGTH
```

ENCLOSURE C

where SENDING-ARRAY and RECEIVING-ARRAY are names of initial and final regions for the move, while the other parameters are fullword binaries. Each OFFSET is added to its ARRAY address to determine the first place to move from or move to (NOTE: If the first character of an ARRAY is required, OFFSET must contain zero). The number of characters to be moved is contained in LENGTH.

MOCHA     START 0

\* SUBROUTINE MOCHA -- THAT IS -- MO(VE) CHA(RACTERS)  
 \* IS DESIGNED TO MOVE A STRING OF CHARACTERS, OF VARIABLE LENGTH,  
 \* FROM ANY POSITION IN ONE ARRAY TO ANY POSITION IN ANOTHER. THIS  
 \* ROUTINE IS DESIGNED TO BE CALLED FROM 360-OS COBOL PROGRAMS TO  
 \* ACHIEVE STRING-MOVES, AS DESCRIBED, IN THE SHORTEST POSSIBLE TIME.

\*

\*

#### USAGE

\*

\*             ENTER LINKAGE

\*             CALL 'MOCHA' USING SENDING-ARRAY

\*                             RECEIVING-ARRAY

\*                             SENDING-OFFSET

\*                             RECEIVING-OFFSET

\*                             STRING-LENGTH

\*

\*             ENTER COBOL

\* THE FIRST TWO PARAMETERS MAY BE ANY ALPHANUMERIC STRINGS. THE OTHER  
 \* PARAMETERS MUST BE FULLWORD BINARIES -- PICTURE S9(N) COMPUTATIONAL  
 \* -- WHERE N MAY BE FROM 5 TO 9. THE FIRST (LEFTMOST) CHARACTER TO  
 \* BE MOVED IS FOUND BY ADDING THE CONTENTS OF 'SENDING-OFFSET' TO  
 \* THE ADDRESS OF 'SENDING-ARRAY', AND THE PLACE TO WHICH IT IS  
 \* MOVED IS FOUND BY ADDING THE CONTENTS OF 'RECEIVING-OFFSET' TO  
 \* THE ADDRESS OF 'RECEIVING-ARRAY'. THE TOTAL NUMBER OF CHARACTERS  
 \* TO BE MOVED IS FOUND IN 'STRING-LENGTH'.

\* NOTE THAT THE FIRST CHARACTER TO BE MOVED IN THE SENDING-STRING  
 \* IS NOT THE (SENDING-OFFSET)-TH CHARACTER, BUT THE (SENDING-  
 \* OFFSET + 1)-TH. THE SAME IS TRUE OF THE RECEIVING-AREA.  
 \* ONLY THOSE CHARACTERS IN THE RECEIVING-AREA ARE DISTURBED.

\*

\*

#### THEORY

\*

\* THE 5 PARAMETER-ADDRESSES ARE STORED IN REGISTERS 3 THROUGH 7, AND  
 \* CONTENTS OF REG 3,4 MODIFIED TO CONTAIN THE SENDING AND RECEIVING  
 \* ADDRESSES WITH OFFSETS ADDED. THE LENGTH OF STRING TO BE MOVED  
 \* GOES INTO REG 6.

\* IF THE MOVE IS LEFTWARD, (RECEIVING ADDRESS SMALLER THAN SENDING-  
 \* ADDRESS) OR IF THE SENDING AND RECEIVING STRINGS DO NOT OVERLAP,  
 \* CONTROL THEN PASSES TO 'LEFT' WHICH INITIALIZES THE LOOP TO MOVE  
 \* CHARACTERS INTO THE RECEIVING-AREA. THE CYCLICAL PART OF THIS

ENCLOSURE C

\* LOOP STARTS AT 'LFCYC'. THE ACTUAL MOVE IS HANDLED BY THE COMMAND  
 \* EX 6,MV1  
 \* IN WHICH THE MVC INSTRUCTION AT 'MV1' IS EXECUTED BY THE EX,  
 \* THE EFFECTIVE LENGTH OF THE MOVE BEING GIVEN BY THE LOW-ORDER  
 \* BYTE OF REG 6. EXIT FROM THE LOOP AND RETURN TO THE CALLING  
 \* PROGRAM IS DONE BY THE BZ INSTRUCTION.  
 \* IF THE MOVE IS RIGHTWARD WITH OVERLAP, DIFFERENT LOGIC IS REQUIRED  
 \* TO PREVENT CHARACTERS IN LEFTMOST PARTS OF THE SENDING-AREA FROM  
 \* DISTURBING CONTENTS OF POSITIONS YET TO BE SENT. HERE CONTROL IS  
 \* SENT TO 'RIGHT' WHICH INITIALIZES THE MOVE-LOOP THE CYCLICAL PART  
 \* BEGINNING AT 'RTCYC'. HERE THE SUCCESSIVE MOVES START AT THE RIGHT  
 \* END OF THE STRING AND WORK LEFT, IN CONTRAST TO LOOP 'LEFT', WHICH  
 \* WORKS LEFT-TO-RIGHT. AS A FURTHER PRECAUTION, LOOP 'RIGHT' PERFORMS  
 \* EACH MOVE IN TWO STEPS -- FIRST FROM THE SENDING-ARRAY INTO A  
 \* BLOCK CALLED 'HOLD', THEN FROM 'HOLD' TO THE RECEIVING-AREA.  
 \*

```

                SAVE (2,7)
                BALR 2,0
                USING *,2
                LM 3,7,0(1)
                A 3,0(5)
                A 4,0(6)
                L 6,0(7)
                CR 3,4
                BE RTN
                BH LEFT
RIGHT LR 7,6
        AR 7,3
        CR 7,4
        BNH LEFT
        AR 4,6
        LR 3,7
        STC 6,TEMP1+3
        S 3,TEMP1
        S 4,TEMP1
RTCYC BCTR 6,0
        EQU *
        EX 6,MV2
        EX 6,MV3
        SRDL 6,8
        SLA 6,8
        BZ RTN
        S 3,=F'256'
        S 4,=F'256'
        BCT 6,RTCYC
LEFT BCTR 6,0
LFCYC EQU *
```

```

        EX      6,MV1
        SRDL    6,8
        SLA     6,8
        BZ      RTN
        SRL     7,24
        LA      3,1(7,3)
        LA      4,1(7,4)
        BCT     6,LFCYC
HOLD    DS      256C
TEMP1   DC      F'0 '
MV1     MVC     0(0,4),0(3)
MV2     MVC     HOLD(0),0(3)
MV3     MVC     0(0,4),HOLD
RTN     RETURN  (2,7),T
        END

```

#### 17. MOVALF.

a. Function. To move an alphanumeric string into a receiving area, which may be longer than the string. The string is left-adjusted into the receiving area, with trailing blanks if necessary.

b. Calling Sequence. The routine may be called from COBOL as follows:

```

CALL 'MOVALF' USING ARRAYA INDEXA LENA
                      ARRAYB INDEXB LENB.

```

where:

- (1) ARRAYA contains the string to be moved.
- (2) INDEXA (fullword binary) gives the starting position of the string relative to ARRAYA (first position of ARRAYA counts as INDEXA=1).
- (3) LENA (fullword binary) is the length of the string to be moved.
- (4) ARRAYB contains the receiving area.
- (5) INDEXB (fullword binary) gives the starting position of the receiving area relative to ARRAYB (first position of ARRAYB counts as INDEXB+1).
- (6) LENB (fullword binary) is the length of the receiving area.

c. Limitation. In the 360 version, it is the user's responsibility to insure that LENA does not exceed LENB. Violation of this rule will cause the string to overflow the receiving area, overlaying whatever is stored in following locations.

ENCLOSURE C



```

MEMBER NAME  MOVALF
MOVALF      START 0
* THE CALLING SEQUENCE OF MOVALF IS
*          CALL 'MOVALF' USING ARRYA INDEXA LENA ARRAYB INDEXB
*          LENB.
          USING *,15
          SAVE (2,9)      SAVE REGISTERS
          LM 2,4,0(1)     LCAD R2,R3,R4
          LM 6,8,12(1)    LCAD R6,R7,R8
          L 3,0(3)        R3=INDEXA
          L 4,0(4)        R4=LENA
          L 7,0(7)        R7=INDEXB
          L 8,0(8)        R8=LENB
          AR 2,3          R2=A(2ND CHAR ARRYA)
          AR 6,7          R6=A(2ND CHAR ARRAYB)
          BCTR 2,0        R2=A( 1ST CHAR ARRYA)
          BCTR 6,0        R6=A(1ST CHAR ARRAYB)
          LTR 4,4         TEST FOR LENA = 0
          BE MVI
          LR 3,4          R3=R4
          BCTR 4,0        R4=LENA -1
LOOP1     EX 4,MOVE1
          SRDL 4,8        SHIFT R4 RIGHT 8 POS DBL
          SLA 4,8         SHIFT R4 LEFT 8 POS SNGL
          BZ CHKSPC       BRANCH IF ZERO
          SRL 5,24        SHIFT R5 RIGHT 24 POS
          LA 2,1(5,2)     UPDTAE R2=A(NEXT CHAR ARRYA)
          LA 6,1(5,6)     LDATE R6=A(NEXT ARRAYB)
          BCT 4,LOOP1     R4=LENA -256
MOVE1     MVC 0(0,6),0(2) MOVE ARRYA TO ARRAYB
CHKSPC    LR 4,3         R4=LENA
          CR 4,8          CCMPARE LENA TO LENB
          BE LEAVE        R6=A(1ST PCS ARRAYB FOR SPACE)
          AR 6,4          MCVE SPACE TO 1ST POS OF ARRAYB
MVI       MVI 0(6),C' '  MCVE ONE SPACE
          LA 4,2(4)       R4=LENA+2
          SR 8,4          R8=NUM OF ADDL SPACES TO BE MOVED - 1
          BM LEAVE        BRANCH IF NO ADDL SPACES REQD
LCOP2     EX 8,MVC
          SRDL 8,8        SHIFT R8 RIGHT 8 POS DBL
          SLA 8,8         SHIFT R8 LEFT 8 POS SNGL
          BZ LEAVE        BRANCH IF ZERO
          SRL 9,24        SHIFT R9 RIGHT 24 POS
          LA 6,1(6,9)     LDATE R6=A(NEXT CHAR AFRAYB)
          BCT 8,LOOP2     R8=R8 -256
MVC       MVC 1(0,6),0(6) PROPOGATE BLANKS RIGHT
LEAVE     RETURN (2,9),T  RETURN
          END

```

## 18. MOVCMP

a. Function. To test and move a numeric display field to a computational field, both within arrays. The to-field (output field) may be halfword or fullword.

b. Calling Sequence. The 360 COBOL Linkage:

```
CALL 'MOVCMP' USING ARRAYA INDEXA LENA  
ARRAYB INDEXB LENB INDICATOR.
```

where:

(1) ARRAYA is the location of the array from which the numeric display field is to be obtained.

(2) INDEXA is the subscript of the high order character of the numeric display field (9(5) COMPUTATIONAL).

(3) LENA is the length of the from-field (9(5) COMPUTATIONAL).

(4) ARRAYB is the location of the array containing the computational numeric field being changed.

(5) INDEXB is the subscript of the high order character of the to-field (9(5) COMPUTATIONAL). Proper halfword or fullword alignment is assumed.

(6) LENB is the length of the to-field (9(5) COMPUTATIONAL). Length 1-4 means halfword computational, 5-9 means fullword computational. LENB is always greater than or equal to LENA.

(7) INDICATOR is a computational (halfword) area to store an indicator of the result where:

0 = Good (Action Completed)  
1 = Bad (Action Not Completed)

ENCLOSURE C

```

MEMBER NAME  MOVCMF
MOVCMF      START 0
*R2=ARRAYA,R3=INDEXA,R4=LENA,R5=ARRAYB,R6=INDEXB,R7=LENB,R8=INDICATOR
* CDMS ROUTINE TO MOVE EXTERNAL DECIMALS INTO COMPUTATIONAL FIELD.
*
* THE CALLING SEQUENCE IS
*
* CALL 'MOVCMF' USING
*
* ARRAYA      ADDRESS FULLWORD REG 2. BINARY
* INDEXA      ADDRESS FULLWORD REG 3. BINARY - STARTING POS
* LENGTHA     ADDRESS FULLWORD REG 4. BINARY
* ARRAYB      ADDRESS FULLWORD REG 5. BINARY
* INDEXB      ADDRESS FULLWORD REG 6. BINARY - STARTING POS
* LENGTHB     ADDRESS FULLWORD REG 7. BINARY
* INDICATOR   ADDRESS HALFWORD REG8 BINARY
*
* USING *,15
* SAVE (2,9)
* LM 2,8,0(1)      ARRAYA ADDRESS
* L 3,0(3)         STARTING POSITION
* AR 2,3           ADD STARTING POSITION TO ARRAYA ADDRESS
* BCTR 2,0         DECREMENT 2 BY 1
* L 4,0(4)
* L 6,0(6)
* AR 5,6           ADD STARTING POS IN ARRAYB TO BEGINNING
* BCTR 5,0         - 1
* L 7,0(7)         ARRAYB LENGTH
* XC 0(2,8),0(8)   ZERO OUT INDICATOR
* CR 4,7           4 GREATER THAN 7
* BH MOVE1        ERROR
* LR 3,4
SCAN  CLI 0(2),X'FC' NUMERIC TEST
      BL MOVE1      DECREMENT 2 BY 1
      CLI 0(2),X'F9'
      BH MOVE1      ERROR
      LA 2,1(2)
      BCT 3,SCAN    TEST TO SEE IF ALL DIGIT ARE NUMERIC
      SR 2,4        SUBTRACT R4 FROM R2
      BCTR 4,0
      C 7,=F'9'
      BH MOVE1      ERROR
      EX 4,PACK1    PACK THE DATA
      CVB 9,LO      CONVERT TO BINARY
      C 7,=F'4'
      BH MOVE4      MOVE FULLWORD
      STH 9,DUBLWORD STORE HALFWORD
      MVC 0(2,5),DUBLWORD MOVE HALFWORD
RETURN RETURN (2,9)
MOVE4 ST 9,DUBLWORD STORE FULLWORD
      MVC 0(4,5),DUBLWORD MOVE FULLWORD
      B RETURN
MOVE1 MVI 1(8),X'01' ERROR ROUTINE
      B RETURN
PACK1 PACK LO(8),0(0,2)
LO DS D
DUBLWORD DS D
END

```

ENCLOSURE C

## 19. MOVCON

a. Function. To provide a subroutine to move two-byte fields from one array, convert them to binary, and move them to a second array.

b. Calling Sequence. The subroutine can be called from a COBOL routine as follows:

```
CALL 'MOVCON' USING ARRAYA, INDEXA, ARRAYB, INDEXB, LENGTHX.
```

where:

(1) ARRAYA is the name of an array of alphanumeric characters from which a string of consecutive 2 byte fields will be moved.

(2) INDEXA is the number of the first byte in ARRAYA to be processed. INDEXA is a fullword (computational).

(3) ARRAYB is the receiving array.

(4) INDEXB is the number of the first position in ARRAYB to where the halfword will be moved. INDEXB is a fullword (computational).

(5) LENGTHX is the number of bytes to be used in ARRAYA. LENGTHX is a fullword (computational).

```
MOVCON    START 0
* THE CALLING SEQUENCE OF MOVCON IS
*       CALL 'MOVCON' USING ARRAYA INDEXA ARRAYB INDEXB LENGTHX
* LENGTH IS A FULLWORD
          USING *,15
          SAVE (2,7)
          LM 3,7,0(1)
          L 7,0(7)
          BCTR 3,0
          BCTR 5,0
          A 3,0(4)
          A 5,0(6)
PAC"      PACK CONAREA,0(2,3)
          CVB 2,CONAREA
          STH 2,0(5)
          LA 3,2(3)
          LA 5,2(5)
          BCTR 7,0
          BCT 7,PACK
OUT        RETURN (2,7),T
CONAREA    DS D
          END
```

## 20. MOVNUM

a. Function. To test and move a numeric field to another numeric field, both with arrays, inserting leading zeroes when necessary.

b. Calling Sequence. MOVNUM can be called from a COBOL routine as follows:

```
CALL 'MOVNUM' USING ARRAYA INDEXA
                      LENA ARRAYB
                      INDEXB LENB INDICATOR
```

where:

(1) ARRAYA is the location of the array from which the numeric field is to be obtained.

(2) INDEXA is the subscript of the high order character of the numeric field (9(5) COMPUTATIONAL).

(3) LENA is the length of the field (9(5) COMPUTATIONAL).

(4) ARRAYB is the location of the array to which the numeric field is to be moved.

(5) INDEXB is the subscript of the high order character of the to-field (9(5) COMPUTATIONAL).

(6) LENB is the length of the to-field (9(5) COMPUTATIONAL). LENB is not less than LENA.

(7) INDICATOR. When this field is numeric, it is set to 1. Normally, it is 0 (9(5) COMPUTATIONAL).

```
MEMBER NAME  MOVNUM
MOVNUM      START 0
* THE CALLING SEQUENCE IS
*           CALL 'MOVNUM' USING  ARRAYA INDEXA LENA
*                               ARRAYB INDEXB LENB INDICATOR
* INDICATOR IS A HALFWORD.
          USING *,15
          SAVE  (2,9)
          LM    2,8,0(1)
          L     3,0(3)
          L     4,0(4)
          L     6,0(6)
          L     7,0(7)
          AR    2,3
```

ENCLOSURE C

```

        AR      5,6
        BCTR    2,0
        BCTR    5,0
        LTR     9,4
        BZ      ACCEPT
CLI     CLI     0(2),X'4C'
        BE      SKPBLK
LOOP    CLI     0(2),X'F0'
        BL      LEAVE
        CLI     0(2),X'F9'
        BH      LEAVE
ACCEPT2 LA      2,1(2)
        BCT     9,LOOP
ACCEPT  SR      2,4
        LR      9,7
        SR      9,4
        BZ      JUMP
        MVI     C(5),X'F0'
        BCT     9,BCTR
        B       ADD1
BCTR    BCTR    9,0
        EX      9,MOVEZ
        LA      9,1(9)
ADD1    LA      9,1(9)
JUMP    AR      5,9
        LTF     4,4
        BZ      FLGZERO
        BCTR    4,0
EX4     EX      4,MOVEN
FLGZERO MVC     0(2,8),=H'0'
RETURN  RETURN  (2,9),T
LEAVE   C       9,=F'1'
        BE      SGNCK
FLGCNE  MVC     0(2,8),=H'1'
        B       RETUPN
SGNCK   CLI     0(2),X'CO'
        BL      FLGONE
        CLI     0(2),X'CA'
        BL      ACCEPT2
        CLI     0(2),X'D0'
        BL      FLGONE
        CLI     0(2),X'D9'
        BH      FLGONE
        B       ACCEPT2
SKPBLK  LA      2,1(2)
        BCTR    4,0
        BCT     9,CLI
        B       ACCEPT
MOVEZ   MVC     1(0,5),0(5)
MOVEN   MVC     0(0,5),0(2)
        END

```

ENCLOSURE C

## 21. MOVRAY

a. Function. To provide a subroutine to move one array of characters to another area (not an array) and to the same character positions.

b. Calling Sequence. The subroutine can be called from a COBOL routine as follows:

```
CALL 'MOVRAY' USING XFROM XINDEX XTO LENGTHX.
```

where:

(1) XFROM is the name of an array of alphanumeric characters from which a string of consecutive bytes are to be moved.

(2) XINDEX is the name of a computational field (fullword) representing the character position within an array (located at XFROM) from which the move is to begin.

(3) XTO is the name of an area (not an array) to which the character(s) are to be moved.

(4) LENGTHX is the name of a computational (fullword) field (value from 1 to 256) representing the number of characters to be moved.

```
MOVRAY  START 0
        USING *,15
        SAVE (3,6)
        LM 3,6,0(1)
        L 6,0(6)
        BCTR 6,0
        A 3,0(4)
        BCTR 3,0
        EX 6,MOVEX
        RETURN (3,6),T
MOVEX   MVC 0(0,5),0(3)
        END
```

## 22. MUVE

a. Function. To provide COBOL programmers with a subroutine which can move a string of characters of arbitrary length from one area to another.

ENCLOSURE C

b. Calling Sequence. The following example is as if the subroutine would be called from a COBOL routine:

```
CALL 'MUVE' USING A(I), B(J), K.
```

(1) Example 1. Assume that areas A and B are defined in COBOL as follows:

```
01  AA.
```

```
02  A PICTURE X OCCURS 1000 TIMES.
```

```
01  BB.
```

```
02  B PICTURE X OCCURS 1000 TIMES.
```

Now assume, before entering linkage, that I=50, J=172, and K=300. Subroutine MUVE will move 300 characters from area A starting in position 50 to area B starting in position 172. The user may want to use 'MUVE' as follows.

(2) Example 2.

```
MOVE 999 TO K.
```

```
MOVE 'X' TO A(1).
```

```
ENTER LINKAGE.
```

```
CALL 'MUVE' USING A(1), A(2), K.
```

```
ENTER COBOL.
```

The character X will be propagated from A(1) through A(1000) with the one call to MUVE. A and B must be character-defined areas. K must be defined as S9(5) computational. The names "A, B, and K" may be user defined.

c. Limitations. No test is made to determine if the storage area B can contain K characters from A. Subroutine 'MUVE' requires 64 bytes of storage.

ENCLOSURE C



```

MUVE    START 0
        USING *,15
        SAVE (2,5)
        LM 2,4,0(1)
        L 4,0(4)
        BCTR 4,0
NEXT    EX 4,MVIT
        SRDL 4,8
        SLA 4,8
        BZ MVOUT
        SRL 5,24
        LA 2,1(5,2)
        LA 3,1(5,3)
        BCT 4,NEXT
MVIT    MVC 0(0,3),0(2)
MVOUT   EQU *
        RETURN (2,5),T
        END

```

### 23. OPR34

#### a. Function.

(1) To obtain from a character array the characters that describe OPR34(B,C).

(2) The format of the OPR is as follows:

T nnnn L LITERAL OPR#

(3) Where:

T = Type (3,4), 1 position numeric FLD.  
 nnnn = Four character subscript or numeric HOP of data field.  
 L = Length of literal (1 position numeric FLD).  
 LITERAL = Characters 'L' in number.  
 OPR# = Two digit number.

b. Calling Sequence. The subroutine can be called by a COBOL routine as follows:

```

CALL 'OPR34' USING ARRAY, INDEX, POSIT, LENGTH,
LIT-HOLD, OPRNUM.

```

ENCLOSURE C

where:

(1) ARRAY is the name of an array of characters that describe a string of OPR's.

(2) INDEX is the position in the array where a particular OPR starts.

(3) POSIT will contain the four character subscript (nnnn) described above.

(4) LENGTH (PICTURE 9(5) COMPUTATIONAL) will be the place where 'L' is moved.

(5) LIT-HOLD (PICTURE X(9)) is place where the literal is moved.

(6) OPRNUM (PICTURE 99 COMPUTATIONAL) is place where OPR number will be moved.

```
OPR34  START      Ø
*      THE CALLING SEQUENCE FOR OPR34
*      IS: CALL 'OPR34' USING ARRAY, INDEX,
*      POSIT, LENGTH, LIT-HOL, OPRNUM
      USING      *,15
      SAVE      (2,8)
      LM        2,7,Ø(1)
      A         2,Ø(3)
      MVC       Ø(4,4),Ø(2)
      IC        8,4(2)
      LA        3,15
      NR        8,3
      ST        8,Ø(5)
      BCTR      8,Ø
      EX        8,MOVIT
      LA        2,6(8,2)
      IC        4,Ø(2)
      NR        4,3
      NR        6,3
      LA        5,10
      MR        4,4
      AR        5,6
      STH       5,Ø(7)
      RETURN    (2,8),T
MOVIT  MVC      Ø(Ø,6),5(2)
      END
```

ENCLOSURE C

## Ancillary System Routines

### Section II - Honeywell

#### 1. BIBCS

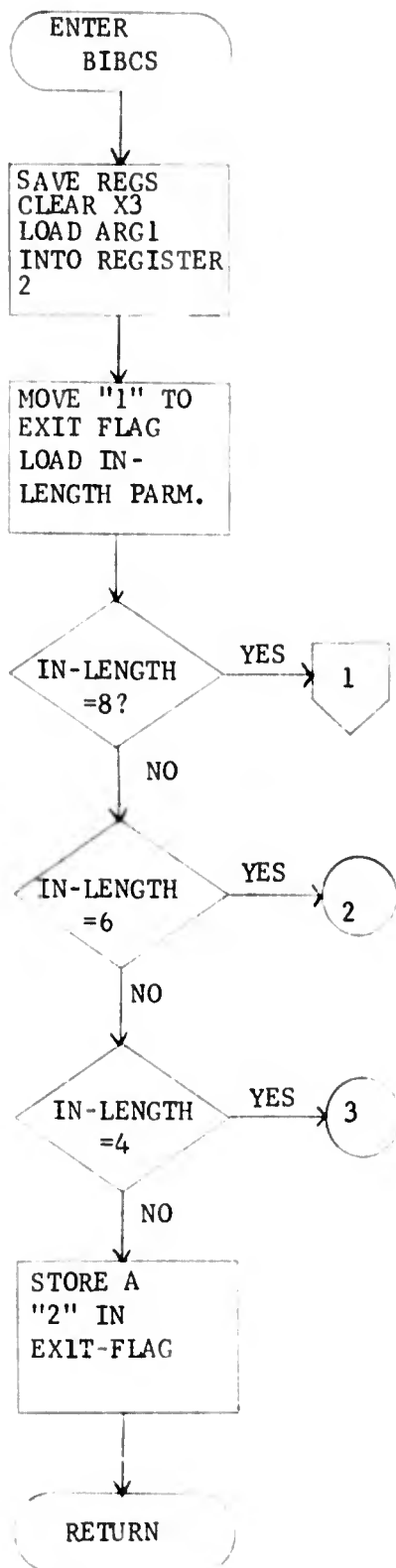
a. Summary. This subroutine converts a variable length binary number to binary coded decimal (BCD). Upon normal completion, an exist-flag is set to "1". If bad in-date is found, the exit-flag is set to "2".

(1) Function. Binary to BCD conversion subroutine.

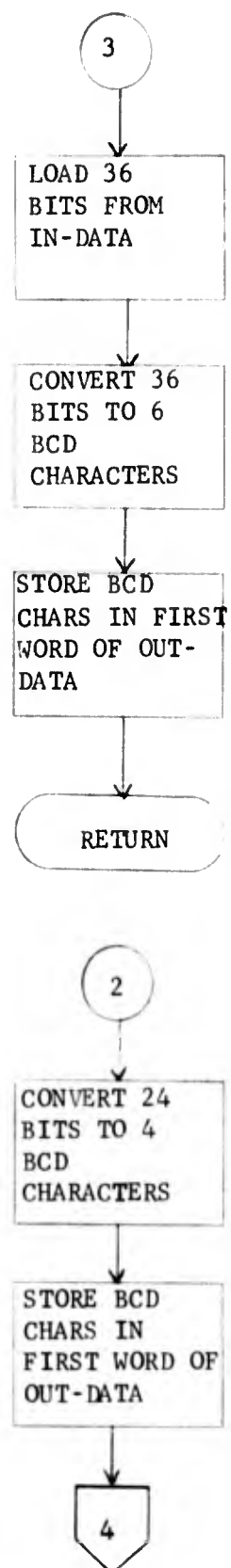
(2) Calling Sequence. Standard user-calling-sequence.  
DIAM 65-9-9A, page D-1.

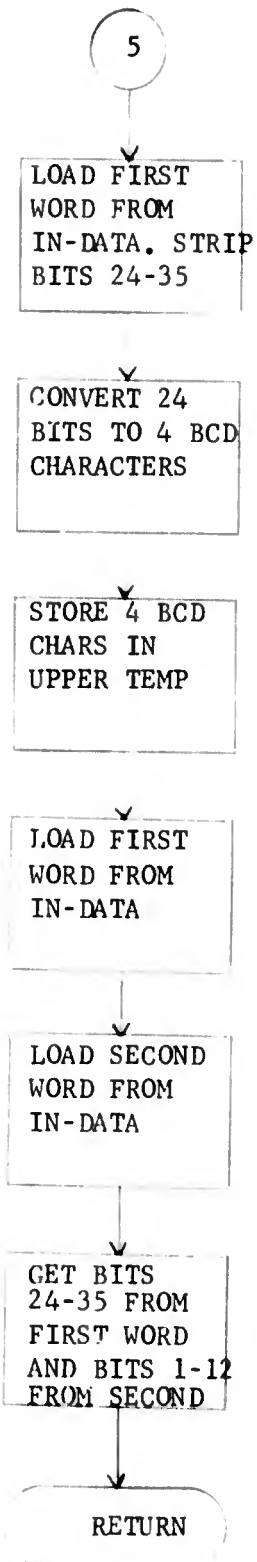
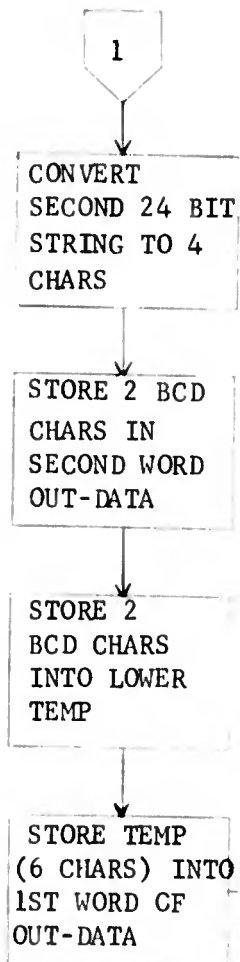
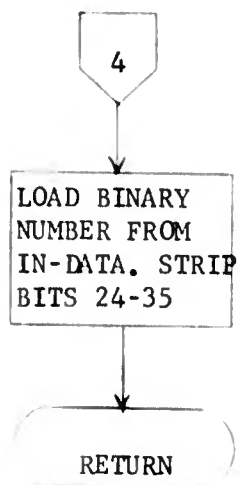
b. Description. BIBCS converts a binary number varying from 24 bits to 48 bits to BCD using the GMAP RPT instruction. The BCD characters (either 4, 6, or 8) are stored in the out-data address of the standard user-calling-sequence.

c. BIBCS Flowchart.



ENCLOSURE C





ENCLOSURE C

## 2. COMLST

a. Summary. The purpose of this subroutine is to compare a field value in an input-record against a list of values in an array. When a match is found between the input record value and the list, an indicator is set to "2" and control passed to the calling program. If no match is found, the indicator is set to zero before returning.

(1) Function. To locate in an array an FMIPVAL OPCODE corresponding to the particular OPCODE in the input-record.

(2) Calling Sequence.

CALL COMLST USING INPUT-REC INDEXA VAL-LIT-AREA INDEXB  
NO-OF-ITEMS LENGTH INDICATOR.

where:

INPUT-REC is the input-record address.

INDEXA is the character offset for the address.

VAL-LIT-AREA is the list address.

INDEXB is the character offset for the list.

NO-OF-ITEMS is the COMP-1 value for the number of items in the list to search.

LENGTHA is the COMP-1 length value of the input-record field.

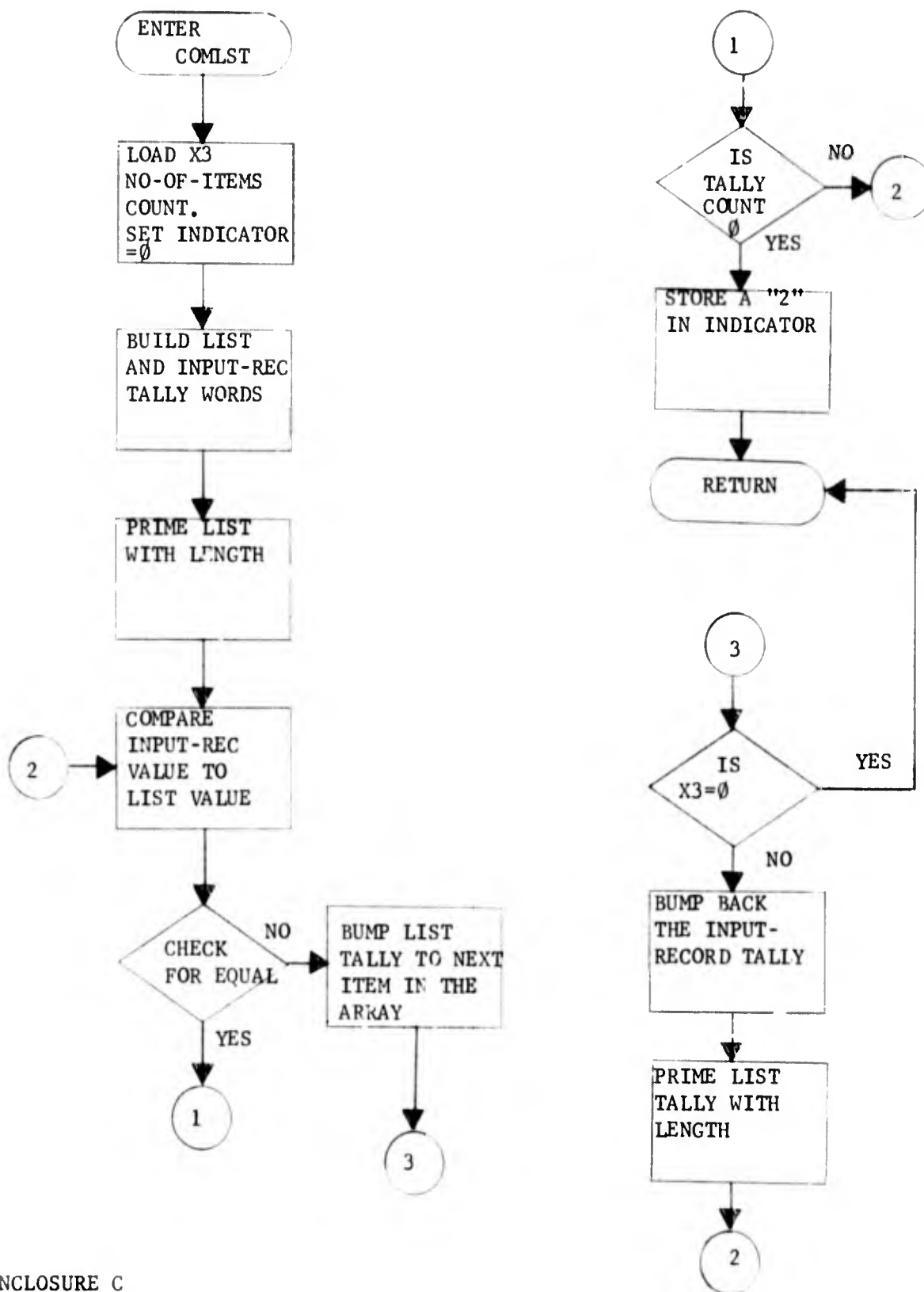
INDICATOR is the COMP-1 indicator where the result is stored.

b. Description. COMLST locates field values in an array by using two tally words to compare the input record against the list.  
For example:

01\*\*\*\*INPUT-RECORD\*\*\*\*GEN\*\*\*\* (INDEXA POINTS TO THE "G")  
01\*\*\*\*VAL-LIT-AREA\*\*\*\*CHGADDGENDEL\*\*\*\* (INDEXB POINTS TO "C")  
THE LENGTH PARAMETER IN THIS CASE IS EQUAL TO 3, AND THE  
NO-OF-ITEMS PARAMETER IS EQUAL TO 4. IN THE ABOVE EXAMPLE  
A COMPARE IS MADE THRU NO-OF-ITEMS (3) AT WHICH TIME A "2" IS  
PASSED TO INDICATOR AND CONTROL PASSED TO THE CALLING PROGRAM.  
IF A MATCH IS NOT FOUND BETWEEN THE VALUE AND THE ITEMS IN  
VAL-LIT-AREA, A ZERO IS STORED IN INDICATOR.  
INPUT-RECORD AND VAL-LIT-AREA ARE PICTURE X.  
INDEXA INDEXB NO-OF ITEMS-LENGTH AND INDICATOR ARE COMPUTATIONAL-1.

Register 2 is used for addresses, and register 3 is used for the NO-OF-ITEMS counter.

c. FCOMLST Flowchart.



ENCLOSURE C



### 3. COMRAY

a. Summary. COMRAY compares one array of characters to another array, not necessarily a one-to-one position correspondence. COMRAY can compare from 1 to 4096 characters. A Computational-1 indicator is set to "2" if the specified number of characters of ARRAYA compare equally to ARRAYB. When a character in ARRAYA is encountered that is higher in collating sequence than the ARRAYB character, the indicator is set to "3". If an ARRAYA character is lower in comparison to the ARRAYB character, indicator is set to "1". When either a high or low condition is encountered, no further comparisons are made, the appropriate value is stored in the indicator, and a return is made to the calling program.

(1) Function. Compare alphanumeric character arrays.

(2) Calling Sequence.

CALL COMRAY USING ARRAYA, INDEXA, ARRAYB,  
INDEXB, LENGTH, INDICATOR.

where:

ARRAYA and ARRAYB are addresses.

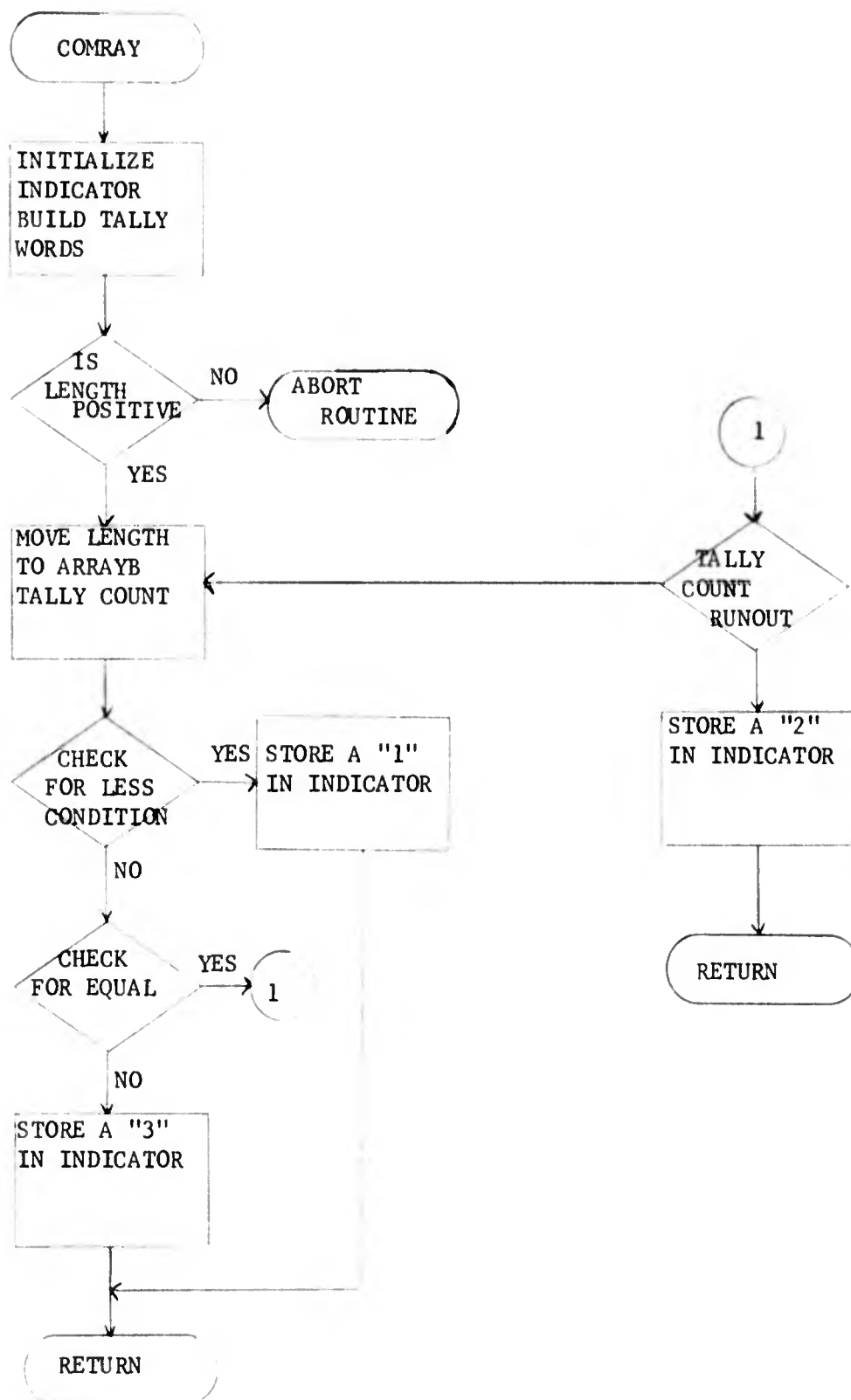
INDEXA and INDEXB are character offsets.

INDICATOR is a location to result the result.

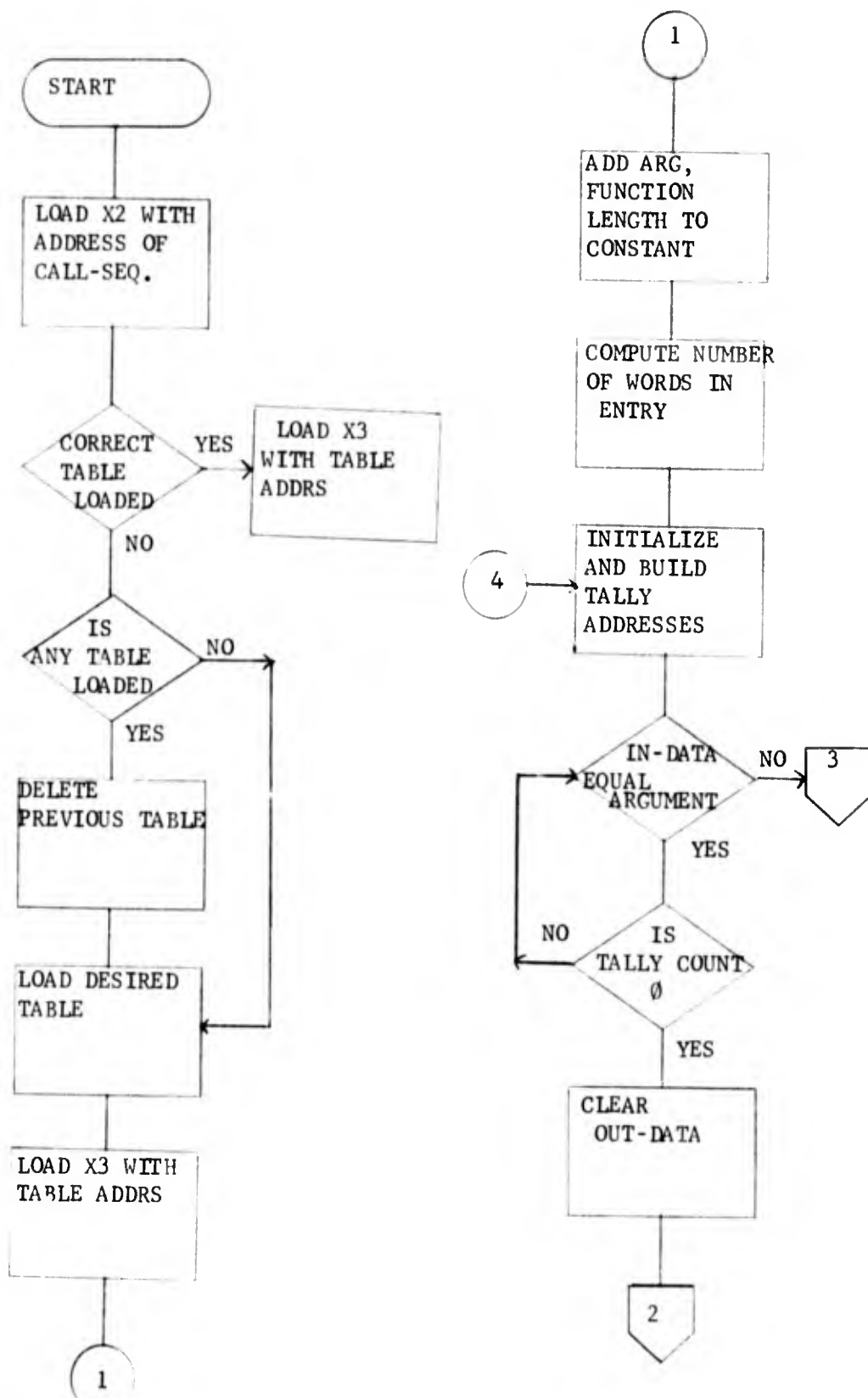
b. Description. The first instruction initializes the indicator to zero. The next 16 instructions build the ARRAYA and ARRAYB tally's consisting of addresses, and starting character positions. The next five instructions accept the length parameter, check it for validity (i.e., a positive integer), and stores the length into the receiving area tally. The next seven instructions compare the specified number of characters and branches to the appropriate address. The last four instructions store the correct value into the indicator address and executes a return to the calling program.

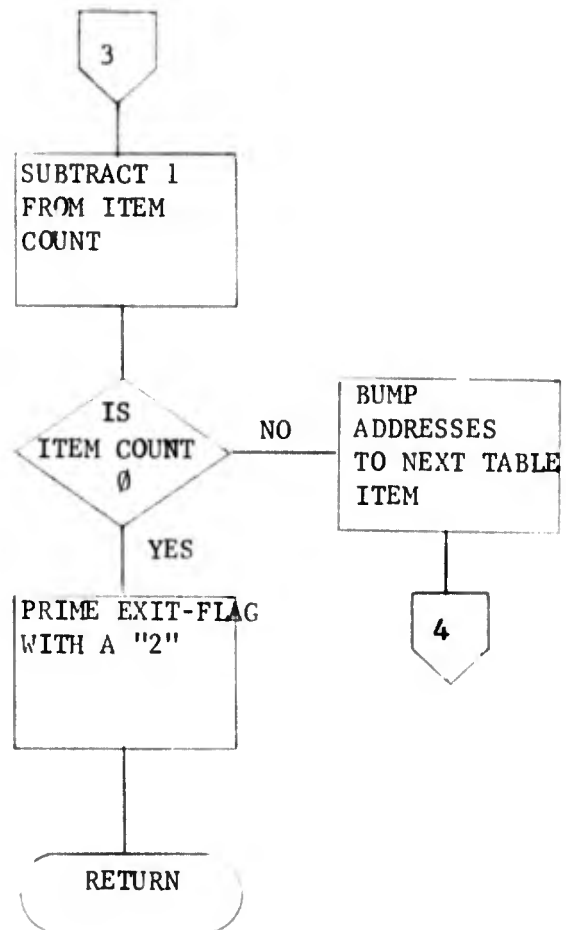
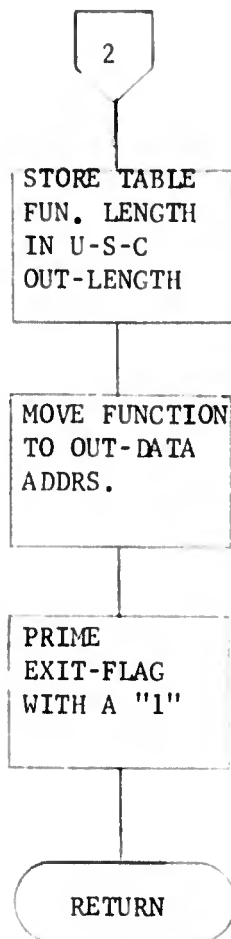
c. COMRAY Flowchart.

ENCLOSURE C



ENCLOSURE C





ENCLOSURE C

#### 4. MOVNUM

a. Summary. This subroutine moves any number of numeric characters from one array to another array.

(1) Function. To test and move a numeric field to another numeric field, both within arrays, inserting leading zeroes when necessary. If non-numeric data is found in the from-field, an indicator is set to "1" and a return made to the calling program. MOVNUM can insert all zeroes in the receiving-region by setting the sending-length to zero and the receiving-length to n (where n is a positive integer from 1 to 4096). MOVNUM terminates and sets the indicator to "2" if receiving-length is less than sending-length. Upon normal termination, the indicator is set to zero.

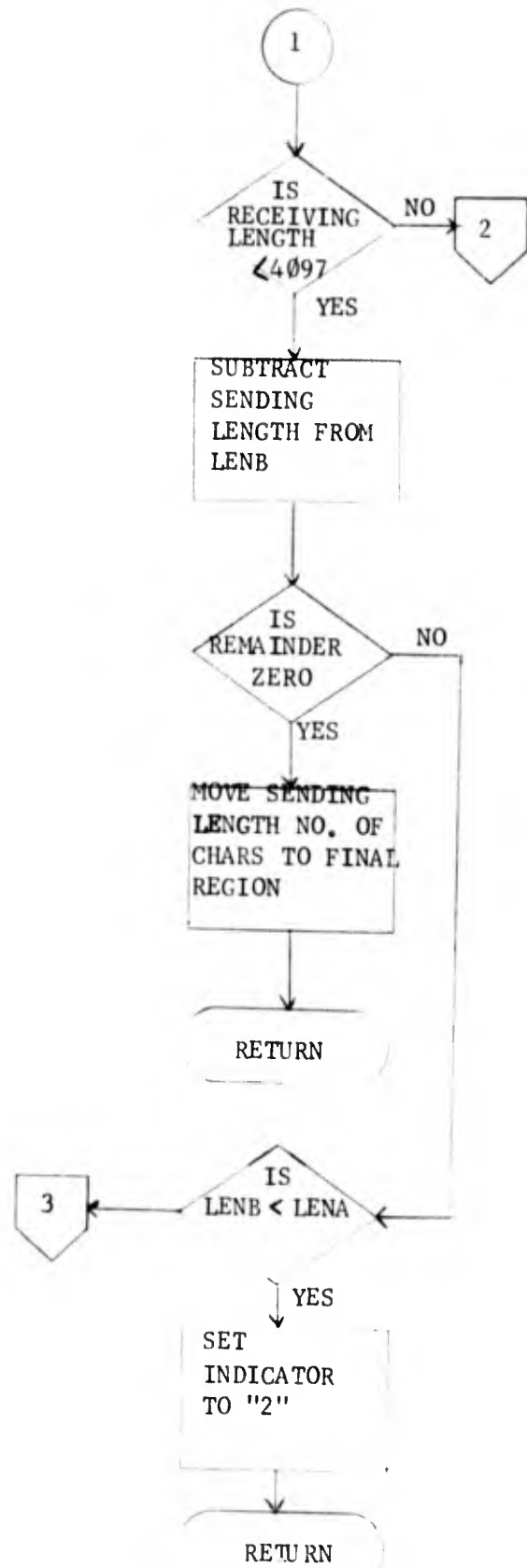
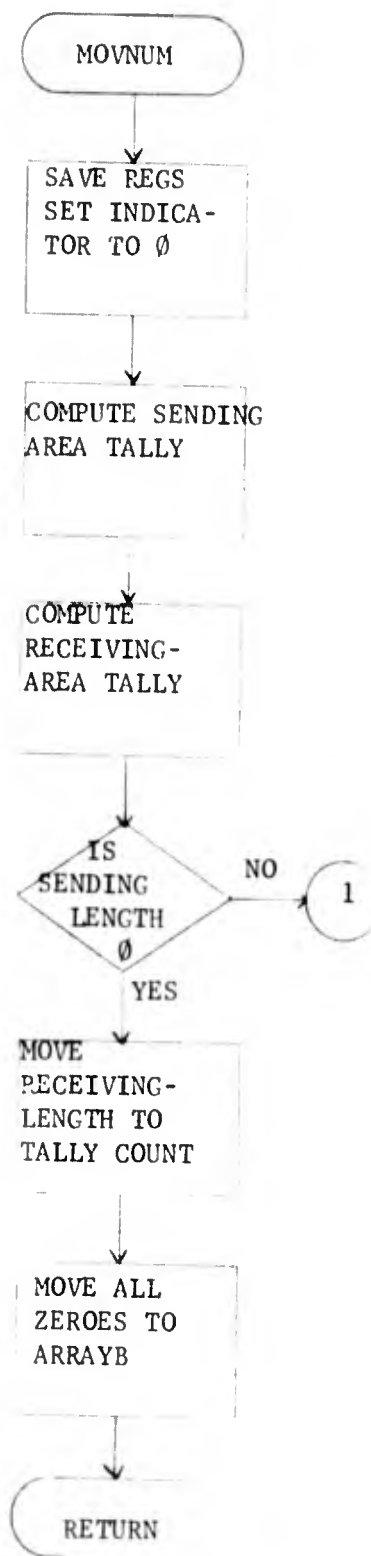
#### (2) Calling Sequence.

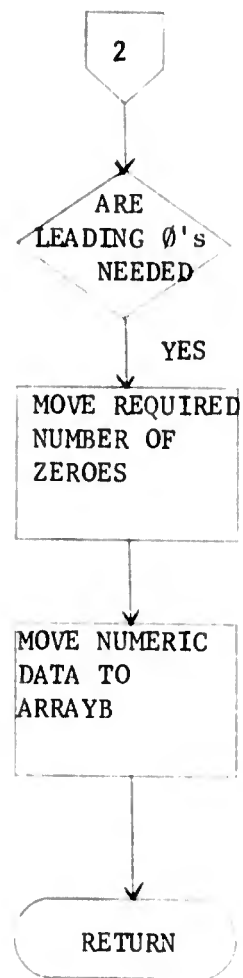
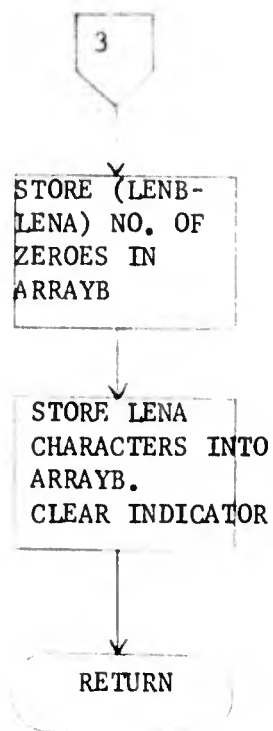
CALL MOVNUM USING ARRAYA, INDEXA, SENDING-LENGTH,  
ARRAYB, INDEXB, RECEIVING-LENGTH,  
INDICATOR.

b. Description. Data is moved from one location in memory to another location by means of two sequence character (SC) tally words. After the tallies are computed, a determination is made whether all zeroes are required in the receiving field. If so, transfer is made to a special routine. If more than 4096 characters are to be moved, a special loop moves them in blocks of 4096 characters. If the receiving-region is larger than the sending-region, leading zeroes are moved prior to the numeric data. Data is checked for validity before it is moved to the receiving-region.

#### c. MOVNUM Flowchart.

ENCLOSURE C





ENCLOSURE C

## 5. MOV PAC

a. Summary. This subroutine is used by OMLP to take a series of two character opcodes from a table, and place them in an array.

(1) Function. To move 'n' number of 12 bit opcodes from a binary table and pack them into an array.

(2) Calling Sequence.

CALL MOV PAC USING POSIT-GROUP, COUNT, VAL-LIT-AREA,  
VAL-LIT-HOP.

where:

POSIT-GROUP is a computational-1 table where opcodes are stored in the low order 12 bits of each word.

COUNT is the number of words to process.

VAL-LIT-AREA is an address for the beginning of the receiving array.

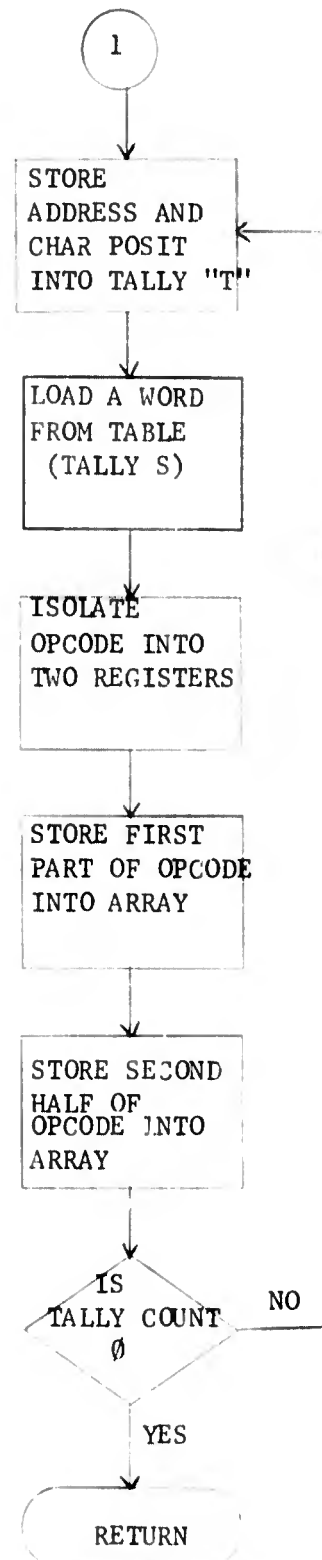
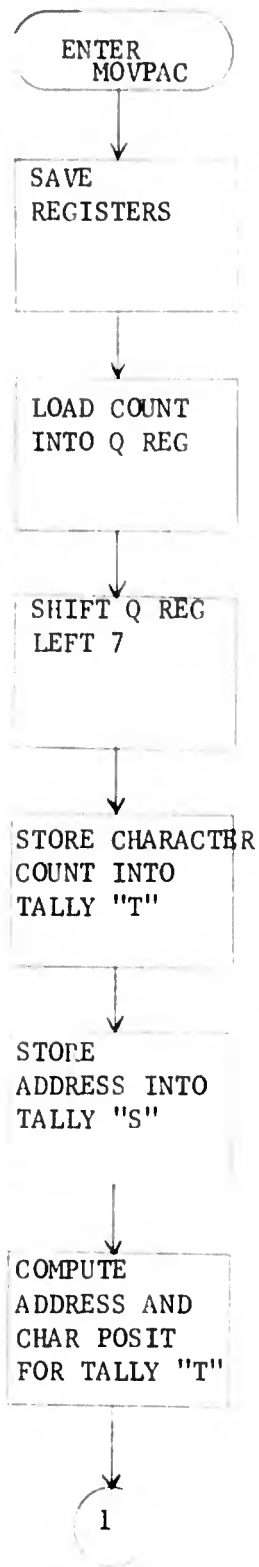
VAL-LIT-HOP is an index to the desired character position within VAL-LIT-AREA.

b. Description. After register 1, 2 and 3 are saved, the count is multiplied by 2 and stored in bits 18-29 of "T", thus forming the runout tally count. The address of the binary table is stored directly in bits 0-17 of tally word "S". The receiving area character position is divided by six and added to the address for the address of tally word "T". The residue character position is stored into bits 30-35 of "T". The 12 bit opcodes are moved from the table to the array in the following fashion: The A register is loaded with the first word of the table. Characters 4 and 5 are shifted to the right 6 positions thus placing character 5 into the left side of the Q register. The Q register is shifted right 30 positions placing the character in the low order end of the Q. The A register (containing character 4) is stored in array "T" with a sequence character tally word. The Q register (containing character 5) is then stored into "T" at the next character position, by the same tally word. The tally count is tested to see if there are any more words to process. If so, transfer is made to the location where the table words are loaded. If the tally count is zero, an immediate return is executed.

c. MOV PAC Flowchart.

ENCLOSURE C





## 6. MOVRAY

a. Summary. The MOVRAY subroutine is a generalized routine called by COBOL programs to move data from one location in core to another.

(1) Function. To move an alphanumeric string of characters from the INDEXA (TH) character of ARRAYA to the INDEXB (TH) character of ARRAYB. The characters may or may not be located on a word boundary. The receiving location may be a different location within a word.

(2) Calling Sequence.

CALL MOVRAY USING ARRAYA, INDEXA, ARRAYB, INDEXB, LENGTH.

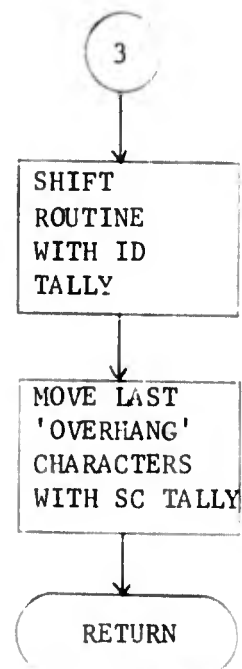
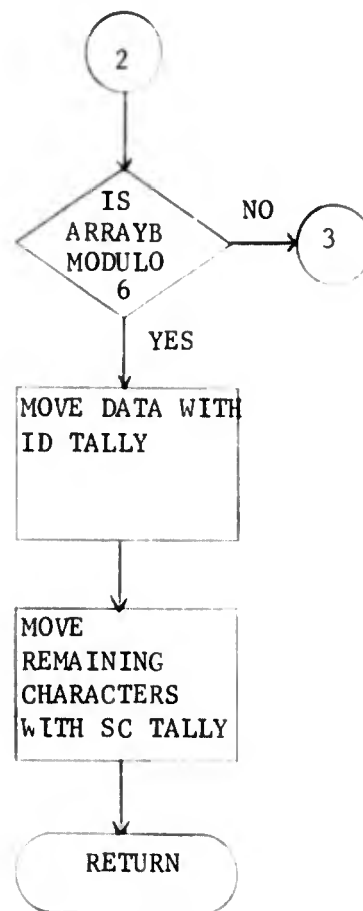
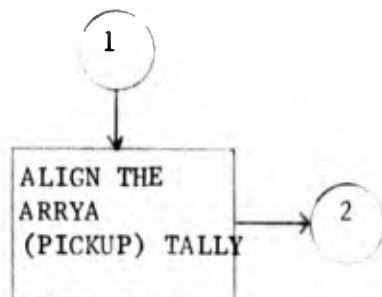
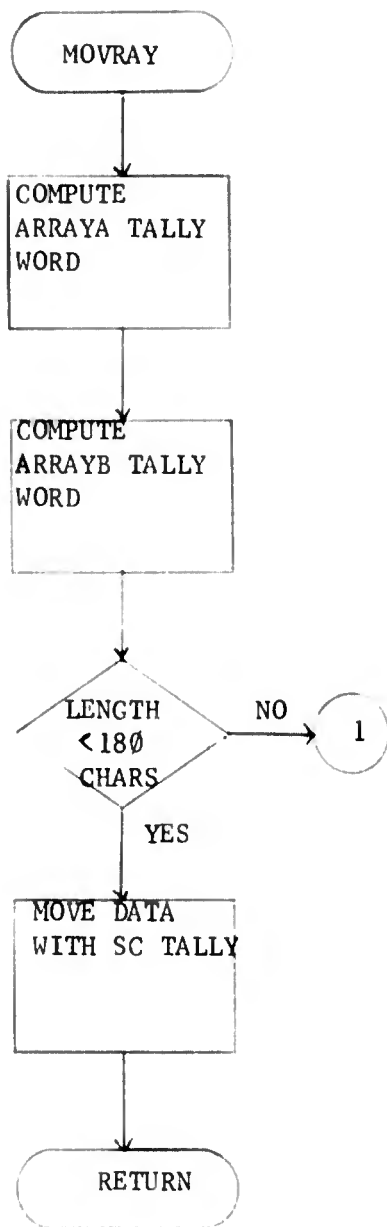
where:

All parameters are 01 levels. The array's are picture X; the indexes and length parameters are fullword binaries.

b. Description. The MOVRAY subroutine can move from 1 character to 4096 words of data. The length parameter is checked for validity, and if less than 180 characters are to be moved a branch is made to an SC tally routine. If LENGTH is 180 or greater, the pickup SC tally is aligned to a module 6 boundary by storing into the receiving area. That number is subtracted from LENGTH, and the remainder resolved into words (COUNT). A determination is made whether the ARRAYB (receiving) area is aligned on a boundary. If it is, both indexes were offset the same, and the move is accomplished with ID tallys. If ARRAYB is not aligned, ID tallys are used and the data is aligned within the registers prior to storing into the receiving area. After the word tally has runout the remaining number of characters is computed and stored with an SC tally word.

c. MOVRAY Flowchart.

ENCLOSURE C



## 7. OPR34

a. Summary. This subroutine obtains from a 6-bit character array the characters that describe OPR34. An OPR has the following format: TNNNNLLITERALOPR#, where T is the OPR type, NNNN is the high order position of the data field, L is a one character field giving the length of the literal, and LITERAL is characters "L" in number. OPR# is a two digit number.

(1) Function. Enter an array containing a string of OPR's, and breakdown a single OPR into four elements, moving the elements to locations specified by the calling sequence.

(2) Calling Sequence.

CALL OPR34 USING ARRAY PNTR POSIT LENGH LIT-HOLD OPRNUM.

where:

ARRAY contains the address of the array containing an OPR string.

PNTR is a COMP-1 number used to modify the ARRAY address, pointing to "T".

POSIT is the location where NNNN is to be moved.

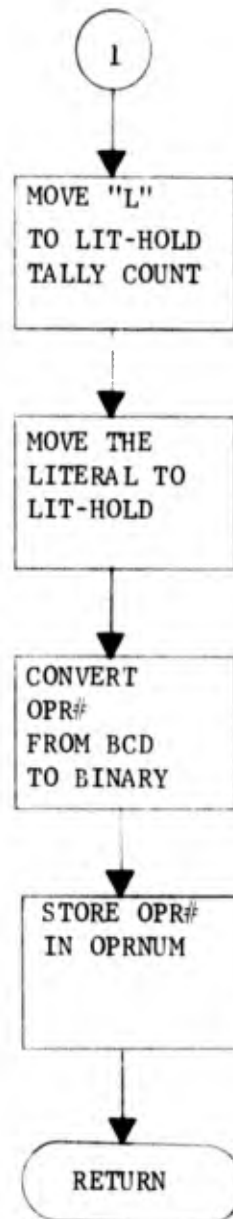
LENGTHA specifies the number of characters in the literal.

LIT-HOLD is the location where the literal is to be moved.

OPRNUM contains the address where the OPR# is to be moved.

b. Description. The first instruction initializes LIT-HOLD to zero. The next seven instructions build the pickup tally word. The next four instructions build the receiving area tally word. Following that is the code to move four 6-bit characters to the location specified by POSIT. The next two instructions move the length parameter to an address. Seven instructions are used to move the literal to LIT-HOLD. The last two characters in the OPR string are converted to binary in the OPR string are converted to binary and stored in OPRNUM. The last instruction is a return to the calling program.

c. OPR34 Flowchart.



ENCLOSURE C

## 8. PUTPSC

a. Summary. Subroutine PUTPSC is called from the OMP module to move all periodic set control fields (PSC's) from a two word area to an array.

(1) Function. To assemble a 48 bit PSC from the data found in two word LOCATIONS, and move the PSC to a character oriented array.

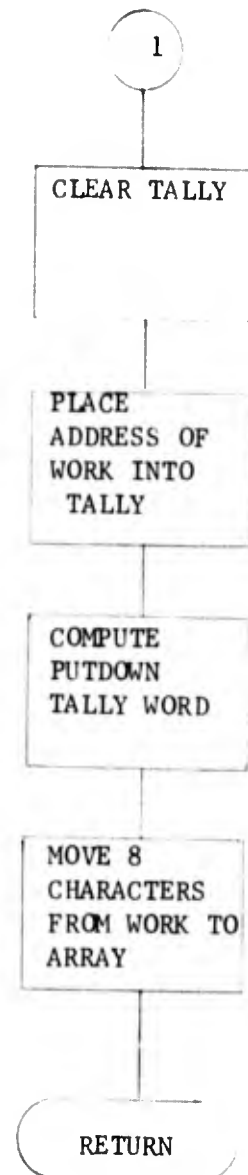
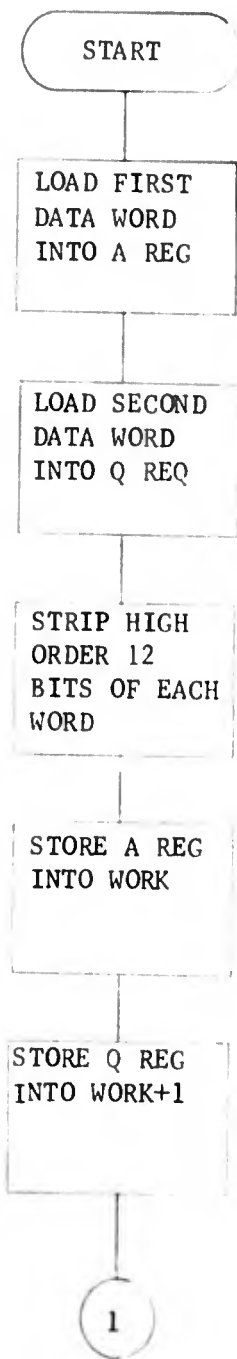
(2) Calling Sequence.

CALL PUTPSC USING FROM-ADDRESS, TO-ARRAY, PSCHOP.

b. Description. Register 2 is loaded with the address of the first word of data. The accumulator is loaded with the first word of data and the second word is loaded into the Q register. The desired data is located in the lower order 24 bits of each word. This data is separated and stored contiguously into WORK and WORK+1. The pickup tally is initialized to zero, and the address of BSS location WORK is placed into TALLY. The receiving tally word is computed erasing the second and third parameters from the calling sequence. The eight characters comprising a PSC are moved from the location WORK to the array, and a RETURN executed.

c. PUTPSC Flowchart.

ENCLOSURE C



ENCLOSURE C

## 9. RDPSCS

a. Summary. The READ periodic set control subroutine is used during execution of MIDMS logical maintenance.

(1) Function. Subroutine RDPSCS moves all periodic set control fields and variable set control fields in a MIDMS logical record to a table that will be referenced during execution of the logic package.

(2) Calling Sequence.

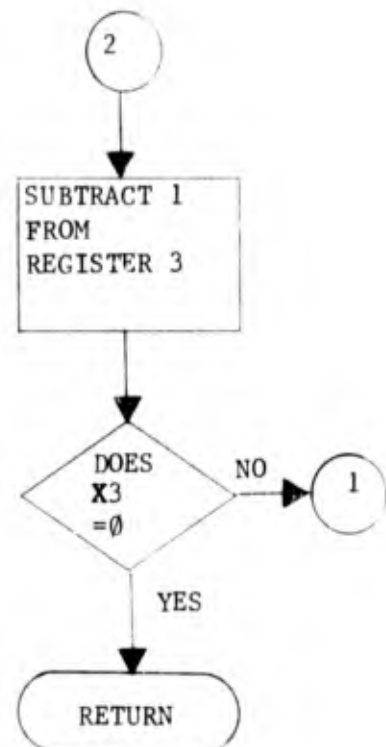
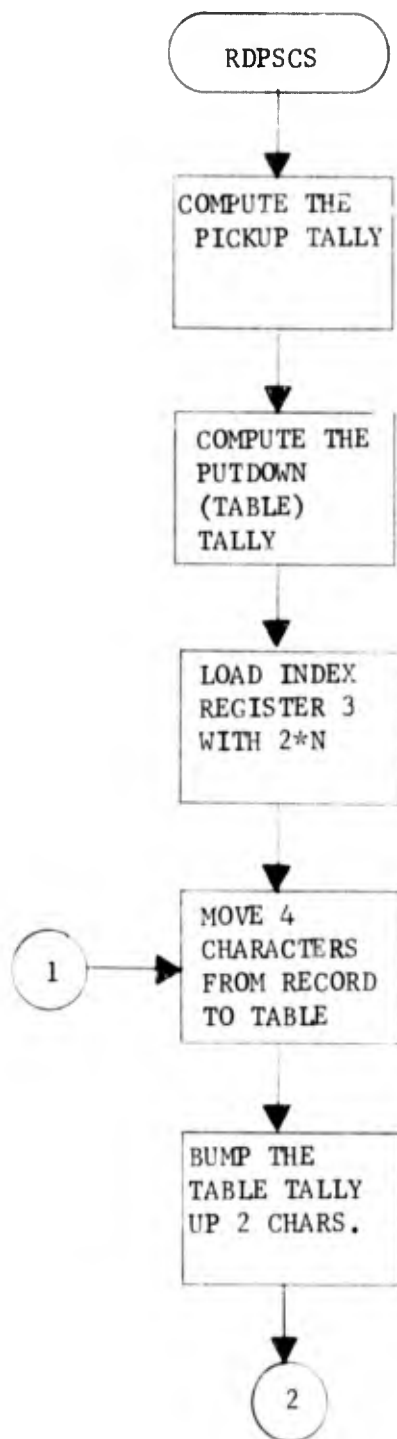
CALL RDPSCS USING RECORD PSCS-HOP TABLE N.

b. Description. PSC and VSC control fields located contiguously in a character string (array) are moved four characters at a time to consecutive word locations in a table. An SC tally word is computed for the RECORD, and functions as the pickup tally. An SC tally with character position two is built for the initial putdown (TABLE) tally. The count (N) is checked for validity, multiplied by two, and placed in index register 3. The control fields are moved by loading and storing the A register four times into the table address. After the fourth storage operation, the TABLE SC tally is incremented twice with the NOP instruction in order to place the TABLE tally on character position two of the next consecutive word. Index register 3 is decremented by one to determine if N is zero. If not zero, transfer is made to the load and store routine. If N equals zero, an immediate RETURN instruction is executed.

c. RDPSCS Flowchart.

ENCLOSURE C





ENCLOSURE C

10. WTPSCS

a. Summary. The write periodic set control subroutine is used during execution of MIDMS logical maintenance.

(1) Function. To move the periodic and variable set control fields from the table holding them to an output MIDMS logical record.

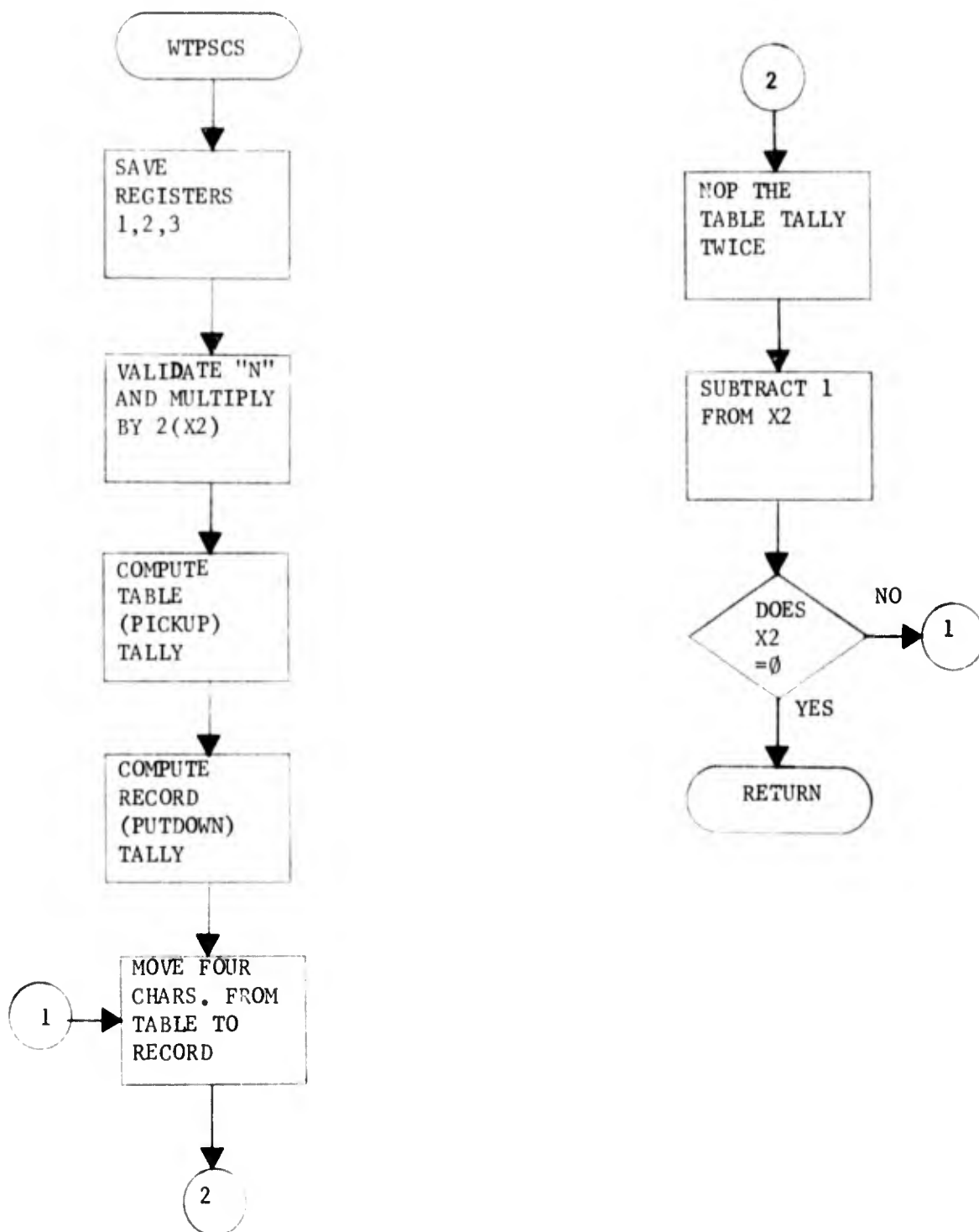
(2) Calling Sequence.

CALL WTPSCS USING TABLE, RECORD, PSC-HOP, N.

b. Description. PSC and VSC control fields are located in bits 12-35 of consecutive word addresses of a table. These control fields are moved to and stored consecutively in an array (MIDMS record). Parameter N is validity checked and multiplied by two, so that the contents of two words are moved if N equals 1. Table (pickup) and record (putdown) tallies are computed and stored in TABLE and RECORD. Movement of data occurs in the loop "MOVE" by loading and storing the TABLE and RECORD SC tally four times, and then bumping the TABLE tally ahead 12 bits to character position two of the next word. The contents of index register 2 determines whether to transfer back to MOVE, or to fall through to a RETURN instruction.

c. WTPSCS Flowchart.

ENCLOSURE C



ENCLOSURE C

## 11. YYDDD

a. Summary. This subroutine converts the HIS 6000 system date to a BCD Julian date format. YYDDD is dependent upon the computer operator entering the correct date into the system.

(1) Function. To compute the Julian date and store it in a five character field accessible to the calling program.

(2) Calling Sequence.

CALL YYDDD USING PARM1.

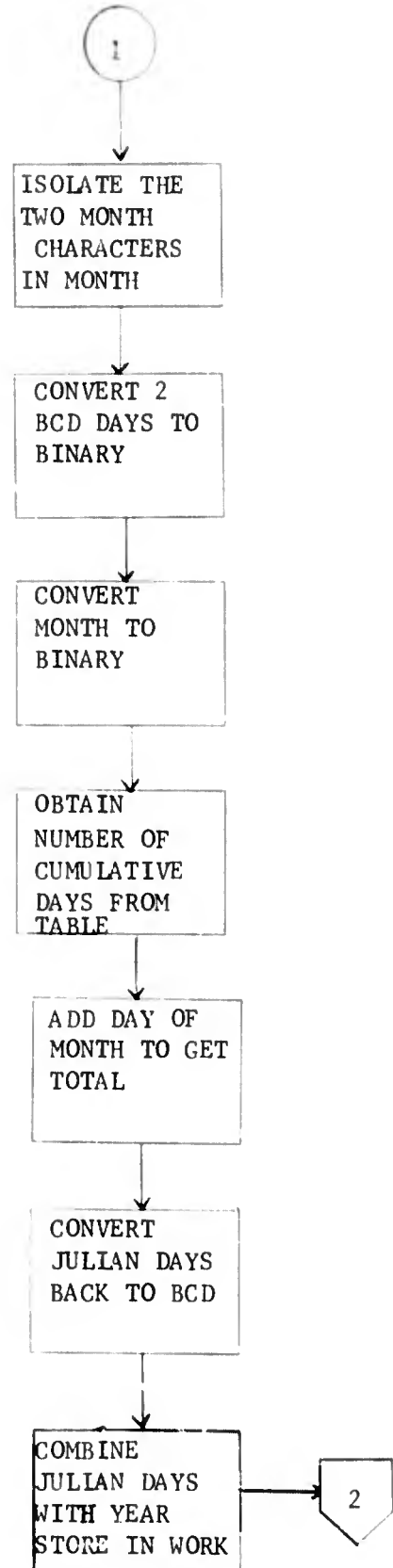
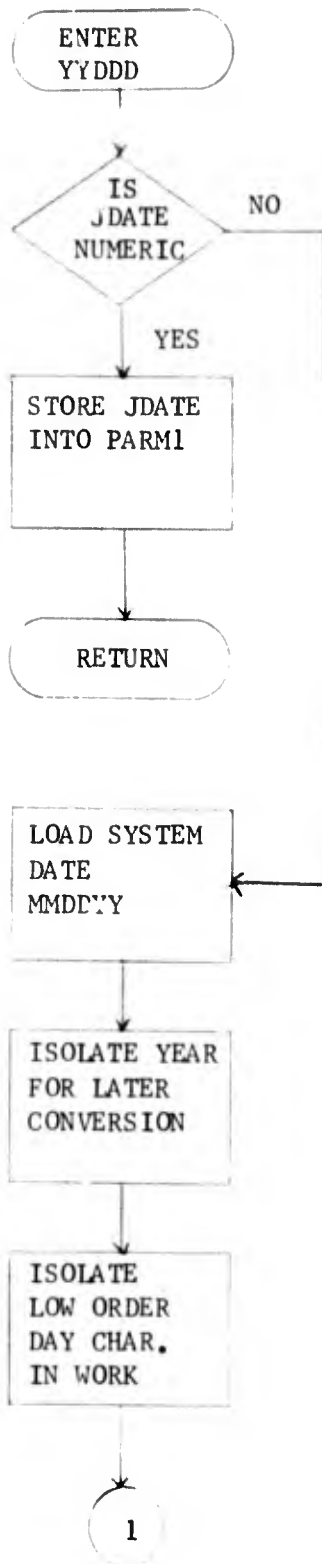
where:

PARM1 is a picture 9(5) address where YYDDD places the five character Julian date.

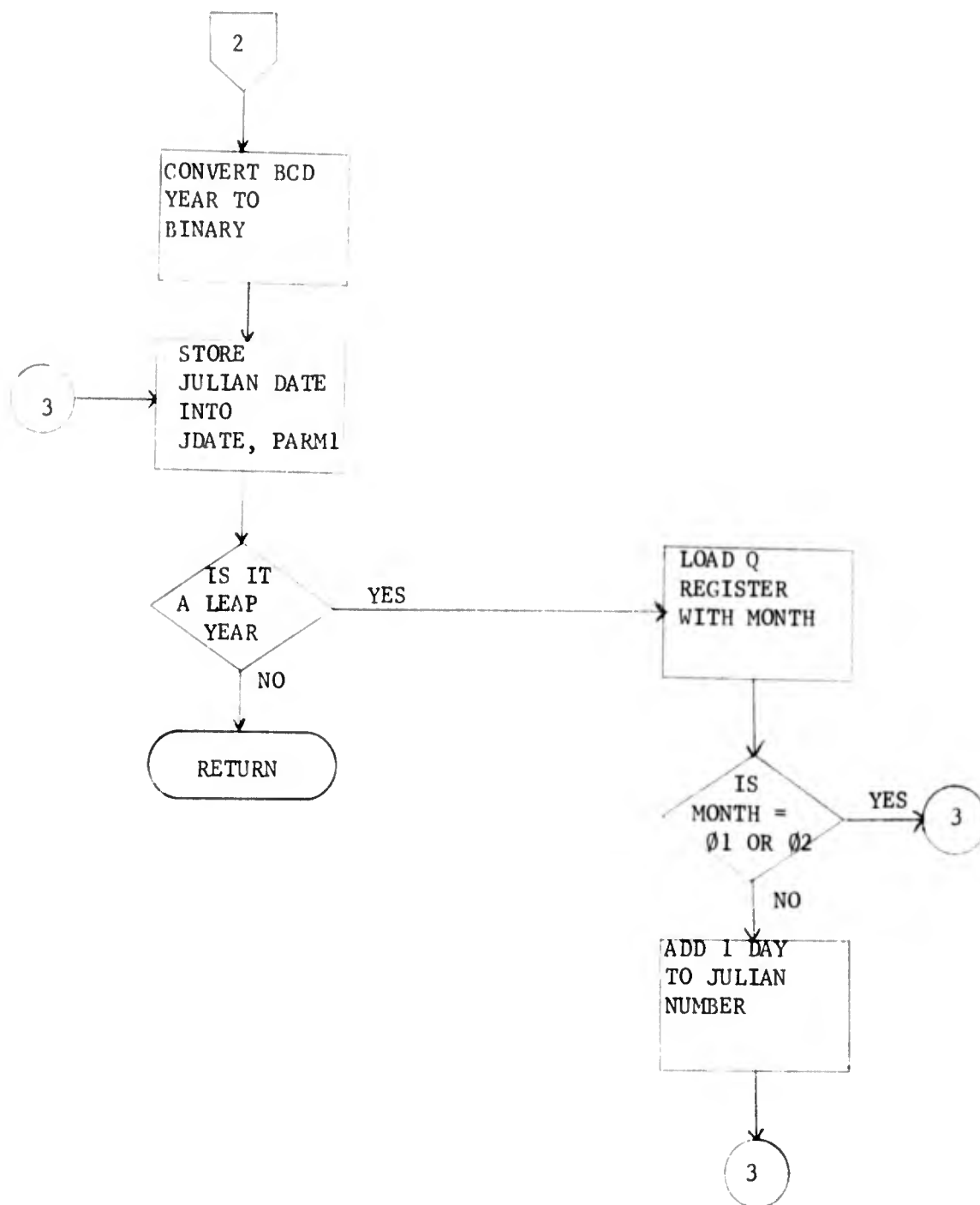
b. Description. Subroutine YYDDD first determines whether the Julian date has been previously computed by checking a one word BCI hold area. If the date is in JDATE, it is stored back into PARM1 and a return is immediately executed. If the date has not been computed, the system BCD date is obtained by issuing a MME GETIME instruction. The year is saved for later conversion to binary. The two month characters are stored in MONTH. Next, the two BCD day characters are converted to binary and stored in WORK. The MONTH is converted to binary and then converted by means of a table to the number of cumulative days. The binary day of the month is added. Next, the binary format Julian days are converted to BCD format, and combined with the year. This gives the Julian date in YYDDD BCD format, which is stored in WORK. To check for a leap year, the year is converted to binary and divided by four. If there is a remainder, it is not a leap year, so the Julian date is stored in PARM1 and JDATE and a return executed. If it is a leap year and the month is either January or February, the date is stored as before and a return made. If it is a leap year and the month is March or later, one day is added to the day portion of the Julian date, the date is stored in PARM1 and JDATE, and a RETURN instruction executed.

c. YYDDD Flowchart.

ENCLOSURE C



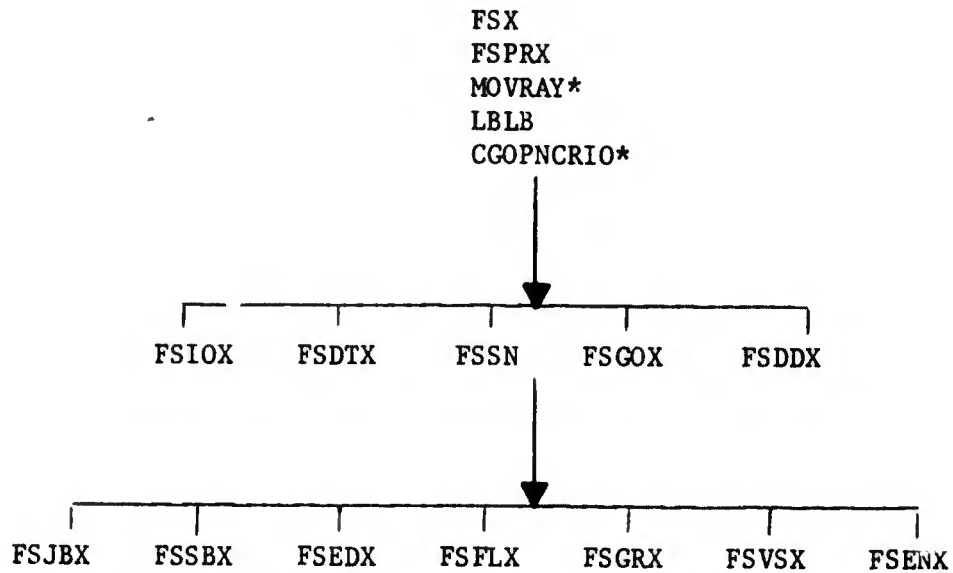
ENCLOSURE C



ENCLOSURE C

## Honeywell Differences

1. FILE STRUCTURING. The Honeywell version of MIDMS differs only in its overlay structure as shown below:

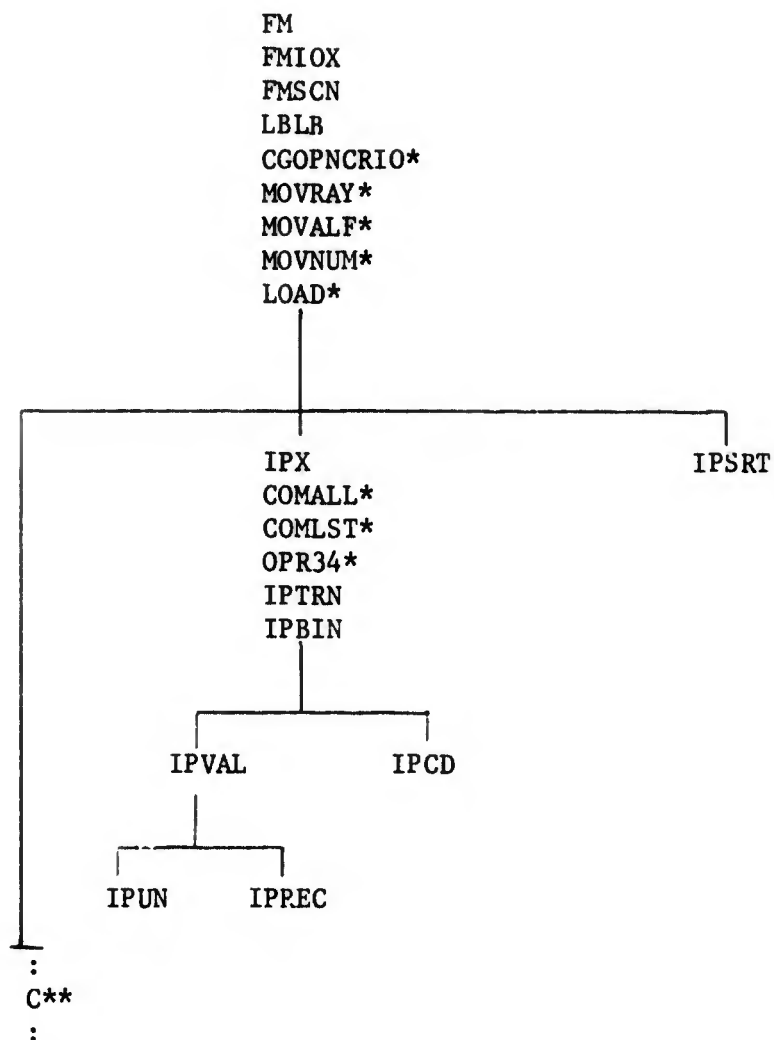


\*GMAP Programs

### Honeywell Overlay Structure (File Structuring)

2. FILE MAINTENANCE. The Honeywell version of MIDMS differs in its structure for the Input Processor and in the configuration of transaction records as shown on the diagrams on the next page.

ENCLOSURE D



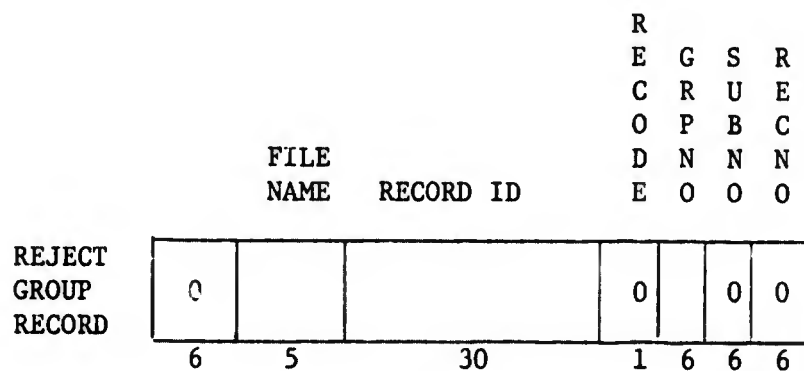
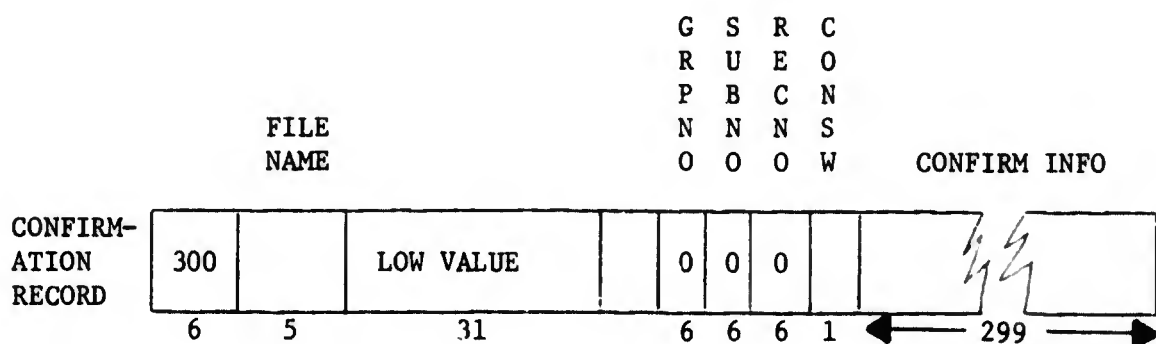
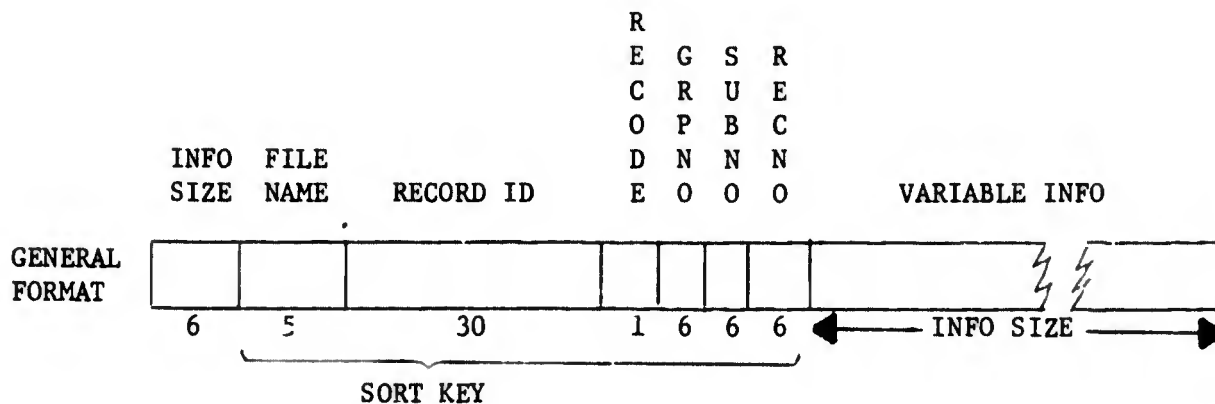
\*GMAP Routines

\*\*C is a variable block common area

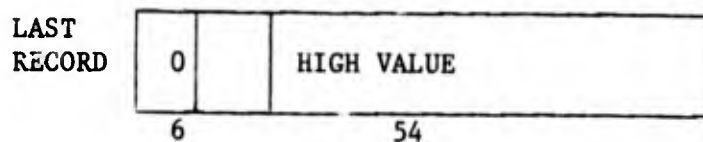
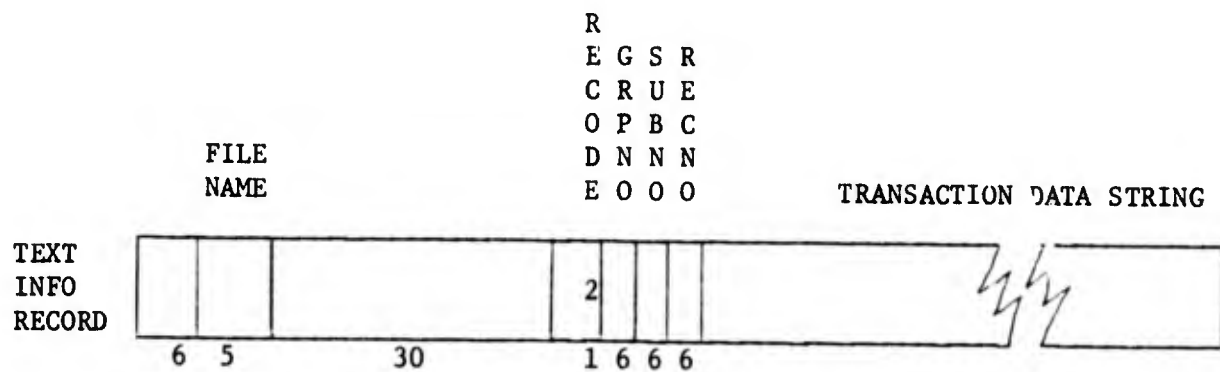
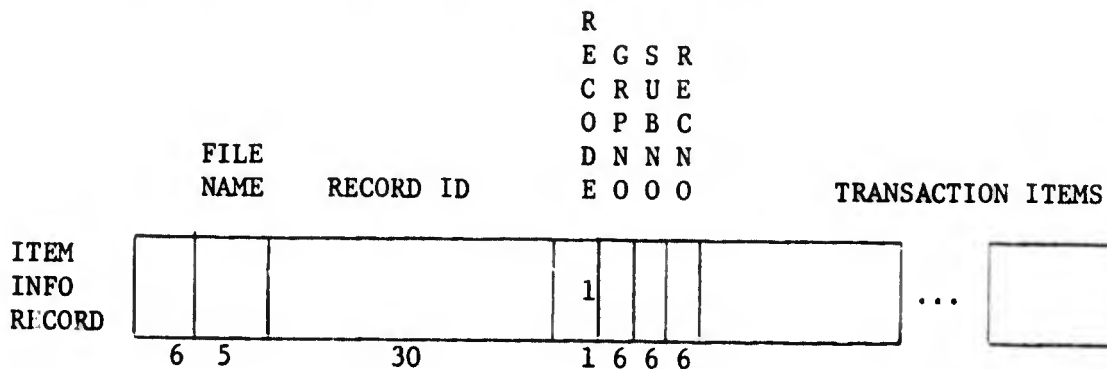
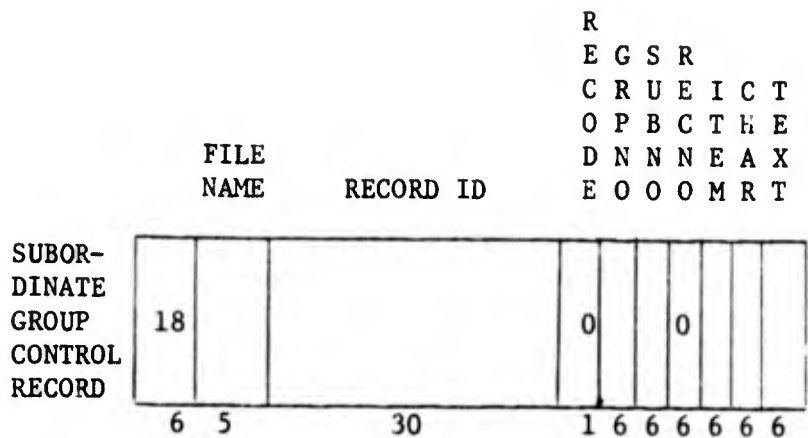
FM Input Processor Structure for HIS 600/6000

ENCLOSURE D





Transaction Record Configuration



Transaction Record Configuration (Continued)

3. LOGICAL MAINTENANCE. Relatively few changes in the generated COBOL program were required in conversion from IBM COBOL to HIS. The most substantial changes were required in the DATA DIVISION portion of the generated COBOL program. The Logical Maintenance modules which were modified for these changes were:

LMLPPG1  
LMLPPG2  
LMLPWP4

Generated code differences (HIS) are as follows:

a. LMLPPG1

(1) OBJECT-COMPUTER/SOURCE-COMPUTER changed to reflect Honeywell G-635.

(2) A SPECIAL-NAMES statement is required for the HIS version to define block common areas. Block 59 is used to copy a portion of the DATA DIVISION (LMPDD) for file-to-file LM and for the passing of data between LM and MP in single-file LM. The GCOS system date is used (GETIM) for producing a run date. A special HIS debug statement is used to speed systax error processing when syntax errors are detected in the generated code

(3) An I/O control paragraph is used to define special control techniques to be used in the object program. The APPLY PROCESS AREA clause provides direction to the compiler to generate a fixed logical record processing area in addition to the normal buffers. System standard file format is used for all records.

b. LMLPPG2

(1) The FILE SECTION remains basically the same. LABEL records are standard rather than omitted as in IBM. Data control block information cannot be used from control cards as in IBM and therefore, the BLOCK CONTAINS clause is omitted and default used.

(2) NO LINKAGE SECTION is used in HIS. This entire section is omitted with communication of data handled via the SPECIAL NAMES clause through the use of BLOCK COMMON. The data elements themselves are defined in the WORKING-STORAGE SECTION.

(3) The '-X' redefinition of the data record (e.g., MAJOR-fileA-RECORD-X) have been eliminated from the HIS version. The data records are now defined as 10K characters in length with no OCCURS DEPENDING ON clause.

ENCLOSURE D

(4) Data items defined as COMPUTATONAL in IBM are used as COMPUTATIONAL-1 (COMP-1) under HIS.

(5) Some data items are no longer used in the HIS code and have been eliminated. Specifically, some of the VAL-n used for reading DCB information have been removed.

(6) A MAJOR- and MINOR-PSC-TABLE is generated in the HIS version along with a MAJOR and MINOR-PSC-HOP for use in manipulation of binary control information in the record PSC fields. The TABLES are a replica of the MAJOR and MINOR record control definitions. Special GMAP routines are used to read and write the control information. Data is moved from the TABLE to a record for a READ and from the record to the TABLE prior to a WRITE.

(7) New data elements have been created to correspond with changes in the HIS PROCEDURE DIVISION statements. These changes are minor but do represent a departure from the IBM coding.

(8) In the PROCEDURE DIVISION, the transfer of DCB information has been eliminated as this type of data cannot be taken from control cards as in IBM.

(9) Under GCOS a GO TO DEPENDING ON clause must have at least two paragraph names to branch to. As the generated code often produced only one paragraph, a second paragraph END-RUN was incorporated into all generated GO TO DEPENDING ON clauses. At no time should this second paragraph be branched to and it acts as a dummy paragraph.

c. LMLPWP4

(1) The CONVERT-FIELD paragraph (one call in IBM) requires the use of three calls (depending on previous usage) under GCOS in order to use dynamic loading of programs. If there has been a previous loading of a routine, it will be deleted (paragraph DELETE) before a new routine is loaded. Paragraph LOAD loads the routine and BRANCH directs the logic flow to the loaded routine for execution.

(2) In order to pass parameters between a COBOL calling program and a GMAP subroutine, both a USING and a GIVING statement is required.

(3) Two new GMAP subroutines (RDPSCS and WTPSCS) are used to read and write records larger than the GCOS system standard record size.

ENCLOSURE D

(4) In the WRITE-MAJOR paragraph, GMAP coding has been inserted to expedite the writing of a larger than system standard record.

4. SPECIAL OPERATORS. The Honeywell Information System (HIS) implementation of the MIDMS special operators. There were several problem areas encountered in translating the MIDMS special operators from IBM to HIS. These problems are discussed below:

a. The linkage of subroutines. Since HIS COBOL requires that parameters passed in calling a subroutine be 01 level items, it was necessary to make special provisions for this constraint. Special 01 level items were added to the Data Division of the calling and called program. In the calling program the data to be passed was first moved to the 01 item and then the subroutine was called. Upon return from the subroutine the returned data was moved from a 01 level item to its appropriate storage area.

b. Floating point and binary item. In IBM COBOL binary usage is designated COMPUTATIONAL. In HIS COBOL binary is COMPUTATIONAL-1. In IBM COBOL floating point single precision is COMPUTATIONAL-1 and double precision is COMPUTATIONAL-2. In HIS COBOL floating point is COMPUTATIONAL-2, with the precision (i.e., number of words of storage) determined by the picture clause. IBM COBOL does not permit a picture clause with floating point data whereas HIS COBOL requires a picture clause. Great care must be used in HIS COBOL when designating the picture of a floating point item, since the data will be truncated according to that picture. For example:

```
01 XYZ PIC S9V9 COMP-2.
```

```
      .  
      .  
      .  
COMPUTE XYZ = 5.0 + 5.0  
      .  
      .  
      .
```

will result in 0.0 being stored in XYZ since the leading 1 of 10.0 will have been truncated.

c. The geographic coordinates. Special provisions were required for handling the geographic coordinates. These provisions are described in the documentation of program CRDVAX.

ENCLOSURE D

d. Square root problem. The HIS COBOL compiler in use when the special operators were converted, failed in squaring negative numbers. A GMAP patch was inserted in the programs to perform the squaring operation.

e. System date. The method of obtaining the system date is machine dependent and the appropriate changes were made to the CONVDATE program.

## 5. RETRIEVAL AND OUTPUT.

a. GEN0. GEN0 calls the executable programs GEN1 through GEN6 by using the area label FRANK. The name of the routine to be called is placed in that area and then a subsequent call to LITTLE LINK brings in the appropriate module for execution. Functionally, the Honeywell GEN0 is identical to the IBM GEN0. The only difference is the manner in which the actual calls to the overlays are made. This concludes GEN0 since that program has no GMAP.

### b. GEN1.

(1) Procedure Division. The initial GMAP in this program establishes the proper names for the input and output work files by modifying the contents of the file control block (FICB).

(2) G2-2. If a query card has a slash in column 80, that is an indication that the MIDMS source language deck being read in is in the 360 character set and must be converted to the 635 character set. Upon determination that this conversion is necessary the switch is set which is subsequently checked on following cards to determine the appropriate action for either conversion or non-conversion.

(3) G100. The GMAP in this paragraph sets up the file control block for the output work file. Since the work files are alternated between input and output between the various subroutines, the name for this file must be dynamic within the program. It needs to be set up prior to opening the file. This concludes the GMAP for GEN1.

c. GEN1A. The only GMAP in GEN1A is used to set up the name of the work file in the file control block.

d. GEN2. HOUSE-KEEP--The GMAP instructions in this paragraph store the appropriate names for the input and output work files in the file control block area. That is the only GMAP used in GEN2.

ENCLOSURE D

e. GEN3.

(1) HOUSE-KEEP. The GMAP instructions in this paragraph provide the correct names for the input and output work files to the file control block (F1CB) area.

(2) C-TAB1. The GMAP in this paragraph is used to force alignment with table entries. If the combined length of the argument and function is not a multiple of six, the function length is padded so that it will be a length in full word increments. If the next available location in the constant pool is not on a word boundary, the constant pool area is padded so that the entries in the table will be word aligned. This concludes the GMAP for GEN3.

f. GEN3A.

(1) START. The GMAP in this paragraph is used to set up the appropriate file names for the input and output work files.

(2) CONVX. The GMAP in this paragraph is used to acquire the current date from the system with a MME GETIME. This concludes the GMAP for the GEN3A.

g. GEN4.

(1) START-1. At Line 002721--There are four GMAP instructions in this paragraph and their function is to pick up the file code for the input work file and the file code for the output work file. These file codes are inserted in the file control block generated by COBOL under the name F1FICB and F2FICB.

(2) SEGEN4. The effective address of the record area is stored in the field IRECX. The effective work address of the constant pool is stored in the field ICONX, and multiplied by six to indicate the number of characters.

(3) LINK7. Going into this GMAP routine the field FA1 contains the starting position of the A-field in characters. This value is loaded into the Q register and divided by six to get the number of words. The remainder which is a character position, is found in the A register. The Q register is shifted left and to the remaining value is added in the contents of field FA2, the length of the field under consideration. The Q register is rotated along with the A register so that the final contents of the Q register in bits 0 through 17 will be the work address of the field. The bits 18 through 29 will contain the length of the field and the bits 30 to 35 contain the character position within the

ENCLOSURE D

word of the beginning of that field. The starting address of the B-field in characters is loaded into the Q register and the length of the B-field is loaded into the A register. The AQ registers are temporarily stored in the work area called the WORK1. The A register is now loaded with the OP code, the OP, for the current object statement. If that OP has a value of 5, a transfer is made to the OP5X paragraph. If the OP is equal to 6, control is transferred to the OP6X paragraph. If the OP was 12, the transfer is made to the OP6X paragraph. The Q register still contains the starting position of the B-field. This value is divided by 6 to get the starting position in words of the B-field. The A and Q registers are shifted, added, and rotated as they were for the A-field computation. The result of this manipulation is similar to that which was obtained for the A-field; that is, the starting word of that address is in bits 0 through 17. The length of the field is in bits 18 through 29 and the character position within the word of the starting address of that field is in bits 30 through 35. This brings us to line number 004950. We have already determined that the OP code does not represent a SATISFY, a special operator or a negation of the special operator type statement. This OP code is loaded back into the A register and a check is made for a BETWEEN operator which is an OP4. In this case we will go to OP4X. If it is not an OP4, check is made for an OP10, a negation of a BETWEEN operator in which case we also go to OP4X. If the OP code is not a 4, 5, 6, 10 or 12, we will transfer out of this GMAP string.

(4) OP6X. Coming into this paragraph the Q register contains the starting position of the B-field in characters. Division by 6 provides the word address of the B-field in the Q register and the character position in the A register. Index register 2 is loaded with the word address of the B-field. Index registers 3 and 4 are loaded with the character position within that word. The A register is loaded with the contents of the address contained within index register 2 which is the first word of the B-field. The Q register is loaded with the second word. Subtract 1 from the character position within the word. If it goes minus, the word is properly aligned. Otherwise, rotate these two words left 1 character position and check the alignment again. Continue this until the word is properly aligned. Once aligned we will be down on line 5110. Save the contents of that word in the area W1. Now we will load the A register with the second word from the B-field area and the Q register with the third word from the B-field area. Subtract 1 from the character offset, check for a minus condition which indicates alignment. If we do not have alignment, shift left 1 character position, loop back and check again. Continue doing this until the word is aligned. The A register now contains the 7th through 12th characters of the U-field.



The Q register, after the load operation, contains the first through 6th characters of the B-field. This is stored in the second word of the B-field area. The B1 field of the object statement is loaded with the value 384. Index register 2 is loaded with the address of the second word of the B-field area. This is stored in the B1 field. The character address of the B-field data is now aligned as was done previously so that the word, the length of the field, and the character position within the word are contained in a single word. This value is stored in the B2 field. At the conclusion of the OP6X processing, the address of the name of the special operator is contained in the B1 field and the address, length and offset of the B-field to be operated on by the special operators contained in the B2 field of the object statement. From here a transfer out of the GMAP string is executed.

(5) OP5X. With the SATISFY operator, the computations are similar except that the B3 field has a computed number of mandatory hits on a SATISFY statement and the B4 field will contain the number of candidates, or the number of possible fields for examination on the SATISFY statement. Other than that the packing of the word containing the word address, length of the field, and starting position offset, in characters, is similar to those fields previously computed.

(6) OP4X. The BETWEEN operator B-field computation is similar to that of the other B-fields with one exception. The BETWEEN operator requires two B-fields, therefore, the fields B1 and B2 of the object statement are used to contain word address, length and character offset.

(7) LINK-END. The first half of the GMAP in this string is used to compute addresses within the S-TABLE in the same manner in which they were calculated previously. The S-TABLE as generated contains character positioning or character addresses which must be converted to word addresses and offsets. The second part of this string is initializing control words and blocks prior to opening the first data file.

(8) RECIN. Each time a record is read computations must be performed to determine the starting position of each of the sets. This, of course, is necessary because of the variable nature of the records. A check of the periodic set control words will indicate whether or not there are subsets for each particular set within that record. If there are subsets, the starting position of the first periodic set is the beginning position of the record plus the length of the fixed set. The starting position for the second periodic would be the starting position for the first periodic plus the number of subsets times the length of each subset in the first periodic. This accumulation process is continued for each set within that record.

(9) TEST-FLAG. In this paragraph the periodic sets are checked to determine whether or not switches have been set. If switches were set to indicate that the subsets are flagged, they are reset to, in effect, unflag these subsets.

(10) QSQ-2. In this string of GMAP the S6 vector in the SET-STABLE is being zeroed out or reinitialized. The S1 vector is being moved to the S8 vector. This initialization is performed on advancing from one subquery to another subquery that involves the same record.

(11) UPONEA. The GMAP in this paragraph is used to increment the address of the next subset as the previous subset addresses are incremented by the length of the subset to arrive at the starting position of the following subset, if there is one.

(12) P6. The SATISFIES operator has a plurality of B-fields. This GMAP is used to increment the B1 field of the object statement to indicate the address of the next B-field to be processed.

(13) OP05-3. The current subset address is incremented by the length of the subset to compute the start position of the next subset within the record.

(14) OP06-1. The OP6 statement is special operator. First the A-field address must be calculated and the contents of the A-field must be moved to the INDATA part of the calling sequence for passing to the special operator. The name of the special operator is located and that name is placed in the calling sequence so that the name can be identified as a special operator. The address of the B-field is calculated and the contents of that address and the contents of the B-field are moved to the OUTDATA part of the calling sequence. The length of the B-field, that is the B2 field of the object statement, was destroyed when using the tally instructions for moving the B-field to the OUTDATA portion. This B2 field must be restored prior to continuing.

(15) COMPARE-DUMMY. The logical compares are executed within this section. In the beginning of the GMAP string, the proper A-field address must be established. It needs to be calculated if the A-field is periodic. When this has been done, there is a branch to the AOK GMAP paragraph which indicates that the A-field is correct. At that point the B-field address is calculated, if necessary, to account for periodic positioning of the field within the record. When this is done a branch is made to the BOK paragraph. In the BOK paragraph a determination is made as to whether the fields are alpha or numeric so that the proper type of compare can be made. At this point the actual compare part of the routine is executed and also at this point

ENCLOSURE D

virtually all of the GMAP instructions have comments describing the function taking place. Indicators are set describing the result of the compare.

(16) OP13 SECTION. Starts at Line 011910--The purpose of the GMAP code here is to insert the data that was passed from a special operator into a DEFINE field, a record name or in the constant pool. This is accomplished by picking up the type of receiving field and computing the address of this receiving field. If the field is in a record and is in a periodic set, the address of the field in the active subset is computed. Then the address of the returned data from the special operator is computed. The length of the returned data is inserted in the tally word and that data is transferred left to right from the returned data to the receiving field. If the receiving field is shorter than the returned data, only as much data as fits into the receiving field will be moved. If the receiving field is larger than the returned data, trailing blanks will be added to the receiving field for the remainder of the field. This is accomplished in the following manner. FA4 is the fourth field of the A-field parameters. This field contains zero if, and only if, the receiving field is a defined field or in the constant pool. Otherwise, this field will contain the set number of the data set of the record. If it is periodic set number 2, it will contain a 3. If it is number 4, it will contain a 5, and so on. When this is the case, ST8 vector contains the address of the subset in words and in character position. This is picked up using an index to it, the contents of register 2, which was loaded from FA4 to get to the right subscript. The resolution and words and character position of the address of the field in the particular subset is done by loading A with the location of the subset, loading Q with the location of the subset, adding to A, the starting position of the field in word and character position, adding to Q, the starting position of the word and character position of A-field, that is, the receiving field. Then an ANAQ is performed with the mask LOVE. The mask LOVE is a double word mask that will mask the length in A and the character position and the remainder of the Q, except that the last six bits in Q. This is done so that when a sum of characters from the last six bits of Q reaches a number that is greater than one word, that is 5 characters, then we know that the character position has to be recomputed and 1 has to be added to the word position. For that we use a table called ADDRSS. This table contains the necessary parameters and we use the remainder of Q registers as an index to that table that will give us the precise translation in characters and word modifiers. After this has been done we store the result in FA1, with FA1 becomes a work field now.

ENCLOSURE D

Then if the field were to be a DEFINE field or in the fixed area of the record, these operations are not performed as seen by the second instruction in this code at line 011980. A TZE at that point will branch around this operation and therefore the receiving data in the DEFINE field or the fixed area of the record are much more efficient than receiving in the periodic data. There is an OVERHEAD for receiving in periodic fields. Then we pick up the address of B-data which is the address of the returned data from the special operator. That is actually a tally word, recomputed already in previous steps, that points to the address of the OUT-AREA and AREA-LINKED that is our standard interface with the user subroutines and special operators. One thing has to be modified about this tally. This is the length, the amount of characters that were returned by the special operator. The length is loaded in A register and shifted six bits to the left for proper alignment and then inserted into the TALLY B. After that, using indirect addressing, we proceed in moving the strings of characters and here we are checking to see if an END occurs. If the field is complete, and if the move is complete, and if the field is not filled to the end with data from the special operator, then trailing blanks are added to the receiving field.

(17) FLAG FIRST. At Line 013160--In this paragraph we have a set of GMAP instructions that will place a flag on a number of subsets the user has specified to be flagged unconditionally. Example:

#### FLAG FIRST 5

The first five periodic subsets will receive a nondestructive bit on the first character of the periodic subset sequence number. This nondestructive bit will not modify the digit in any manner whatsoever and at output time or update time the digit will appear just the way it is without a flag. This is accomplished in the following manner. The subset number of the periodic set, the periodic set number, is picked up from FA4. Then the address of the ABM vector is picked up in register 3. That is a work area where we keep track of where we set flags into the records. Then we compute the address of the first subset in the periodic set. A flag is placed upon the first digit of the subset sequence number. The number of flags are counted, the address of the flag, the absolute address in the computer, is saved. In flag A area, 1 is subtracted from the number of subsets to be flagged and if any are remaining to be flagged, we loop back to label OP151 at line 013250. The operations are repeated until all the subsets required are flagged. Then we save the next available location in the FLAG-A area and in the ABM fields and we exit.

(18) FLAG-LAST. Line 013410--The purpose of this paragraph is to, as its name implies, flag n periodic subsets from the end of the subset. First a determination is made if that many subsets exist. If no more subsets than the required number exist, then we transfer back to FLAG-FIRST and we flag as many as there exist in the record. However, if more subsets are present in the record than the user is requiring to flag, we have to compute the address of the last where to start flagging. This is accomplished in the following manner. We pick up the number of subsets in the periodic set. We subtract the number of subsets to be flagged. We store the result in the work area called WORK1. We get the length of subsets in words and characters from ST02 vector for this particular periodic set. We multiply by the number of subsets to be bypassed. We store the result in field FAL. We keep the product of characters only, mask out the rest in Q register. We get the starting address from the first subset. We store zeros in the work area, WORK1. We store character position of the first subset in WORK1. We add to it the product of characters in Q and then we divide by six to obtain words and characters of the address. Insert 1 in length bit 29 of A register. We store the length and the character position in FAL. We get the word amount computed from characters and add to it the product of subset-word-length. Then we add logically the word address of the first subset. We store the result in FAL upper part of the word, that is the address part of FAL. After this has been accomplished, the job remains to place the flags as we have done in FLAG-FIRST with exception now that the address is down at the subset that we have to start with.

(19) UP-A. Line 013860--The GMAP in this paragraph is concerned with computing an address of a field in a periodic set that needs to be flagged if it meets certain requirements of number of subsets to be flagged of low value or high value. That switch is being set in previous paragraph with a 1 if it needs to be flagged high, with a 3 if it needs to be flagged low. This is accomplished by getting the address of the first subset of the particular periodic set and placing that value into SAVEA1 and SAVEB1. Each time the subroutine is executed, the field will be increased by the size of the periodic subset; consequently, the addresses will be pointing to the field of the next periodic subset.

(20) UP-B. Line 013980--It works in conjunction with the code in paragraph UP-A. The only difference between the two is that in this paragraph we are updating the second field that we are comparing with. The address is being saved in area called SAVEB1.

ENCLOSURE D

(21) FLAG34. Line 014180--The three GMAP instructions in this paragraph have a very simple function: to set the second bit of the first digit of the periodic subset sequence number.

(22) UP-SET. Line 014320--The GMAP in this paragraph advances the address of the periodic subset under consideration. The address of the periodic subset is saved in area SAVE-S1. The addresses are done in words and characters of the particular element. The word is placed in the first 18 bits of the field and the character position is placed in the last 6 bits of the particular field.

(23) AGO. Line 014760--The GMAP in this paragraph functionally will perform and decide the logic at the end of a retrieval when a record is qualified. There are several modes of retrieval, which are called search modes. They are normal search, search flag, search all, and search terminate. Normal search means that if the logic is satisfied, we don't go on to anything else further. We want to proceed to provide an answer record or a summary record from this input record. That is done here. Next is the search flag mode. Here we have to place a flag on every periodic subset that participated in selecting this record and proceed to develop the output record. If we are in a SEARCH ALL mode, we would have to place a flag on all the periodic subsets that participated in this selection. At this time hold the record and look forward to see if there are more subsets remaining to be processed. If that is the case, we set up an indicator and rather than producing an answer at this time we return back, go through the logic again, and see if more subsets qualify for the criteria and subquery. The SEARCH TERMINATE on the 635 version of MIDMS behaves in the same manner as SEARCH ALL. There is no such thing as SEARCH TERMINATE. Although the statement is accepted by the compiler we process it as SEARCH ALL on the 360 version.

(24) OP21. Line 015440--This paragraph takes a look at the record after the logic was not satisfied while doing the retrieval. There are several cases in here when the logic was not satisfied because of some fields that were periodic. If that is the case, the GMAP code looks to see if there is another periodic subset that contains those fields that might satisfy the logic. If that is the case, an indicator will be set and the addresses of the active subsets are advanced to that particular new subset and will proceed back to try again the logic. If no subsets are left to be tried, then we proceed to do a final analysis of the record since if we were in a SEARCH ALL mode the record may have been previously selected already for retrieval. It is just this pass of the logic that was not satisfied. The final decision is done in GO-6 paragraph.

(25) LOC-SUB SECTION. Line 016460--This GMAP code is a subroutine in itself and its function is to locate periodic subsets that do carry a flag. This is done so that we can perform secondary logic on the subsets. This is being accomplished in the following manner. When a subset is required for processing for secondary logic, this subroutine is invoked. This subroutine searches through the subsets until it locates one subset that carries the flag. That address, then, is saved and we proceed to compare as being a normal compare. Every time a compare operation of secondary type logic is performed, this subroutine is used to locate those subsets that carry the flag.

(26) SP'61. Line 017090--The GMAP code in this paragraph will build a sort key for the retrieval. The subroutine locates the field within the active subset that participated in the retrieval of the record. The data is picked up from this record and is placed in the sort key. If the sort key is descending, a transformation of data is being performed where the inverted value of the data is placed instead of the actual data. That is, for a 00 value, a 77 value is placed and so on. The sort will perform only an ascending sort, consequently this inversion of data on the character set will produce a descending sequence sort.

(27) SP163. Line 017550--In this paragraph the code does the actual movement of data from the record to the sort key area. This is being accomplished through conventional tally words that were precomputed in the first part of the program when it was loaded.

(28) SP165. Line 017960--The GMAP code in this paragraph reestablishes the periodic set control at the initial value after a SORT FLAGGED operation was performed. That is, producing multiple copies of the record as indicated in the sort flag option and indicating in the set control that one subset exists for that particular periodic set. After all the copies have been created, the periodic set control word is reestablished at the initial value.

(29) SP166. Line 018130--The GMAP code in this paragraph recomputes a periodic set control word for the SORT FLAGGED option of the SORT/MERGE operators. This is being done for each copy of the record that will go on the answer file.

(30) SP14. Line 018500--SP14 is the "SELECT IF FLD-A-HITS n" statement. This GMAP code looks into the particular periodic set and counts to see if there was a number of subsets flagged as required in the SELECT statement. If so, it accepts the record for further processing. If not, it rejects the record as not meeting the logic criteria.



(31) OUT-WRITE. Line 019090--The GMAP code in this paragraph picks up, scans all the periodic subsets, and turns the flag off where they are on after they have been put out as a copy in the SORT FLAG option.

(32) OP-SUMMARY SECTION. Line 019380--The GMAP code in this paragraph computes the address of the input record and of the output record. NOTE: The address of the output record is the address of the logical records in the output buffer. Then it moves the record from the input buffer to the output buffer through repeat-double instructions until all the data is transferred to the output area.

(33) SUM-OPEN. Line 019770--There are two GMAP instructions in this paragraph and their function is to pick up the file code for the summary file to be created for this subquery and place it into the file control block word minus 4 called 12 FICB definition that was done in the file section of this program.

(34) SHIFT-R SECTION. Line 019870--The GMAP code in this paragraph computes the address of the periodic set control word of the periodic set that the SORT FLAG/MERGE FLAG option is exercised. After the location of the periodic set control word, it inserts the address of the periodic subset that is under consideration.

h. GEN4A. GEN4A is wholly contained within GEN4 and will not be separately identified.

i. GEN5. GEN5 contains no GMAP and therefore will not be included in this set of documentation.

j. GEN5A. START-PROC--The GMAP in this paragraph is used to assign the correct name to the work file. This name is stored in the file control block (FICB). This is the only GMAP used within GEN5A.

k. GEN6.

(1) START-PROC. Line 038900--Two GMAP instructions (LOAD Q WITH DDIN and STORE Q IN F1FICB-4,03). These instructions pick up the file code of input the work file and place it in the file control block word minus 4 (two characters in the last two digits of the field).

(2) GET-RIT2D. Line 057800--In here we have one GMAP instruction which is TSX1 to ENCODE. This instruction transfers control to subroutine ENCODE and returns from the subroutine back to the instruction MOVE N TO J. The ENCODE routine has the function of packing the statement and placing it into the STATE (N) vector.



(3) LINK-EDIT. Line 060000--Two GMAP instructions here where we pick up the file code for the output data from the field called IN-DATA and we place it in the RVFICB-4. That permits us to write a variable file as output from report generator.

(4) LOOP-A. Line 061700--Two GMAP instructions here. We pick up the file code of the SOURCE DIRECT data file and we place it in the file control block I2FICB.

(5) LOOP-B. Line 063700--The GMAP code in this paragraph has the function of determining if the operation was table lookup or not. If it were to be a table lookup, which is operation code 40, then the pointers are saved in YESGO vector and NOGO vector.

(6) LOOP-C. Line 064700--The GMAP code in this paragraph finds the location in the constant pool where the convert routine is placed and converts that location to words and computes the absolute address of the table in memory. That address is stored in the NOGO field of the vector.

(7) LOOP-LOAD. Line 066300--The GMAP code in this paragraph will compute the address of a table that was just loaded in core and places it into the constant pool. After the data has been placed in there the pointers are saved in the vectors for a later reference.

(8) LOOP-E2. Line 068900--The GMAP that starts at line number 069200 has a function of precomputing the addresses in word and character position of the periodic set controls and file to be processed. Later on the references will be done more efficiently and will not have to be computed each time the records need to be referenced.

(9) STEP8-RET. Line 077700--There are three GMAP instructions in this paragraph. Their function is to compute the address of the answer record and through a mask to save the segment number of the record just read in. This is done so to know when the record has been completely read in core it would have the segment number 99.

(10) STEP9-1. Line 081100--The function of this paragraph is to zero out the W vector. The W vector is a vector parallel to the instructions generated by the compiler. Each instruction in sequence created by the compiler has a location in this vector and this vector is used as a switch to indicate that this instruction has been executed once and need not be executed again while looping through the periodic subsets. It contains the pointer and compare instructions where the next instruction is to be performed. The code in this paragraph has no other functions except to initialize it when a new record is considered for processing.

(11) CALARASI. Line 084800--The GMAP code in this paragraph is adequately explained in the program. The instructions are explained in this paragraph logically; and furthermore, it has COBOL statements equivalent if the decision is made to replace the GMAP with COBOL code.

(12) OP41-42. Line 092600--The GMAP in this paragraph will compute the subscripts for A-field and B-field. The subscripts are provided in characters and the 635 unit of reference is the word. Consequently the characters are being converted to word addresses and character position between those words. Those addresses are being saved and further processed for address modification on the periodic fields or defined elements, matrix, vectors, etc.

(13) OP13-C. Line 097700--The GMAP code in this paragraph will pick up the character position in the record and convert it to word address and character position within those words for A-field and B-field of all operations. Then those addresses are adjusted with the address of the particular area that the field resides in, i.e., record address, constant pool address, and so on. These addresses are used and execution is actual address for further processing.

(14) LOAD-A. Line 103100--The function of the GMAP code in this paragraph is to place in A-field the data from a work area as a result of a computation.

(15) LOAD-B. Line 104200--The function of the GMAP code in this paragraph is to pick up the result of a computation from temporary word area WORK2 and place it in B-field of the operation.

(16) OP0710. Line 105400--The code in this paragraph is used to obtain data from A-field and place it into a work area after it has been converted to binary data and to obtain data from the B-field and place it in the second temporary work area after it has been converted to binary data. This subroutine is utilized in obtaining numeric fields from the constant pool and from the record and to perform operations like add, subtract, multiply, divide, and so on.

(17) X40. Line 110402--This paragraph contains the table lookup subroutine. It is properly documented in the COBOL program, functionally and otherwise. Each line of code is documented.

(18) X11. Line 113400--The GMAP in this paragraph has the function of moving the A-field to the B-field. The fields could be alpha-numeric or numeric. The code is properly documented in the listing with comments, functionally and otherwise, for each instruction.

(19) COMPN. Line 117100--The code in this paragraph performs a compare between two fields which we shall call A-field and B-field. These fields can be alpha-numeric fields or numeric fields. Depending on the type of fields, different type of compares will take place. If the fields are alpha-numeric, they are assumed to be of equal length; otherwise, the compiler would not have let them through. The compare is done logically left to right, bit for bit, and the greater logical compare, that field will be indicated as being greater. Within the numeric fields the compare is performed in a different fashion, blanks which are an Octal 20 and are consequently much greater than any number are treated in this compare as zeros. The bits that form a blank are masked out. Furthermore, the fields do not have to be of equal length. Here we are doing a true algebraic compare. The one exception is that the blanks first are converted to zero if they exist in the field. The operation is performed in the following manner. Leading zeros are provided for the shorter fields. We are looking for the greater magnitude first. Once we determine which is the greater magnitude of the two fields that we are comparing, field-A and field-B, then we proceed in analyzing the sign of the numbers. If the sign is negative, the field with a greater magnitude in the greater field. If negative signs are found within digits other than the last digit of the fields, they are ignored by being masked out and they are not considered in the compare. Alpha-numeric data encountered in numeric fields cannot be compared properly unless they are of equal length and the fields are equal. Otherwise the sign bit will be wiped out and the characters are transformed before any comparison takes place. The addresses of A-field and B-field to be compared are precomputed during the previous paragraph that we have identified as OP13-C at line 097700.

(20) X12. Line 127800--The GMAP code in this paragraph has the function of moving the input data to AREALINKED. That data will be passed through the convert routine the user has provided under the operation (OP12), which is the CMOVE. It does not do anything but move the data from the constant pool or the input record to AREALINKED.

(21) OP1A. Line 134500--The GMAP code in this paragraph has a function of moving data from the output area where the user has built it into the buffer of the output record that the user wants to write. OP18, operation 18, means it is a write. At the command "WRITE" the code will be executed and its places into the output buffer the data that needs to be written out. The operations are done through repeat double instructions for efficiency.

ENCLOSURE D

(22) X26. Line 137700--The GMAP in this paragraph, rather long, functionally places segments of the variable set into the output specification of the users. Due to the nature of the variable data certain analysis has to be performed on the text. This analysis is as follows. When possible, words may not be split but are spilled over in the repeated line. Therefore, backward tracking for blanks has to be checked, and, where possible, the entire word will be placed into the output area. When not possible, this word will be split. The code is properly documented in the program listings.

(23) OP24. Line 147500--The GMAP code in this paragraph computes the address of the subset to start with and to process in the output the last n periodic subsets as required in the LINE subparagraph of the LINE SECTION. This is done by counting backwards how many subsets there are and by saving that point as the first subset.

(24) OP25-1. Line 149800--The GMAP code in this paragraph is similar to the code in paragraph OP24 at line 147500. One exception--the counting backwards is done only on flagged subsets rather than any subsets. It is used with SET N LAST M FLAGGED.

(25) OP29. Line 152800--The GMAP code in this paragraph has the function to decode the information about periodic sets in the input record. This decoding is being done for every record read in and it is placed in the set vectors, SET1, SET3, SET5, and so on. What actually takes place in here is picking up information from periodic set control from the input records and placing them in vectors in order using the periodic set number as a subscript to the vector set. An insertion of the value takes place. Also a conversion takes place. From BCD information they are being converted to binary values to be used more efficiently.

(26) X30. Line 158002--This subroutine is the edit move code. It is properly documented in the program.

(27) X33. Line 159400--This paragraph computes the address of the program number to be performed. That program number is used as a subscript in the table where the list of programs are loaded. That address is picked up and inserted and used as a subscript in executing the instruction where the program is located. The program number is converted from BCD to binary value to be used as a true subscript in COBOL in further processing.

ENCLOSURE D

(28, LINE-START. Line 164800--The GMAP code in this paragraph initializes the vectors to zeros as their initial values for starting to process a LINE SECTION in the OUTPUT module. Repeat double instructions are being used.

(29) X36. Line 172700--The GMAP code in this paragraph is performing IF CHANGE operation. The code is properly documented in the program listings.

(30) X37. Line 174900--The code in here performs the IF COMPLETE operation. The code is properly defined in the program listings.

(31) ENCODE. Line 182502--The code in this paragraph, functionally, takes the expanded statement vector and packs it. The packing rules are properly documented in the COBOL listing indicating what bits occupy what.

(32) DECODE. Line 182534--In this paragraph the operation is a reverse of the ENCODE operation that is started at line 182502. Again, the code is properly defined and can be easily seen cross referenced to the ENCODE documentation.

1. GEN6A. GEN6A is wholly contained within GEN6 and will not be separately identified.